Paper 028-31

Squeezing Information out of Data

William C. Murphy
Howard M. Proskin and Associates, Inc., Rochester, NY

ABSTRACT

The SAS® system offers a superb set of tools for analyzing data. However, the data you receive from a client may not be as clean as you would like. Character data may contain comments with embedded control characters that makes print out impossible. There may be stray text in an otherwise numeric variable making summarization difficult. Important information may be contained in variables that contain both numeric and character components. Prior to version 9, the undesired information was removed out of the variables by a combination of programming and the use of the COMPRESS function with a long list of characters for the function's second argument. However, the new COMPRESS function has a third argument that allows the user to remove or retain selected types of data. This argument greatly mitigates the need for extensive programming to remove undesired characters from a text. Further, the second argument of the COMPRESS function can be greatly shorten or even eliminated with the proper use of this third argument. We illustrate how you may have extracted information from your data in the past and how the new COMPRESS function offers a better alternative. Indeed, we will show that the COMPRESS function with the third argument is a very powerful tool for squeezing out the information that you need for your databases and reports.

INTRODUCTION

Working for a small statistical consulting firm in western New York has allowed me to work will data from a variety of clients. We receive these data in many forms: flat files, spreadsheets, SAS data sets, and various proprietary formats. However, one thing is common among all these data sources: the data is rarely useable in the form presented to us. We usually do extensive restructuring and formatting to make the data more compatible with our internal programs. Furthermore, since the SAS system is our main database and analysis tool, all the data regardless of original source must ultimately be converted into one or more SAS data sets. But this rebuilding of the database does not address all of the problems. All too often the values contained in the data are prone to crash our programs. For example, the data set creation process may have lead to the insertion of control characters in text fields that make displaying and listing the information impossible. Researchers also seem all too ready to insert comments in what would otherwise be a numeric field making analysis of the numbers difficult. Finally, in an apparent attempt to save 'space', information containing both numeric and character components is combined in the same variable. This information needs to be separated before further processing can occur.

In the past, these issues were address by use of the COMPRESS function often with extensive second arguments. This function in turn may have been imbedded in a long program to achieve our data cleaning objective. Fortunately, the COMPRESS function now has a third argument that facilitates the squeezing of information out of the data.

COMPRESS FUNCTION

The COMPRESS function is a routine available in the DATA step that allows you to remove unwanted characters from a string variable. The syntax is simple:

```
Slim = compress(Bloated,Arg2,Arg3);
```

This function will remove characters from the string variable Bloated and put the result in the string variable Slim. The second argument (*Arg2*) is a text string or a variable that contains a list of the characters that you want the function to remove from Bloated. Up until version 9 of the SAS System, this is all you had to work with. With the new version, the third argument (*Arg3*) was introduced. This argument adds a great deal of power to the function. Like the second argument, the third argument is a text string or a variable containing a text string. However, this text string is composed of a list of letters that instructs the COMPRESS function to perform in a particular way. For example, the letter 'a' contained in the third argument would lead to the removal of all alphabetic characters from the variable Bloated.

In the following sections, we will illustrate how this new feature of the COMPRESS function can be used to clean data and increase program efficiency.

EMBEDDED CONTROL CHARACTERS

The data has been processed and all that you have to do is print out the comments. Since these comments are often longer than one line, you use PROC REPORT with the FLOW option for printing. You run you program and the first few pages of your listing looks fine. But then the listing abruptly stops or goes to the next page right in the middle of a comment. What happened? You look at your data in the SAS VIEWTABLE and notice that the comment where the listing breaks down contains some strange characters ('▮▮▮'). These are control characters that the computer system uses for such things as line breaks, tabs, and form feeds. Somehow these characters were inserted into the data set in the creation process, but they must be removed for the comment listing to come out as expected. Since these control characters make up the first 32 characters in the byte coding sequence, we can readily removed them from the comments by writing a DATA step with the following statements:

```
do i=0 to 31;
       Comment=compress(Comment,byte(i));
       end;
```

where the BYTE function produces the designated character from the coding sequence. This DO loop goes through the first 32 characters of the sequence, and each time through the loop, a control character is removed from the variable Comment. If you now try to print out your comments, everything will be fine.

The above is the solution to the problem of embedded control characters if you ignore the third argument in the COMPRESS function. Using the third argument, however, we can rewrite the entire solution to our problem in one line:

```
Comment=compress(Comment, ,'c');
```

The third argument 'c' tells the COMPRESS function to get rid of all the control characters in the comment. Since these are the only characters that we wish to remove from the data, a space can be inserted for the second argument. Using the third argument of COMPRESS is a far more elegant solution to the problem.

ALMOST NUMERIC DATA

You just received the lab data for your project. Among other things it includes the red blood cell count (RBC). In order to check the data you run a PROC MEANS on RBC and get in the log:

```
ERROR: Variable RBC in list does not match type prescribed for this list.
```

So you open up the lab data with the SAS VIEWTABLE to investigate. You scan down the RBC column. Everything looks like a reasonable number and then you encounter a 'N/A'. A little further down the column, you run into a 'Low'. You keep scanning the column and you come up with a plethora a text comments: 'na', 'NAV', 'HIGH', 'normal', 'not done', 'dropped sample', *etc*. The lab investigators decided to use what should have been a numeric field as a comment pad. To take the mean we must first process the RBC:

```
RBC=upcase(RBC);
RBC=compress(RBC,'ABCDEFGHIJKLMNOPQRSTUVWXYZ/');
RBCnum=input(RBC,10.);
```

Since the comments are of mixed case, we use the first line of code to make sure everything is in the same case. The second line of code has the complete alphabet along with the slash written out for the second argument. The last line of code converts the RBC to a number. If we now run our PROC MEANS on the RBCnum, we get our summary statistics.

But this could have been done much simpler if we used the third argument in COMPRESS in conjunction with the second argument. The first two line of code could be replaced with

```
RBC=compress(RBC, '/','a');
```

The second argument of the COMPRESS function still contains the slash, but the alphabet has been removed. A third argument of 'a' tells the COMPRESS function to remove any letters of the alphabet from RBC. Any letter means either case so the UPCASE function is no longer necessary. If you had wanted to remove the letters of a

specific case, you could have used the 'u' for upper case or the 'l' for lower case. Again the use of the third argument of COMPRESS, in conjunction with the use of the second argument, provides a very concise solution.

NUMBER-CHARACTER MIX

Quite often we deal with dental data, with various measurements given for each tooth. We prefer our data sets with one observation per tooth, but we are usually provided with one observation per subject. The analysis variables have a name descriptive of the quantity and a suffix number that indicates the tooth (*e.g.* Decay21 is the decay score for tooth number 21). If we had four consecutive tooth scores, we could readily convert the data into the desired form by writing:

```
array xxx{4} Decay11 Decay12 Decay13 Decay14;
do i=1 to 4;
       Tooth=i+10;
       Score=xxx[i];
       output;
       end;
```

But what if we did not have consecutive tooth numbers? If we wanted to use the same program for different data sets with different numbers of Decay variables, how would we then identify the tooth? Using the power of COMPRESS, these problems can readily be solved:

```
array xxx{*} Decay: ;
do i=1 to dim(xxx);
       Tooth=compress(vname(xxx[i]),,'kd');
       Score=xxx[i];
       output;
       end;
```

where we use the asterisk (*) to let the SAS system count the members of the array, the colon (:) to indicate all variables starting with the string 'Decay' (Murphy 2004), and the DIM function to provide the upper count of the DO loop. Since we can no longer get the tooth identification from the index variable, we first acquire the variable name with the function VNAME. Then we use the COMPRESS function to keep ('k' in third argument) only the numbers ('d', for digits, in the third argument) in the variable name. Again the COMPRESS function provides a solution to a complex problem.

We could even expand on the above example to deal with different types measurements. The teeth in the dental study could also be scored for their root recession (Root1, Root2…) and their fluoride content (Fluoride1, Fluoride2…) or dozens of other variables. So we could rewrite the previous example to be

```
array xxx{*} Decay: Root: Fluoride: ;
do i=1 to dim(xxx);
       Measure=compress(vname(xxx[i]),,'d');
       Tooth=compress(vname(xxx[i]),,'kd');
       Score=xxx[i];
       output;
       end;
```

where we have added another COMPRESS statement to determine the type of measure by dropping the digits in the variable name. Thus use of the COMPRESS function and a small modification to the program allows us to deal with all of the different measurements in the study.

When processing the COMPRESS function in a loop, the SAS system has another modifier for the third argument that can improve the speed of execution. We could rewrite the COMPRESS line in the preceding example as

```
       Tooth=compress(vname(xxx[i]),,'okd');
```

where the 'o' in the third argument tells the SAS system to process the second and third arguments of the COMPRESS function only once, rather that every iteration of the loop. This modifier can save time in large loops but only works if the second and third arguments do not change.

PUTTING THE SQUEEZE ON MACROS

By now you are probably thinking that the third argument of the COMPRESS function is a very useful feature, even without me pointing out there are many other useful letters that can be used in the third argument (*e.g.*, 'p' for punctuation marks, 's' for space characters, and 'g' for graphics characters). But can it help you with your macro coding?

Alas, the macro compression function %CMPRES does not even have a second argument, let along a third. It can only be used to get rid of blanks from your variable. However, with the use of the %SYSFUNC function, we can still use the power of COMPRESS on macro variables:

```
%let MacVar=%sysfunc(compress(&MacVar,,d));
```

where all of the numbers are removed from the value of the macro variable &MacVar. It should be noted that the third argument is not put in quotes, since macros expect text strings.

CONCLUSIONS

The COMPRESS function has increased functionally with the use of its third argument. You can readily extract or keep a variety of characters from a text string. For cleaning and parsing your data, the COMPRESS function offers a quick, efficient solution. It is ideal for squeezing the information out of the data.

REFERENCES

Murphy, W.C. (2004), "Colonoscopy for the SAS® Programmer", *Proceedings of the Twenty-Ninth Annual SAS® Users Group International Conference*, SAS Institute Inc., Cary, NC, paper 054-29 (url: http://www2.sas.com/proceedings/sugi29/054-29.pdf).

SAS Institute Inc. (2002-2003), "Functions and CALL Routines: COMPRESS Function". *SAS OnLineDoc® 9*. SAS Institute Inc., Cary, NC. (url: http://v9doc.sas.com/91doc/docMainpage.jsp).

CONTACT INFORMATION

Your comments and questions are values and encouraged. Contact the author at

William C. Murphy
Howard M. Proskin & Associates, Inc.
300 Red Creek Dr., Suite 330
Rochester, NY 14623
Phone 585-359-2420
FAX 585-359-0465
Email wmurphy@hmproskin.com or wcmurphy@usa.net
Web www.hmproskin.com