PharmaSUG2012 – Paper CC17

Efficiently Trim Character Variable Lengths to Fit Data, Reduce Dataset Size

Wayne Zhong, Octagon Research Solutions Inc.

ABSTRACT

In environments where saving datasets with the (COMPRESS=yes) option is not permitted, unnecessarily large datasets clog up hard drives and increase processing time, costing patience and efficiency alike. Somewhere between the extremes of simply defaulting character variables to 200 characters or spending time to manually set lengths - risking truncation when data updates - there is a happy middle ground to automate the process: always let the length be as short as it can without truncation.

INTRODUCTION

Suppose a character variable length is 200 and the value is “H”, a SAS® dataset will store this data as “H” followed by 199 spaces. If the longest value stored in this variable has length 1, then this variable will take up 200 times the data space absolutely needed. Taking all the variables in a dataset into consideration, if a 70% reduction in dataset size can be achieved, this equates to a 70% reduction in wait time for a hard drive or network transfer and much faster program execution.

The FDA conducted a study on 1000 datasets, trimming the character variables lengths resulted in an average size reduction of 70%.

MACRO GOALS

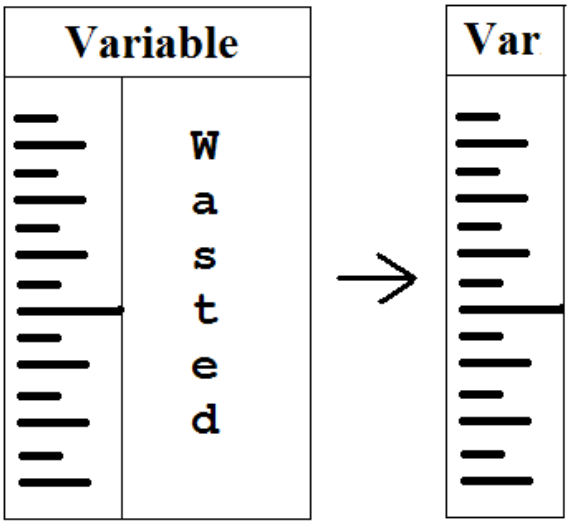
1. reduce character var lengths

2. do not modify numeric vars

3. keep variable order, labels, formats, and dataset label the same

4. macro should be simple to use and does its job efficiently

As this is a paper for Coders, two coding methods will be presented. A comparison serves to showcase different approaches to the same problem as well as efficiency considerations.



DATA STEP METHOD

```
%macro trim1(indsn=,outdsn=);
```

%TRIM1 will serve as the first method. While only one SAS procedure is used – a DATA STEP – a number of macro variables and one helper macro is needed to define the characteristics of the input dataset: the metadata, so that the program will essentially write itself.

First, the number of variables, the number of observations, and the label of the input dataset are saved to macro variables. &nvars helps loop through variables, &nobs identifies empty datasets, and &label labels the output dataset.

```
%let dsid=%sysfunc(open(&indsn,i));
%let nvars=%sysfunc(attrn(&dsid,nvars));
%let nobs=%sysfunc(attrn(&dsid,nobs));
%let label=%sysfunc(attrc(&dsid,label));
%let rc=%sysfunc(close(&dsid));
```

Expand &label value to be more usage friendly later on.

```
%if %length(&label)>0 %then %let label= label=&label;
```

The following helper macro makes the name, type (N or C), and length for the (&num)th variable available in global macro variables.

```
%macro help(num);
%global var type len;
%let dsid=%sysfunc(open(&indsn,i));
%let var=%sysfunc(varname(&dsid,&num));
%let type=%sysfunc(vartype(&dsid,&num));
%let len=%sysfunc(varlen(&dsid,&num));
%let rc=%sysfunc(close(&dsid));
%mend help;
```

Turning off the variable length check warning is something that should only be done here, where lengths will be intentionally reduced.

```
options varlenchk=nowarn;
```

The following code doesn't work for a zero-observation-dataset, so we do those separately.

```
%if &nobs>0 %then %do;
```

The goal of the DATA STEP is to create a LENGTH statement which the input dataset can be set against. The retained variables _1 - _&nvars will be used to store the minimum length for each variable, _ALL is used to create the length statement when the last record has been read.

```
data temp(compress=Y);
    set &indsn end=last;
    retain _1-_&nvars 1;
    length _all $10000;
```

When looping through and accessing each variable's metadata using the %help macro, character variables are checked record by record to see if the minimum length needs a bump. Numeric variables are simply set once to the metadata length.

```
    %do i=1 %to &nvars;
        %help(&i);
        %if &type=C %then  &i=max(_&i,length(&var));
        %else if _n_=1 then  &i=&len;
        ;
    %end;
```

To build the length statement at the very end, loop through each variable and concatenate the variable name, a $ sign if character, and the minimum length to the variable _ALL

```
    if last then
        do;
            %do i=1 %to &nvars;
                %help(&i);
                 all=cat(strip( all)," &var ",
                         %if &type=C %then '$',;
                         strip(put(_&i,best.)));
            %end;
```

Cheat here to remove a DATA STEP from the code by using CALL EXECUTE, which will execute the string as code after the current step ends. Dropping temporary variables is easy since they start with "_".

```
            call execute("data &outdsn(&label); length "||strip(_all)||
                      '; set temp; run;');
            drop _: ;
        end;
run;
%end;
```

If the input dataset is empty, this macro makes no changes. Don't forget to turn VARLENCHK back on!

```
%else %do;
data &outdsn(&label);
    set &indsn;
run;
%end;

options varlenchk=warn;
%mend trim1;
```

Final note for the first coding method: the COMPRESS=Y option used on the TEMP dataset greatly reduces dataset size, allowing faster write and future access time. A compressed dataset can be much smaller than even a trimmed dataset, however few environments allow datasets to be stored in the compressed state. Still, it is possible to add this option to an OPTIONS statement at the start of a program to compress all datasets created and experience reduced processing time, just remember to turn it off when outputting datasets.

SQL METHOD

```
%macro trim2(indsn=, outdsn=);
```

Rather than try to invent a better way to access metadata, we reuse proven code.

```
%let dsid=%sysfunc(open(&indsn,i));
%let nvars=%sysfunc(attrn(&dsid,nvars));
%let nobs=%sysfunc(attrn(&dsid,nobs));
%let label=%sysfunc(attrc(&dsid,label));
%let rc=%sysfunc(close(&dsid));

%if %length(&label)>0 %then %let label= label=&label;

%macro help(num);
%global var type len;
%let dsid=%sysfunc(open(&indsn,i));
%let var=%sysfunc(varname(&dsid,&num));
%let type=%sysfunc(vartype(&dsid,&num));
%let len=%sysfunc(varlen(&dsid,&num));
%let rc=%sysfunc(close(&dsid));
%mend help;

options varlenchk=nowarn;
```

The macro variable &all is used to build the LENGTH statement using a loop through all variables. %help is used to get variable metadata, a MAX(LENGTH(&var)) calculation give the minimum required length for character variables while the metadata variable length is used for numeric variables. This information is concatenated to &all.

```
%let all=;
proc sql noprint;
    %do i=1 %to &nvars;
        %help(&i);
        %if &type=C %then
            %do;
                select cats('$',max(length(&var))) into :len
                    from &indsn(keep=&var);
                %if &len=$. %then %let len=$1;
            %end;
        %let all=&all &var &len;
    %end;
quit;
```

Two additional notes, for zero observation datasets the character length is set to 1, hence those datasets are not processed separately. The KEEP statement used on the input dataset is a must, otherwise processing time is increased by a factor proportional to the number of variables.

```
data &outdsn(&label);
    length &all;
    set &indsn;
run;
options varlenchk=warn;

%mend trim2;
```

COMPARISON

So, which way is better? The DATA STEP method is "one" data step, needed temporary variables and separate code for zero-observation-datasets. On the other hand, the SQL method uses macro variables, and the MAX(LENGTH()) function simplifies the code.

The best way to judge is by benchmarking the code. The platform used is a laptop with a 1.66GB SAS dataset stored on its hard drive (networked sources can have large variations in performance). To keep the test times reliable, a complete system restart was performed between runs so no data will be stored in memory.

As this is Coder's Corner, the following code serves as the stopwatch, the executing code is placed between start and stop. 4 different tests were conducted.

```
%let start=%sysfunc(time());

%let stop=%sysfunc(time());
%put Runtime = %sysfunc(round(%sysevalf(&stop-&start),.01)) seconds;
```

(1) create work version of source data: 212 seconds

```
data adlb;
    set test.adlb;
run;
```

(2) run DATA STEP trim method: 159 seconds

```
%trim1(indsn=test.adlb, outdsn=adlb_t);
```

(3) run PROC SQL trim method: 205 seconds

```
%trim2(indsn=test.adlb, outdsn=adlb_t);
```

The source dataset started at 1.66GB, the trimmed version is 0.92GB.

(4) create work version of trimmed source data: 120 seconds

```
data adlb_t;
    set test.adlb_t;
run;
```

The results seem surprising, how can trimming a dataset (all output is to work library) take 25% less time than simply creating a work version of the same dataset? The following are educated guesses:

The overwhelming majority of execution time in the code we are running is to read and write datasets. For the DATA STEP method (2), reading in the source dataset takes the same time as (1), however the TEMP output dataset is COMPRESSED (70% reduction in size) taking less time to write. The time saved covers the time to write the trimmed dataset (45% reduction in size). For this last step, the read time is eliminated because the dataset is still stored in computer memory.

The breakdown in time is estimated as:

(1) read dataset (100%); write dataset (100%);
(2) read dataset (100%); write dataset (30%); write dataset (55%);
(3) read dataset (90%); read dataset (100%); write dataset (55%);
(4) read dataset (55%); write dataset (55%)

The SQL method performs almost two full reads, the first only reads character variables to determine the minimum variable length. However, to create the trimmed dataset another full read is necessary because the previous step did not create a compressed work dataset nor commit the entire source dataset to memory. Still, this method managed to best (1) by reducing the write time needed.

CONCLUSION

Not only do trimmed datasets save storage space, transfer and execution time in the long run, the macro that does it can be programmed to be faster than a simple SET statement DATA STEP. The code provided is simplified, please consider spending additional time: 1. to make sure macro variables are compatible with local and global macro environments 2. expand input and output parameters to handle a list of datasets and an entire library at once.

CONTACT INFORMATION:

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

Author Name:	Wayne Zhong
Company:	Octagon Research Solutions Inc.
Address:	585 East Swedesford Road, Wayne, PA
Work Phone:	(610) 535-6500 x5535
Email:	wzhong@octagonresearch.com
Web:	www.octagonresearch.com

APPENDIX

```
%macro trim1(indsn=,outdsn=);

%let dsid=%sysfunc(open(&indsn,i));
%let nvars=%sysfunc(attrn(&dsid,nvars));
%let nobs=%sysfunc(attrn(&dsid,nobs));
%let label=%sysfunc(attrc(&dsid,label));
%let rc=%sysfunc(close(&dsid));

%if %length(&label)>0 %then %let label= label=&label;

%macro help(num);
%global var type len;
%let dsid=%sysfunc(open(&indsn,i));
%let var=%sysfunc(varname(&dsid,&num));
%let type=%sysfunc(vartype(&dsid,&num));
%let len=%sysfunc(varlen(&dsid,&num));
%let rc=%sysfunc(close(&dsid));
%mend help;

options varlenchk=nowarn;

%if &nobs>0 %then %do;

data temp(compress=Y);
    set &indsn end=last;
    length _all $10000;
    retain _1-_&nvars 1;
    %do i=1 %to &nvars;
        %help(&i);
        %if &type=C %then  &i=max( &i,length(&var));
        %else if _n_=1 then  &i=&len;
        ;
    %end;
    if last then
        do;
            %do i=1 %to &nvars;
                %help(&i);
                _all=cat(strip(_all)," &var ",%if &type=C %then '$',;
strip(put(_&i,best.)));
            %end;
            call execute("data &outdsn(&label); length "||strip(_all)||'; set temp;
run;');
            drop _: ;
        end;
run;

%end;

%else %do;

data &outdsn(&label);
    set &indsn;
run;

%end;

options varlenchk=warn;

%mend trim1;
```

```
%macro trim2(indsn=, outdsn=);

%let dsid=%sysfunc(open(&indsn,i));
%let nvars=%sysfunc(attrn(&dsid,nvars));
%let nobs=%sysfunc(attrn(&dsid,nobs));
%let label=%sysfunc(attrc(&dsid,label));
%let rc=%sysfunc(close(&dsid));

%if %length(&label)>0 %then %let label= label=&label;

%macro help(num);
%global var type len;
%let dsid=%sysfunc(open(&indsn,i));
%let var=%sysfunc(varname(&dsid,&num));
%let type=%sysfunc(vartype(&dsid,&num));
%let len=%sysfunc(varlen(&dsid,&num));
%let rc=%sysfunc(close(&dsid));
%mend help;

options varlenchk=nowarn;

%let all=;
proc sql noprint;
    %do i=1 %to &nvars;
        %help(&i);
        %if &type=C %then
            %do;
                select cats('$',max(length(&var))) into :len from &indsn(keep=&var);
                %if &len=$. %then %let len=$1;
            %end;
        %let all=&all &var &len;
    %end;
quit;

data &outdsn(&label);
    length &all;
    set &indsn;
run;

options varlenchk=warn;

%mend trim2;
```