Paper CT05

Dynamic Testing For Empty Datasets

Timothy Lachlan-cope, AstraZeneca, Cambridge, England

ABSTRACT

Dynamic data requires a dynamic approach. Notes, warnings and errors regarding the existence of data and datasets should not be ignored, although can occur given a changing stream of data. When developing code we should be paying close attention to the log, however, false positives can hide real messages of concern or disrupt automatization for no reason. A developer should have this in mind when creating robust code and consider when it might be appropriate to apply techniques to conditionally select objects.

Within this presentation I will detail several techniques for checking datasets including a null data step and the SASHELP/Dictionary location to check observations, and then detail how this can be combined with SAS® macro language to create a truly dynamic program with less pesky notes, warnings and errors.

INTRODUCTION

When creating SAS® programs at a study level, it can be expected that data may change in terms of depth and breadth. A robust program should be able to handle different situations whilst retaining a log clean of errors, warnings and notes of concern. SAS offers a variety of tools for identifying these issues. Which can be combined with a number of techniques to enhance program flexibility in managing different situations. This paper will explore a several different tools and look to expand the readers tool box.

Three states of missing data will be examined, with a few techniques highlighted for each. The paper will progress as follows:

- Missing datasets
  - EXIST
  - Dictionary tables and SASHELP views
- Missing data
  - _null_ data step metadata
  - Dictionary tables and SASHELP views
- Missing columns
  - Varnum
  - Dictionary tables and SASHELP views

This paper will address three specific scenarios. Firstly, there is neither a dataset nor metadata. Secondly, metadata exists but lacks associated data. Finally, a dataset exists with data, however, it may lack the desired columns for interaction.

MISSING DATASET

The first situation to be examined is that in which a datasets existence is possible but not guaranteed. For example, a data cut may or may not include a dataset specifying important protocol deviations that should ideally be merged into a wider dataset. In this case the dataset should be included in a `PROC SQL` or data step merge (or any other step/procedure that uses only if it exists.

EXIST

The first solution is that of the `exist` function. A BASE SAS provided function, that can provide the existence, or not, of a dataset. This function serves multiple purposes, with a simple implementation being demonstrated below.

```
%if %sysfunc(exist(raw.AESI)) %then %do;
  Conditional processing
%end;
%else %if…
```

The `exist` function gives a Boolean response, if the dataset exists. This can be inserted into macro processing using `%sysfunc` or `%gsysfunc` to have exist evaluated at the macro compile stage. The exist function provides a very simple method of testing if a dataset exists. Whilst shown in macro language here it could also be used in a regular data step for conditional processing.

DICTIONARY TABLES AND SASHELP VIEWS

Dictionary tables and SASHELP views are a repository of metadata about tables or views that have been stored. This repository doesn't just look at permeant file location as directed to by libnames, but also keeps account of metadata from the work location. This information can be pulled into a developers program and then used in a number of ways. For a full look into Dictionary tables and views please see Lafler (2010). One example is:

```
proc sql noprint;
  select '"'!!strip(memname)!!'"'
  into :data_set separated by ','
  from dictionary.table
  where libname = "libname";
quit;
```

The example code provides a macro variable that can be used in further conditional processing. This produces a list of datasets that do exist, and hence if they aren't on this list they don't exist. This particular example produces a macro variable that is using quotations and comma separated so is suitable for data step processing, but if this was to be used for further macro processing quotations should not be used. Alternatively if a list of datasets wants to be run conditional on their existence in a library, this SQL step could be used to create a dataset and then feed this dataset into a call execute data step, for a full look into call execute please read Usov, (2014).

MISSING DATA

Unlike the first scenario covered, it is possible that a dataset has been created but it contains no data. This can also cause issues when integrating into a wider program. Therefore, once again, defensive programming should accommodate this potential scenario.

NULL DATA STEP

The null data step method loads the metadata from a single data step which can then be used within the data step without loading any of that datasets data. This makes it particularly efficient for finding the number of observations in a single dataset. An example of the code is as follows:

```
data _null_;
  if 0 then set &ds_name. Nobs = n;
  call symputx ("obs",n);
run;
%if &obs >0 %then %do;
…
%end;
```

The key to the efficiency of this method is using `if 0` to not load the data, the statement is always false given the value 0. However, the metadata is still loaded so `Nobs = n` can still be used for further processing later in the program. In this case the value of n is taken passed into `call symputx` to create a macro variable that could be used in conditional macro processing.

DICTIONARY TABLES AND SASHELP VIEWS

Dictionary tables and SASHELP views can also be used in determining the quantity of data contained within a dataset. For this example a null data step will be used with the SASHELP VTABLE view.

```
data _null_;
  set sashelp.vtable (where = (libname = "MAPS" and memname = "AFRICA"));
  call symputx ("chk_obs",nobs);
run;
```

The example code only needs to pass the value of the column nobs to a macro variable so a null data step may be used. On the set statement the where dataset option is important to only identify the specific dataset to be checked. Next the `call symputx` is used to pass the value into a macro variable called "chk_obs". This can then be used for conditional macro processing.

MISSING COLUMNS

When free text is collected in data, it is common to break it into small sections, resulting in an undermined number of columns. Given a changing stream of data, such as new observations in an overall set or a defined subset of data, the amount of columns that reflect a free text box may not be optimal and can vary. Missing columns can also appear when transposing data, A complete dataset may have a full set of values that are being transposed upon, but when a dataset is subset you may lose some of these values, and hence columns. So then when referring to columns post transpose you may get unsightly notes such as: NOTE: Character values have been converted to numeric values.

VARNUM

The VARNUM function returns a value corresponding to the column number given a variable name, with a value of 0 if a column does not exist. This can therefore be used in conditional processing. When a variable it returns a value greater than 1, giving a positive response for the if statement and hence starting conditional processing.

```
data _null_;
  vnum=0;
  dsid = open('maps.africa');
  if dsid then
    do;
      call symputx('x_exist',varnum(dsid,'x'));
      dsid = close(dsid);
    end;
run;
%if &=x_exist %then do;
…
%end;
```

In this example the value resulting from Varnum has been entered into a macro variable, &x_exist, which can then be passed into further macro processing. Both `if-then` and `%if-%then` logic will take a value above 0 as true and start processing hence any valid value for a variable number will result in processing. For an implementation that goes straight into macro processing please check Carpenter, A. (2017).

DICTIONARY TABLES AND SASHELP VIEWS

For missing columns a new dictionary or SASHELP view is required. Unlike the previous situations columns or vcolumn will be used.

```
proc sql noprint;
  select name
  into :column separated by ','
  from DICTIONARY.columns
  where libname = "MAPS" and memname = "AFRICA";
quit;
```

In this example macro variable called “column” is created that holds a list of columns within the SAS dataset MAPS.AFRICA. This macro variable can then be used to subset processing to only include columns that exist (or perhaps create dummy columns for columns that should exist!) Another use of the dictionary tables and SASHELP views is to, instead of creating a macro variable, create a dataset that could be used with call execute to loop through values within that dataset.

CONCLUSION

This paper has given a selection of possible implementations to make code more dynamic given three situations. Each have their own unique set of advantages and disadvantages and the solution used should reflect the developers specific set of challenges. Further reading should be taken within the SAS help documentation (see further reading section) to better understand the functions and techniques suggested within this paper, each could certainly be used in other unique and interesting ways to help solve new and exciting problems.

A developer should also consider how best to implement conditional processing post checking. This will certainly give some bare on which the best implementation at the “check” stage as detailed in this paper. This decision is likely to be led by the specific situation the developer finds them self in. As such this paper does not try to prescribe what to do next as the possibilities are substantial!

REFERENCES

Carpenter, A. (2017). *Building Intelligent Macros: Using Metadata Functions with the SAS® Macro Language*. [online] Available at: https://www.lexjansen.com/pharmasug/2017/TT/PharmaSUG-2017-TT02.pdf [Accessed 2 Oct. 2023].

Lafler, K. (2010). *Exploring DICTIONARY Tables and SASHELP Views*. [online] Available at: https://support.sas.com/resources/papers/proceedings10/155-2010.pdf [Accessed 2 Oct. 2023].

Usov, A. (2014) *Call Execute: Let Your Program Run Your Macro*, lexjansen. Available at: https://www.lexjansen.com/phuse/2014/cc/CC06.pdf [Accessed 2 Oct 2023].

APPENDIX

Appendix 1:
Code:

```
data example1;
  var1 = 1;
  var2 = 2;
run;

data example2;
  var1 = 1;
  var2 = 2;
run;

proc sql noprint;
  create table tables as
  select *
  from DICTIONARY.tables
  ;
quit;
Proc sql noprint;
  Select '"'!!strip(memname)!!'"'
  Into :data_set separated by ','
  From DICTIONARY.tables
  Where libname = "WORK";
Quit;
%put &=data_set ;
```

Log:

DATA_SET="EXAMPLE1","EXAMPLE2","TABLES"

Appendix 2:

Code:

```
Proc sql noprint;
  Select name
  Into :column separated by ','
  From DICTIONARY.columns
  Where libname = "MAPS" and memname = "AFRICA";
Quit;

%put &=column ;
```

Log:

```
COLUMN=CONT,ID,SEGMENT,DENSITY,X,Y,LAT,LONG
```

Appendix 3:

```
data _null_;
  vnum=0;
  dsid = open('maps.africa');
  if dsid then
    do;
      call symputx('x_exist',varnum(dsid,'x'));
      dsid = close(dsid);
    end;
run;
%put &=x_exist ;
```

Log:

X_EXIST=5

ACKNOWLEDGMENTS

I would like to thank Dr Sofia, Sarah, Korak and all my mentors and colleagues at AstraZeneca.

READING

https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/lefunctionsref/p1thpm9d6f5itan1c37zh8uz273z.htm

https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/mcrolref/p1o13d7wb2zfcnn19s5ssl2zdxvi.htm#n0x97dl6jtc5ujn1iowwm5bp8p4m

https://support.sas.com/resources/papers/proceedings10/155-2010.pdf

https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/sqlproc/n02s19q65mw08gn140bwfdh7spx7.htm

https://documentation.sas.com/doc/en/sclref/9.4/n1d3q4nv946nedn1781klx83awnk.htm

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:
Timothy Lachlan-cope
AstraZeneca