Paper TU06

The Dataset Diet – How to transform short and 'fat' into long and 'thin'

Kathryn Wright, Oxford Pharmaceutical Sciences, UK

ABSTRACT

What do you do when you are given a dataset with one observation per subject and hundreds of variables? The dataset is not particularly usable in this format. It is generally more useful to convert the data into several observations per subject with fewer variables. There are a few methods in SAS® that can be used to achieve this effect. This paper will look into simple DO LOOPs, DO LOOPs with SQL and MACROS, ARRAYS and PROC TRANSPOSE.

INTRODUCTION

WHAT IS A 'FAT' DATASET?

A 'fat' dataset has few observations per subject and many variables. For example:

Subjid	Adverse1	Related1	Severity1	Adverse2	Related2	Severity2
0001	cough	yes	mild	flu	no	severe

WHAT IS 'THIN' DATASET?

A 'thin' dataset has multiple observations per subjects with fewer variables per observation. For example:

Subjid	Adverse	Related	Severity
0001	cough	yes	mild
0001	flu	no	severe

WHY CONVERT 'FAT' TO 'THIN'?

There are many different reasons why you may wish to convert a 'fat' dataset to a 'thin' one. The primary reason for converting this particular dataset from 'fat' to 'thin' was to conform to a company standard and make use of pre-existing macros for creating outputs for reporting purposes.

This paper is based on a particular dataset containing all the data for a specific study with 50 observations and 1075 variables. Initial review of the dataset indicated that it was extremely cumbersome to work with in this format. This was for a multitude of reasons:

- It took 5 minutes just to scroll from one side of the dataset to the other!
- It was very time consuming to search the 'fat' dataset for specific results for a subject.
- Variables were listed within the dataset in collection order not logical order (for example, adverse events could be scattered throughout)
- Any manipulation required to produce summary tables and listings took a lot more SAS code than the equivalent 'thin' dataset.

METHODS FOR CONVERTING

Whilst working with this dataset, methods were assessed that could be employed to make the conversion of the dataset from 'fat' to 'thin'. These included:

- Simple DO LOOPs
- DO LOOPs with SQL
- ARRAYS
- PROC TRANSPOSE

SIMPLE DO LOOPS

A simple DO LOOP can be used where the variable names are sequential. For example an adverse events dataset where the variables are called ADVERSE1, ADVERSE2, ADVERSE3 etc. For example

SUBJID	AE1	ADVERSE1	RELATED1	SEVERITY1	AE2	ADVERSE2	RELATED2	SEVERITY2
0001	1	cough	yes	mild	1	flu	no	severe

The 'thin' version of this dataset is:

SUBJID	ADVERSE	RELATED	SEVERITY
0001	cough	yes	mild
0001	flu	no	severe

The code that can be used to convert this dataset to the 'thin' alternative is

```
%macro adverse;

 data adverse (keep = subjid adverse related severity);
   set rawdata.adverse;
    %do a = 1 %to 2;
      if ae&a = 1 then do;
        adverse = adverse&a;
        related = related&a;
        severity = severity&a;
        output;
       end;
    %end;
run;

%mend adverse;
%adverse;
```

In order to use this code you need to know the number of variables of each type that there are. For example, in this case there are only 2 adverse variables – ADVERSE1, ADVERSE2 therefore the DO LOOP is:

```
%do a = 1 %to 2;
```

If there were 5 adverse events, the DO LOOP would read:

```
%do a = 1 %to 5;
```

For this particular study, there were 30 adverse event observations for all subjects: however, not all of them were populated. In the 'thin' dataset only the observations where an event was recorded were required to be kept. For example, if subject 1 only had 2 adverse events the 'thin' dataset would only have 2 observations for that subject. In the dataset if an adverse event was recorded at the first visit the variable AE1 would have a value of 1. The line

```
if ae&a = 1 then do;
```

ensures that variables are only kept if they are populated.

This method is useful if the variables are sequential numerically.

DO LOOPS WITH PROC CONTENTS AND SQL

If the variables are non-sequential the simple DO LOOP method will not work. However DO LOOPs can still be used by utilizing PROC CONTENTS and some SQL prior to the DO LOOP. An example of this is a vital signs dataset. Here is an example of a 'fat' vital signs dataset.

SUBJID	SBP	DBP	PULSE	HEIGHT	WEIGHT
0001	120	80	75	175	80

The 'thin' dataset would look like

SUBJID	TEST	RESULT
0001	SBP	120
0001	DBP	80
0001	Pulse	75
0001	Height	175
0001	Weight	80

The code used to produce this output is:

```
%macro efficacy;
 proc contents data=rawdata._all_ noprint out=conts (keep=name
       where =(upcase(name)in 'SBP', 'DBP', 'WEIGHT', 'HEIGHT', 'PULSE')));
  run;

 proc sql noprint;
       select count (distinct name) into: totnames
       from conts;

       select distinct name into: name1 - : name&totnames
       from conts;
 quit;



       data scrfdata.vitals (keep = subjid test result);
         set rawdata.rawdata;
              %do loop=1 %to &totnames;
                 test = "&&name&loop";
                 result = &&name&loop;
                 output;
              %end;
      run;


%mend efficacy;

%efficacy;
```

PROC CONTENTS produces a dataset with a list of all the datasets and the variables in a specified library. In this case a KEEP statement has been used to only save the NAME variable. NAME is the variable name produced by PROC CONTENTS that lists the variable names in the datasets. As the dataset I was working with had other information as well as vital signs data a WHERE= clause has been used to specify the variable names that I wish to retain. NAME is the variable name that the PROC CONTENTS output uses for the list of variables.

The SQL section produces MACRO variables. The first macro variable, TOTNAMES, is the total number of variables in the dataset produced by PROC CONTENTS. The second section creates MACRO variables for each variable name found in the NAME variable of the CONTS dataset. For example SBP would be held in macro variable NAME1.

The first MACRO variable, TOTNAMES, can be used to specify how many times the DO LOOP has to re-iterate. This is useful when you do not know how many variables you have.

The DO LOOP is then executed to map the new variables to the MACRO variables. The variable TEST will take the value of "&&name&loop". During the first iteration of this loop this resolves to &name1 which is equal to 'SBP'. Therefore TEST has a value of 'SBP'. RESULT also resolves to SBP but as &&name&loop is not in quotes the variable takes the value of the variable SBP. For example, if SBP has a value of 120 in the raw dataset RESULT would become equal to 120.

By using an output statement within the DO LOOP, each iteration will produce a new observation. The KEEP statement then ensures that only the variables that are required are kept.

This method is useful when all variables are required and they are not sequential numerically.

ARRAYS

Both the above methods utilized SAS procedures and functions that are used frequently; the next method that was investigated was ARRAYS. Being less familiar with ARRAYS, this was not an obvious choice initially. Closer inspection of this method indicates that converting 'fat' to 'thin' datasets is well suited to the use of ARRAYS and are a very useful way to manipulate data.

ARRAY 1 (ALTERNATIVE TO SIMPLE DO LOOPS)

Firstly, ARRAYS were used to create a simple adverse events dataset from the following format:

SUBJID	AE1	ADVERSE1	RELATED1	SEVERITY1	AE2	ADVERSE2	RELATED2	SEVERITY2
0001	1	cough	yes	mild	1	flu	no	severe

```
data adverse;
   set rawdata.rawdata;
      array adverse{2} adverse1- adverse2;
      array related{2} related1 - related2;
      array severity{2} severity1 - severity2;
        %do i = 1 to 2;
              adverse = adverse{i};
              related = related{i};
              severity = severity{i};
              output;
        %end;
run;
```

When using ARRAYS, it is required to set up an array for each type of data. In this case, one array was set up for ADVERSE, RELATED and SEVERITY. In this example, it is necessary to specify how many variables are required in each array – only 2 in this case. This information is found in the {} brackets. With sequential variables, first and the last can be specified, for example, ADVERSE1-ADVERSE6.

The DO LOOP is required to map the new variables to the array variables. During the first iteration of this loop, ADVERSE will have the value of ADVERSE1, with a value of 'cough'.

As previously mentioned using an output statement within the DO LOOP, each iteration will produce a new observation. The KEEP statement ensures that only the variables that are required in the new 'thin' dataset are kept.

The 'thin' dataset produced by this code would be:

SUBJID	ADVERSE	RELATED	SEVERITY
0001	cough	yes	mild
0001	flu	no	severe

ARRAY 2 (ALTERNATIVE TO DO LOOPS WITH PROC CONTENTS AND SQL)

The methodology explored in the above example was extended to covert the vital signs dataset.

As before, the 'fat' dataset:

SUBJID	SBP	DBP	PULSE	HEIGHT	WEIGHT
0001	120	80	75	175	80

Only minor changes were required to the code:

```
  array vital {5} variable--another_variable;
```

The ARRAY will include all variables between VARIABLE and ANOTHER_VARIABLE, in the order they are found in the original dataset, as long as '--' is used to separate the two variable names.

The following code would transform the dataset above to the 'thin' alternative:

SUBJID	TEST	RESULT
0001	SBP	120
0001	DBP	80
0001	Pulse	75
0001	Height	175
0001	Weight	80

```
data vitals;
  set rawdata.rawdata;
   array vital{5} sbp--pulse;

    do i = 1 to 5;
              test = "array{i}";
              result = array{i};
               output;
   end;
```

```
run;
```

If all the variables that are required are adjacent to each other in the dataset, it is not necessary to specify the number of variables in the array. Using an * in the {} brackets specifies that all the variables between SBP and PULSE should be included:

```
array vital{*} sbp--pulse;
```

The ARRAY would include all variables found within that range.

ARRAYS are useful to transform datasets where multiple output variables are required.

PROC TRANSPOSE

Although it may seem like the obvious choice for converting a 'fat' dataset to a 'thin' dataset, PROC TRANSPOSE was actually the last method explored. Previous usage had been to use PROC TRANSPOSE to convert 'thin' to 'fat'.

In fact it is very straightforward. The code below can be used to convert a 'fat' vital signs dataset to a 'thin' one.

```
proc transpose data=rawdata.rawdata out = vitals (keep = subjid col1);
        var sbp dbp height weight pulse;
        by subjid;
run;
```

This code is very simple and can be used in this form, if there are only a small number of variables to transpose.

However, there are a large number of vital signs collected in a dataset, a similar piece of code to that indicated above for arrays can be used:

```
proc transpose data=rawdata.rawdata out = vitals (keep = subjid _name_ col1);
        var sbp -- pulse;
        by subjid;
run;
```

This will include all variables between SBP and PULSE.

This method is only useful if all the variables between two specific variables within the dataset are transposed into one variable in the 'thin' dataset. For example:

SUBJID	SBP	DBP	PULSE	HEIGHT	WEIGHT
0001	120	80	75	175	80

Would be converted to:

SUBJID	_NAME_	COL1
0001	SBP	120
0001	DBP	80
0001	Pulse	75
0001	Height	175
0001	Weight	80

However, this method gives undesired results on the adverse event dataset:

Input dataset:

SUBJID	AE1	ADVERSE1	RELATED1	SEVERITY1	AE2	ADVERSE2	RELATED2	SEVERITY2
0001	1	cough	yes	mild	1	flu	no	severe

The output dataset would look like:

SUBJID	_NAME_	COL1
0001	Ae1	1
0001	Adverse1	Cough
0001	Related1	Yes
0001	Severity1	mild
0001	Ae2	1
0001	Adverse2	flu
0001	Related2	no
0001	Severity2	severe

This is not in the required format, as ADVERSE, RELATED and SEVERITY were needed to be separate variables.

It is worth noting that when using PROC TRANSPOSE, the dataset must be sorted by the BY variable(s) before it can be transposed correctly.

CONCLUSION

Having been faced with a 'fat' dataset that needed converted into many standard 'thin' datasets, a lot of time was spent looking into the options available to perform these transformations. There are many different methods available for use in SAS and they are all useful in specific circumstances.

When variable names are sequential, a simple DO LOOP can be the easiest way to make 'fat' into 'thin'. If the variables are not sequential, this method will not work on its own. In these circumstances, PROC CONTENTS and SQL can be utilized to create sequential MACRO variables which can then be used in the DO LOOP. This method is good if all the variables in the dataset are being used. If however, as in this case, only some of the variables are required it can be less efficient to list the variables required in the WHERE= statement.

ARRAYS can be useful in both the scenarios mentioned above, especially where the output dataset requires the data to be transformed into more than one variable. Beware - arrays will only work where all the variables within an array are the same type, you cannot mix character and numeric variables.

PROC TRANSPOSE is the final option and it is the simplest and requires the shortest amount of code as long as there are only a few variables or the variables are adjacent to each other. If a lot of variables are required and they are spread across the dataset or more than one variable is required in the output dataset PROC TRANSPOSE is not the best method.

In summary, if there are a lot of adjacent variables that need to be converted, PROC TRANSPOSE should be the method of choice. Where the variable names are sequential and not all the variables are required to be kept, a DO LOOP is the simplest method to use. In any situations where more than one variable is required in the output dataset ARRAYS are the most effective method.

CONTACT INFORMATION

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

Author Name	Kathryn Wright
Company	Oxford Pharmaceutical Sciences
Address	The Stables, 114 Preston Crowmarsh, Wallingford
City / Postcode	Oxon, OX10 6SL
Work Phone:	+44 1491 821679
Fax:	+44 8704 580729
Email:	Kathryn.wright@ops-web.com
Web:	www.ops-web.com