

Long verses Wide – A Different Way of Looking at SAS® Coding

Richard Goldsmith, ASG, New York, NY

ABSTRACT

SAS data sets tend to be created wide, i.e. a single observation has many variables. This intuitive approach makes the most effective use of data storage space and allows the simplest approach to writing SAS code. However, what will happen if a long data set is used instead of a wide data set? Long data sets can increase the flexibility of the underlying data structure and can aid in the construction of reusable code when used with the SAS macro facility. A simple example showing how a long data set and the SAS macro facility can be used to construct a generalized SAS program with PROC GLM will be presented. This approach demonstrates the use of the SAS macro indirect referencing facility and shows how indirect referencing can be used to write reusable code.

Introduction

A typical SAS data set represents what I will refer to as a 'wide' structure, i.e. there are 'many' variables and 'few' observations. 'Many' and 'few' are relative terms, I am sure that everyone has at least heard about the three variable, five zillion observations data set that somebody somewhere uses. Figure 1 shows an example of a traditional data set, containing data for Martian height (in cm), weight (in kg), and hair color.

```

Data htw1;
  infile cards;
  input sex $ height weigh
        hair $;
cards;
m 31 7.8 red
f 32 8.3 red
. . . . .
f 45 13.6 green
f 46 14.2 green
;

```

Figure 1. Traditional 'wide' data set.

However, what will happen if the structure of Figure 1 is altered to produce a data set that contains an observation for each of the variables in a traditional data set? This reformulation is shown in Figure 2.

```

data htw1;
  infile cards;
  input @1 id $2.
        @4 var_is $6.
        @11 val_is $6.
        @18 val_fmt $1.;
  count = _n_;
cards;
01 sex      m      c
01 height  31      n
01 weight  7.8     n
01 hair    red     c
02 sex      f      c
02 height  32      n
02 weight  8.3     n
02 hair    red     c
. . . . .
;

```

Figure 2. Nontraditional 'long' data set.

The reformulated data set contains several metavariables (for this paper, a metavariable is defined as a variable that contains information about a data variable). In the *htwt_1* data set, *var_is* and *var_fmt* are examples of metavariables – *var_is* contains the name of a variable and *var_fmt* contains information on the type (numeric or character) of the variable. The variable *val_is* contains the value of a variable as a character. Making *val_is* a character variable allows both numeric and character values to be stored in a single variable. This saves space, but adds the requirement of additional processing because an INPUT statement is required for each numeric variable. The variable *id* is an identifier that allows multiple instances of data to be combined into a single instance (observation). The variable *count* is introduced to retain the relative position of variables. Its use will become evident in later steps.

A Solution

Obviously data set *htwt_1* cannot be used in SAS without some additional modifications. One way to modify *htwt_1* would be through the use of PROC TRANSPOSE, however, I prefer to use the macro facility to restructure this data set, since it provides more flexibility. For this restructuring, four macros are required:

- *obsnvars* – determines the number of observations and variables in any data set (see page 10 of the draft for Macro Facility Enhancements for Release 6.09E and Release 6.11)

- mak_str – creates an 'array' of variables, var_1 .. var_n.
- mak_wid – transforms a 'long' data set to a 'wide' data set
- renam – renames variables in a data set

In the initial step the data set htwt_1 is created.

The next step is to create a data set, unique, that contains a single observation for each variable name in htwt_1. The contents of the data set unique are shown in Figures 3a and 3b.

OBS	VAR_IS	COUNT	VAL_FMT
1	HAIR	4	C
2	HEIGHT	2	N
3	SEX	1	C
4	WEIGHT	3	C

Figure 3a. Data set UNIQUE (before sorting by variable count).

OBS	VAR_IS	COUNT	VAL_FMT
1	SEX	1	C
2	HEIGHT	2	N
3	WEIGHT	3	N
4	HAIR	4	C

Figure 3b. Data set UNIQUE (after sorting by variable count).

The SAS macro obsnvars is then called with the first argument (&dsn) set to unique. This macro will return the arguments nvars, which will be set to 3, and nobs, which will be set to 4. The macro variables str_is1 and str_is2 are then created using the number of observations found in &nobs. The macro variable &str_is1 will resolve to var_1 var_2 var_3 var_4 (this is the reason there is no semicolon after the &var.&i in the macro mak_str, if there were a semicolon, the resulting string would be var_1; var_2; var_3; var_4;), while the macro variable &str_is2 will resolve to fmt_1 fmt_2 fmt_3 fmt_4. Note that we have created array-like variables in &str_is1 and &str_is2. Data set null_1 is then created to load values into the macro 'arrays'. Selected variables found in the null_1 data set are shown in Figures 4a and 4b.

OBS	VAR	VAL_IS
1	VAR_1	SEX
2	VAR_2	HEIGHT
3	VAR_3	WEIGHT
4	VAR_4	HAIR

Figure 4a. Partial contents of data set null_1

OBS	FMT	VAL_IS
1	FMT_1	C
2	FMT_2	N
3	FMT_3	N
4	FMT_4	C

Figure 4b. Partial contents of data set null_1

After the null_1 data set has been processed, the macro variables will resolve as follows:

Macro	Value
&val_1	SEX
&val_2	HEIGHT
&val_3	WEIGHT
&val_4	HAIR
&fmt_1	C
&fmt_2	N
&fmt_3	N
&fmt_4	C

Figure 5. Macro Variable Resolution.

Now that the macro variables are set, a single, traditional data set can be created using the macro mak_wide. This macro has two arguments, an input data set, dsn_in, and an output data set, dsn_out. In the example, the htwt_1 data set is the input data set. It is used to create the output data set, wide. Wide is created by first creating a series of data sets, data1 through data&nobs, in this case data1, data2, data3, and data4. Note that a data set is created for each variable. Each of the dataN data sets contains three variables, id, val_is, and var_n. Variables id and val_is are obtained from the original htwt_1 data set, while var_n is created as either a numeric variable, using the input function with a best. format, or as a character variable.

OBS	ID	VAR_1
1	01	m
2	02	f
...	...	...
22	22	f

Figure 5a. Data set data1.

OBS	ID	VAR_2
1	01	31
2	02	32
...	...	...
22	22	46

Figure 5b. Data set data2.

OBS	ID	VAR_3
1	01	7.8
2	02	8.3
...	...	...
22	22	14.2

Figure 5c. Data set data3

OBS	ID	VAR_4
1	01	rd
2	02	red
...	...	...
22	22	green

Figure 5d. Data set data1.

In the actual code you will note that both the variable that determines the format and the variable used in the where clause to subset the data set are indirect macro variables, that is they begin with &&. When SAS parses a macro, it resolves all macro references and places the resulting pseudo code in the macro catalog, usually work.sasmacr. If an indirect macro is found, SAS will first resolve all macro references, && resolving to &, then make a second pass through the code to resolve any remaining macro variables. Thus when nob=1, &&var.&i will first be resolved to &var1 (first pass), which is then resolved to sex (second pass). Likewise, &&fmt.&i resolves first to &fmt1 (first pass), then to c (second pass). Each data set is then sorted using the variable id. After the four data data sets have been created, the output data set, wide, is created by merging the data data sets together. A label is created for each variable using values obtained from the variable var_is in the hwt1 data set: as shown in Figure 6. Note that data set wide has the structure of a traditional SAS data set.

Variable	Label
var_1	SEX
var_2	HEIGHT
var_3	WEIGHT
var_4	HAIR

Figure 6. Created Labels for Variables.

Now all that is left to do is to print the data set.

The final step in the generic program is to run a simple PROC GLM. It is assumed that the first numeric variable will be the dependent variable, and that all subsequent numeric variables are dependent variables. All character variables are assumed to be class variables -- we want to see if sex or hair color cause differences in the height-weight model. The data set unique is first sorted by decreasing val_fmt (in order to place the numeric variables at the 'top' of the data set) and count to produce the data set shown in Figure 7.

OBS	VAR_IS	COUNT	VAL_FMT
1	HEIGHT	2	N
2	WEIGHT	3	N
3	SEX	1	C
4	HAIR	4	C

Figure 7. Data set UNIQUE (after sorting by descending val_fmt count).

The unique data set is then used to build the model and class statements for the glm procedure in data set null_2. Two variables, model and class are created that have a length of 40 characters each. Since we will be building these variables using multiple observations, the SAS retain command is

used. Two 'place holder' variables are created and initialized to 1. The contents of the variables are shown in Figures 8a and 8b.

OBS	MODEL	P_MODEL
1	HEIGHT =	10
2	HEIGHT = WEIGHT	17
3	HEIGHT = WEIGHT SEX	20
4	HEIGHT = WEIGHT SEX HAIR	25

Figure 8a. Contents of variables model and p_model in data set null_2.

OBS	CLASS	P_CLASS
1		1
2		1
3	SEX	4
4	SEX HAIR	9

Figure 8b. Contents of variables class and p_class in data set null_2.

Note that the use of the SAS substr function on the left hand side of the equivalence sign causes the string on the right hand side of the equivalence sign to be placed into the variable on the left hand side of the equation. When all observations in the unique data set are processed, the SAS routine symput is used to place the contents of the model and class variables in macro variables of the same name.

The rename macro is then invoked to change the name of the var_n variables to something more meaningful. In this case var_1 is renamed to SEX, var_2 is renamed to HEIGHT, var_3 is renamed to WEIGHT, and var_4 is renamed to HAIR using the datasets procedure. The glm procedure is then run using the following code (shown with macros resolved):

```
proc glm;
  class sex hair;
  model height = weight sex hair / ss3;
run;
quit;
```

If interactions, contrasts, estimates, etc. are needed they can be added through the inclusion of additional metavariables, through the use of programming rules or through a combination of metavariables and programming rules -- but, hey, that's probably a topic for another paper!

The approach presented in the paper gives a general solution that may prove to be useful. Once established a general, reusable set of code is in existence that can be applied to a variety of data without making any coding changes. This flexibility, however, come with costs -- data structure

complexity is somewhat increased and program complexity is greatly increased.

Want a copy of the program? E-mail your request to agrag@ibm.net.

Acknowledgements

I would like to thank Ms. Sheri Michener, Senior Biometrician, Animal Health Group, Pfizer Inc for the seeds of this idea and much critical discussion during its evolution, and Ms. Alice Cheng, Statistical Data Analyst Clinical Data Operations, Pfizer Inc, for her comments on this paper.

```
options pageno=1;
/* obs and vars macro */
/* finds number of observations (nobs) and variables (nvars) */
/* in any dataset (dsn) */
macro obsnvars(dsn,nvars,nobs);
  %global &nvars &nobs;
  /* open the dataset */
  %let dsid = %sysfunc(open(&dsn));
  %if (&dsid) %then
    %do;
      /* use sysfunc to find the number of obs */
      %let &nobs = %sysfunc(attrn(&dsid,NOBS));
      /* use sysfunc to find the number of variables */
      %let &nvars = %sysfunc(attrn(&dsid,NVARS));
      /* close the dataset */
      %let rc = %sysfunc(close(&dsid));
    %end;
  %else
    %put Open for dataset &dsn failed -- %sysfunc(sysmsg());
  %mend obsnvars;

/* mak_str macro */
/* creates psuedo string of variables, varn, numbered from 1 to n */
macro mak_str(var,n);
  %do i = 1 %to &n;
    &var.&i
  %end;
%mend;

/* mak_wide macro */
/* creates wide dataset from long dataset */
macro mak_wide(dsn_in,dsn_out);
  /* create a data set for each observation in data set unique */
  %do i = 1 %to &nobs;
    data data&i (keep=id var_&i);
      set &dsn_in (keep=id var_is val_is where=(var_is="&&var_&i"));
      %if (&&fmt_&i=n) %then %str(var_&i = input(val_is,best.););
      %else %str(var_&i = val_is;);
    run;
    proc sort;
      by id;
    run;
  %end;
  data &dsn_out;
    merge
      %do i = 1 %to &nobs;
        data&i
      %end;
  ;
  by id;
  label
    %do i = 1 %to &nobs;
      var_&i = "&&var_&i"
    %end;
  ;
run;
%mend mak_wide;

/* renam macro */
/* renames variables within a dataset */
macro renam(dsn);
  /* use datasets to rename variables */
  proc datasets lib=work nolist;
```

```

        modify &dsn;
        rename
            %do i = 1 %to &nobs;
                var_&i = &&var_&i
            %end;
    ;
run;
quit;
%mend renam;

/* traditional data set */
data htw1;
    infile cards;
    input sex $ height weight hair $;
cards;
m 31 7.8 red
f 46 14.2 green
;
/* nontraditional 'long' data set */
data htw1_l;
    infile cards;
    retain count 0;
    input @1 id $2. @4 var_is $6. @11 val_is $6. @18 val_fmt $1.;
    count = count + 1;
cards;
01 sex      m      c
01 height   31     n
01 weight   7.8    n
01 hair     red    c
22 sex      f      c
22 height   46     n
22 weight   14.2   n
22 hair     green  c
;
run;

/* find unique occurrences of a value in variable var_is */
proc sort data=htwt_1 (keep=var_is count val_fmt) out=unique nodupkey;
    by var_is;
run;
/* reorder data set by instance of occurrence */
proc sort data=unique;
    by count;
run;
/* find number of variables and observations */
%obsnvars(unique,nvars,nobs);
/* create pseudo array */
%let str_is1=%mak_str(var_,&nobs);
%let str_is2=%mak_str(fmt_,&nobs);
/* want pseudo array to be global macro variable */
%global &str_is1 &str_is2;
/* model and class statements for glm */
%global model class;
/* load pseudo array with values */
data null ;
    set unique;
    length var fmt $8;
    /* create variable name */
    var = compress('var_' || put(_n_,3.));
    /* load pseudo array element with a variable name */
    call symput(var,trim(left(var_is)));
    /* create format name */
    fmt = compress('fmt_' || put(_n_,3.));
    /* load pseudo array element with a format */
    call symput(fmt,trim(left(val_fmt)));
run;
/* create wide (traditional) data set from long data set */
%mak_wide(htwt_1,wide);
/* print the data set */
title "Data Values for Wide";
proc print data=wide split='';
    var id &str_is1;
run;
quit;
/* create glm commands */
/* create glm commands */
proc sort data=unique;

```

```

by descending val_fmt count;
run;
data null ;
set unique end=done;
by descending val_fmt;
length model class $40;
retain model class ' '
      p_model p_class 1
;
/* if the format is numeric, n, put in model statement */
if (val_fmt='n') then
do;
  if (_n_=1) then
  do;
    len = length(var_is);
    substr(model,1,len) = var_is;
    len = len + 1;
    substr(model,len,1) = '=';
    p_model = len + 1;
  end;
  else
  do;
    len = length(var_is);
    substr(model,p_model,len) = var_is;
    p_model = p_model + len + 1;
  end;
end;
/* if the format is character, c, put in both class and model statements */
if (val_fmt='c') then
do;
  len = length(var_is);
  substr(class,p_class,len) = var_is;
  p_class = p_class + len + 1;
  substr(model,p_model,len) = var_is;
  p_model = p_model + len + 1;
end;
if (done) then
do;
  call symput('model',trim(model));
  call symput('class',trim(class));
end;
run;
*renam(wide);
proc glm data=wide;
  class &class;
  model &model / ss3;
run;
quit;

```