SAS® GLOBAL FORUM 2018

USERS PROGRAM

April 8 – 11 | Denver, CO
Colorado Convention Center

#SASGF

An Easier and Faster Way to Untranspose a Wide File

Arthur Tabachneck
Toronto, ON Canada

Joe Matise
Chicago, IL

Matt Kastin
Chicago, IL

Gerhard Svolba
Wien, Austria

presentation overview:

- What the Untranspose macro is and where you can download it
- Why we created it and why it should be in your toolbox
- How you can use the macro
- Some useful techniques we used in creating it

question

Have you ever needed to untranspose a wide file back to the not-quite-as-wide file that was used to create it?

for example

You were given/sent the following file

id	income2015	income2016	income2017	expenses2015	expenses2016	expenses2017	debt2015	debt2016	debt2017
1	70000	75500	80000	60000	70000	81000	no	no	yes
2	50000	52000	55000	42000	53000	60000	no	yes	yes
3	80000	90000	99000	70000	75000	85000	no	no	no

label: Yearly Income

label: Yearly Expenses

label: Expenses>Income
format: $deptcode.

and you needed to untranspose it to look like

id	year	income	expenses	debt
1	2015	70000	60000	no
1	2016	75500	70000	no
1	2017	80000	81000	yes
2	2015	50000	42000	no
2	2016	52000	53000	yes
2	2017	55000	60000	yes
3	2015	80000	70000	no
3	2016	90000	75000	no
3	2017	99000	85000	no

there are at least two ways to do it with PROC TRANSPOSE: Way #1

```
proc transpose data=have out=long;
  by id;
  var income2015--debt2017;
run;
```

make file long

```
data long;
  set long;
  year=input(substr(_name_,anydigit(_name_)),8.);
  _name_=substr(_name_,1,anydigit(_name_)-1);
run;
```

parse name and varname from _name_

```
proc sort data=long;
  by id year;
run;
```

sort the file

```
proc transpose data=long out=wide (drop=_name_);
  by id year;
  var col1;
  id _name_;
run;
```

make file wide

```
data _null_;
  set long (where=(id eq 1)) end=last;
  if _n_ eq 1 then do;
    call execute('proc datasets library=work nolist;modify wide;');
  end;
  forexec=cat("label ",_name_,"=",_label_,";");
  call execute(forexec);
  if last then do;
    call execute('quit;');
  end;
run;
```

add variable labels

there are at least two ways to do it with PROC TRANSPOSE: Way #1

```
proc transpose data=have out=long;
  by id;
  var income2015--debt2017;
run;

data long;
  set long;
  year=input(substr
  _name_=substr(_na
run;

proc sort data=long
  by id year;
run;

proc transpose data
  by id year;
  var col1;
  id _name_;
run;

data _null_;
  set long (where=(
  if _n_ eq 1 then
    call execute('p
  end;
  forexec=cat("labe
  call execute(fore
  if last then do;
    call execute('c
  end;
run;
```

Problems:

requires a lot of steps and coding

loses formats

converts numeric fields to character

will output records even if they only contain missing values

procedure creates output files that contain a lot of redundant metadata

there are at least two ways to do it with PROC TRANSPOSE: Way #2

```
proc transpose data=have out=longi prefix=income;
  by id;  var income2015-income2017; run;
```

transpose Income

```
data _null_;
  set longi (obs=1);
  call execute('proc datasets library=work nolist;modify longi;');
  forexec=catt("label income1='",_label_,"';quit;");
  call execute(forexec);
run;
```

assign label

```
proc transpose data=have out=longe prefix=expenses;
  by id;  var expenses2015-expenses2017; run;
```

transpose expenses

```
data _null_;
  set longe (obs=1);
  call execute('proc datasets library=work nolist;modify longe;');
  forexec=catt("label expenses1='",_label_,"';quit;");
  call execute(forexec);
run;
```

assign label

```
proc transpose data=have out=longd prefix=debt;
  by id;  var debt2015-debt2017; run;
```

transpose debt

```
data _null_;
  set longd (obs=1);
  call execute('proc datasets library=work nolist;modify longd;');
  forexec=catt("label debt1='",_label_,"';quit;");
  call execute(forexec);
run;
```

assign label

```
data want (drop=_:);
  set longi (rename=(income1=income) drop=_:);
  set longe (rename=(expenses1=expenses) drop=_:);
  set longd (rename=(debt1=debt));
  year=input(substr(_name_, 5), 4.);
run;
```

combine files

there are at least two ways to do it with PROC TRANSPOSE: Way #2

```
proc transpose data=have out=longi prefix=income;
  by id;  var income2015-income2017; run;

data _null_;
  set longi (obs=1);
  call execute('proc datasets library=work nolist;modify longi;');
  forexec=catt("label income1='",_label_,"';quit;");
  call execute(forexec);
run;

proc transpose dat
  by id;  var expe

data _null_;
  set longe (obs=1
  call execute('pr
  forexec=catt("la
  call execute(for
run;

proc transpose dat
  by id;  var debt

data _null_;
  set longd (obs=1
  call execute('pr
  forexec=catt("la
  call execute(for
run;

data want (drop=_:
  set longi (renam
  set longe (renam
  set longd (renam
  year=input(subst
run;
```

Fewer Problems but:

- code requires more thought
- greater chance of introducing error
- code is less extensible
- will output records even if they only contain missing values
- procedure creates output files that contain a lot of redundant metadata

or .. just use the %untranspose macro

```
%untranspose(data=have, out=want, by=id,
                        id=year, var=income expenses debt);
```

- less code thus lower probability of coding errors
- more extensible
- you choose whether to output records with all missing values
- all variables maintain their original types, formats and labels
- uses same options and statements as proc transpose
- lets you create a file with metadata
- much faster than either method 1 or method 2

or .. just use the %untranspose macro

%untranspose(data=have, out=want, by=id

- less
- mo
- you
- all
- use
- lets
- much faster than either Method 1 or Method 2

Speed differential increases with both
number of records and number of variables

with 50,000 records and three variables

70 times faster than Method 1

11.6 times faster than Method 2

with 500,000 records and three variables

207.8 times faster than Method 1

27.2 times faster than Method 2

the macro uses named parameters

Benefits

You determine the default values for each parameter

You only need to specify the parameters if they're different than their default values

No need to differentiate between options and statements, or the order in which they're included

No relearning necessary .. the macro uses the same names as proc transpose for all parameters

the macro's named parameters

- libname_in
- libname_out
- data
- out
- by
- prefix
- var
- id
- id_informat
- id_format
- var_first
- delimiter
- suffix
- copy
- missing
- metadata
- makelong
- max_length

the macro's named parameters

libname_in
libname_out
data
out
by
prefix
var
id
id_informat
id_format
var_first
delimiter
suffix
copy
missing
metadata
makelong
max_length

you can change the default libnames used for one-level file names

e.g. libname_in = some_path

the macro's named parameters

libname_in
libname_out
data

id_format
var_first
delimiter

id
id_informat

makelong
max_length

example transposed variable name

_04APR2018_Weight_Actual

the macro's named parameters

libname_in
libname_out
data
out
by
prefix
var
id
id_informat

id_format
var_first
delimiter
suffix
copy
missing
metadata
makelong
max_length

example transposed variable name

_04APR2018_Weight_Actual

the macro's named parameters

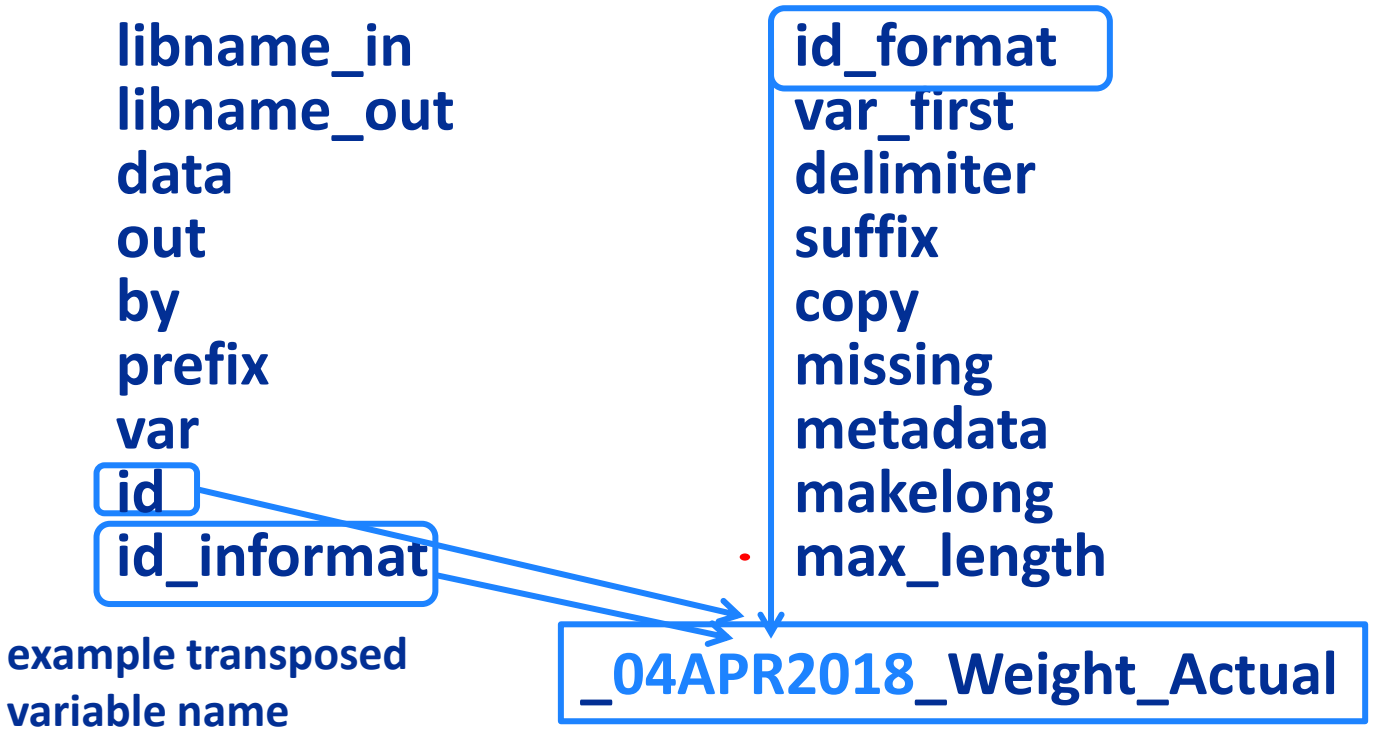
libname_in
libname_out
data
out
by
prefix
var
id
id_informat

id_format
var_first
delimiter
suffix
copy
missing
metadata
makelong
max_length

example transposed variable name

_04APR2018_Weight_Actual

the macro's named parameters



the macro's named parameters

libname_in
libname_out
data
out
by
prefix
var
id
id_informat

id_format
var_first
delimiter
suffix
copy
missing
metadata
makelong
max_length

example transposed variable name

_04APR2018_Weight_Actual

the macro's named parameters

libname_in
libname_out
data
out
by
prefix
var
id
id_informat

id_format
var_first
delimiter
suffix
copy
missing
metadata
makelong
max_length

example transposed variable name

_04APR2018_Weight_Actual

the macro's named parameters

libname_in
libname_out
data
out
by
prefix
var
id
id_informat

id_format
var_first
delimiter
suffix
copy
missing
metadata
makelong
max_length

example transposed variable name

_04APR2018_Weight_Actual

the macro's named parameters

libname_in
libname_out
data
out
by
prefix
var
id
id_informat
id_format
var_first
delimiter
suffix
copy
missing
metadata
makelong
max_length

Specify whether the macro should output records that only have missing values for the variables listed in the var parameter?

the macro's named parameters

libname_in
libname_out
data
out
by
prefix
var
id
id_informat
id_format
var_first
delimiter
suffix
copy
missing
metadata
makelong
max_length

Specify that you want the macro to output a dataset containing all of the variable types, lengths, formats, informats and labels

the macro's named parameters

libname_in
libname_out
data
out
by
prefix
var
id
id_informat
id_format
var_first
delimiter
suffix
copy
missing
metadata
makelong
max_length

Specify that you want the macro to output a long dataset (i.e., a separate record for each by, id and var combination)

the macro's named parameters

- libname_in
- libname_out
- data
- out
- by
- prefix
- var
- id
- id_informat
- id_format
- var_first
- delimiter
- suffix
- copy
- missing
- metadata
- makelong
- max_length

Minimize processing time (when untransposing to a long format) by indicating the maximum variable length

where to get the macro

Copy the SAS code from:

http://www.sascommunity.org/wiki/An_Easier_and_Faster_Way_to_Untranspose_a_Wide_File

save the file as *untranspose.sas* in a directory that exists in your SASAUTOS* path

*** see: http://analytics.ncsu.edu/sesug/2008/SBC-126.pdf**

The page lets you download the paper, code, and a tip sheet that includes a number of examples. e.g.:

Usage Examples: The following are some examples of how you might use the macro. For each example the wide dataset's name is *have* and resides in the work library, and the less wide or long dataset created by the macro is called *want* and also resides in the work library.

dataset: ***have***

id	income2015	income2016	income2017	expenses2015	expenses2016	expenses2017	debt2015	debt2016	debt2017
1	70000	75500	80000	60000	70000	81000	no	no	yes
2	50000	52000	55000	42000	53000	60000	no	yes	yes
3	80000	90000	99000	70000	75000	85000	no	no	no

macro call: ***%untranspose***(data=have, out=want, by=id, id=year, var=income expenses debt)

dataset: ***want***

id	year	income	expenses	debt
1	2015	70000	60000	no
1	2016	75500	70000	no
1	2017	80000	81000	yes
2	2015	50000	42000	no
2	2016	52000	53000	yes
2	2017	55000	60000	yes
3	2015	80000	70000	no
3	2016	90000	75000	no
3	2017	99000	85000	no

dataset: ***have***

id	income2015	income2016	income2017	expenses2015	expenses2016	expenses2017	debt2015	debt2016	debt2017
1	70000	75500	80000	60000	70000	81000	no	no	yes
2	50000	52000	55000	42000	53000	60000	no	yes	yes
3	80000	90000	99000	70000	75000	85000	no	no	no

macro call: ***%untranspose***(data=have (obs=1), out=want, by=id, id=year, var=income expenses debt)

dataset: ***want***

id	year	income	expenses	debt
1	2015	70000	60000	no
1	2016	75500	70000	no
1	2017	80000	81000	yes

dataset: ***have***

id	income2015	income2016	income2017	expenses2015	expenses2016	expenses2017	debt2015	debt2016	debt2017
1	70000	75500	80000	60000	70000	81000	no	no	yes
2	50000	52000	55000	42000	53000	60000	no	yes	yes
3	80000	90000	99000	70000	75000	85000	no	no	no

macro call: ***%untranspose***(data=have (where=(id eq 2)), out=want, by=id, id=year, var=income expenses debt)

dataset: ***want***

id	year	income	expenses	debt
2	2015	50000	42000	no
2	2016	52000	53000	yes
2	2017	55000	60000	yes

development: some techniques we used

Identifying whether a 1 or 2-level name was used and, if necessary, parsing out all dataset options. e.g., to parse a data parameter like: data=mylib.have(where=(gender eq 'male'))

```
%let lp=%sysfunc(findc(%superq(data),%str(%())));
%if &lp. %then %do;
  %let rp=%sysfunc(findc(%superq(data),%str(%)),b));
  %let dsoptions=%qsysfunc(substrn(%nrstr(%superq(data)),
     &lp+1,&rp-&lp-1));
  %let data=%sysfunc(substrn(%nrstr(%superq(data)),1,%eval(&lp-1)));
%end;
%else %let dsoptions=;
```

development: some techniques we used

Creating one record datasets and then using PROC SQL to create macro variables from dictionary.columns

```
data t_e_m_p;
   set &libname_in..&data. (obs=1 keep=&copy.);
  run;

  proc sql noprint;
    select name
      into :to_copy separated by " "
        from dictionary.columns
          where libname="WORK" and
                memname="T_E_M_P";
  quit;
```

development: some techniques we used

Using a macro variable that contains a space-separated list of variable names to create a SAS dataset

```
data t_e_m_p;
   array vars(*) &var.;
   output;
run;
```

summary of what I just presented:

- What the Untranspose macro is and where you can download it
- Why we created it and why it should be in your toolbox
- How you can use the macro
- Some useful techniques we used in creating it

Your comments and questions are valued and encouraged

Contact the Authors

Arthur Tabachneck, Ph.D., CEO
Analyst Finder, Inc.
Toronto, ON
art@analystfinder.com

Joe Matise
NORC @ the University of Chicago
Chicago, IL
snoopy369@gmail.com

Gerhard Svolba
SAS Institute
Wien, Austria
Gerhard.Svolba@sas.com

Matt Kastin
NORC @ the University of Chicago
Chicago, IL
fried.egg@verizon.com