Next Presentation:

A Better Way to Flip (Transpose) a SAS® Dataset

Presenter: **Arthur Tabachneck**

Art holds a PhD from Michigan State University, has been a SAS user since 1974, is president of the Toronto Area SAS Society and has received such recognitions as the SAS Customer Value Award (2009), SAS-L Hall of Fame (2011), SAS Circle of Excellence (2012) and, in 2013, was recognized as being the first SAS Discussion Forum participant to be awarded more than 10,000 points

A Better Way to Flip (Transpose) a SAS® Dataset

Art297
Arthur Tabachneck
Thornhill, ON Canada

KSharp
Xia Ke Shan
Beijing, China

Joe Whitehurst
Atlanta, GA
Joe Whitehurst

Robert Virgile
Lexington, MA
Astounding

Presentation Overview

- What the %transpose macro is
- The macro's benefits
- How you would use the macro
- How the macro works
- Potential applications

What the %transpose macro is

- It's a SAS macro
- Looks and feels almost exactly like PROC TRANSPOSE
- Doesn't have all of the capabilities of PROC TRANSPOSE as it was designed for just two purposes: to convert tall files into wide files
  to make wide files even wider
- Has virtually the same options and statements as PROC TRANSPOSE + a few more
- Is easier to use than PROC TRANSPOSE
- Runs significantly faster than PROC TRANSPOSE

Have you ever had to flip a SAS dataset from being tall to being wide?

i.e., from:

idnum	date	var1
1	2001JAN	SD
1	2001FEB	EF
1	2001MAR	HK
2	2001JAN	GH
2	2001APR	MM
2	2001MAY	JH

Have you ever had to flip a SAS dataset from being tall to being wide?

to:

idnum	var1_2001JAN	var1_2001FEB	var1_2001MAR	var1_2001APR	var1_2001MAY
1	SD	EF	HK		
2	GH			MM	JH

if you have, then you are probably already familiar with PROC TRANSPOSE

```
proc transpose
          data=have
          out=want (drop=_:)
          prefix=var1_;
    by idnum;
    var var1;
    id date;
run;
```

if you have, then you are probably already familiar with PROC TRANSPOSE

```
proc transpose
         data=have
         out=want (drop=_:)
         prefix=var1_;
    by idnum;
    var var1;
    id date;
run;
```

Call the procedure

if you have, then you are probably already familiar with PROC TRANSPOSE

```
proc transpose
        data=have
        out=want (drop=_:)
        prefix=var1_;
    by idnum;
    var var1;
    id date;
run;
```

data to be transposed

if you have, then you are probably already familiar with PROC TRANSPOSE

```
proc transpose
         data=have
         out=want (drop=_:)
         prefix=var1_;
   by idnum;
   var var1;
   id date;
run;
```

output filename

if you have, then you are probably already familiar with PROC TRANSPOSE

```
proc transpose
        data=have
        out=want (drop=_:)
        prefix=var1_;
   by idnum;
   var var1;
   id date;
run;
```

drop unwanted automatic system variables

if you have, then you are probably already familiar with PROC TRANSPOSE

```
proc transpose
        data=have
        out=want (drop=_:)
        prefix=var1_;
    by idnum;
    var var1;
    id date;
run;
```

string you want inserted at far left side of new variable names

if you have, then you are probably already familiar with PROC TRANSPOSE

```
proc transpose
         data=have
         out=want (drop=_:)
         prefix=var1_;
    by idnum;
    var var1;
    id date;
run;
```

$\leftarrow$ transposed record level

if you have, then you are probably already familiar with PROC TRANSPOSE

```
proc transpose
        data=have
        out=want (drop=_:)
        prefix=var1_;
    by idnum;
    var var1;
    id date;
run;
```

variable(s) to be transposed

if you have, then you are probably already familiar with PROC TRANSPOSE

```
proc transpose
        data=have
        out=want (drop=_:)
        prefix=var1_;
    by idnum;
    var var1;
    id date;
run;
```

variable whose formatted values will be part of transposed variable names

if you have, then you are probably already familiar with PROC TRANSPOSE

```
proc transpose
          data=have
          out=want (drop=_:)
          prefix=var1_;
    by idnum;
    var var1;
    id date;
run;
```

$\leftarrow$ execute all of the above statements

not difficult, but you need to know:

```
proc transpose
          data=have
          out=want (drop=_:)
          prefix=var1_;
    by idnum;
    var var1;
    id date;
run;
```

which are options

not difficult, but you need to know:

```
proc transpose
          data=have
          out=want (drop=_:)
          prefix=var1_;
    by idnum;
    var var1;
    id date;
run;
```

which are options

which are statements

not difficult, but you need to know:

```
proc transpose
          data=have
          out=want (drop=_:)
          prefix=var1_;
    by idnum;
    var var1;
    id date;
run;
```

which are options
which are statements
that you have to "drop" unwanted system variables

•

not difficult, but you need to know:

```
proc transpose
          data=have
          out=want (drop=_:)
          prefix=var1_;
    by idnum;
    var var1;
    id date;
run;
```

which are options
which are statements
that you have to "drop" unwanted system variables
that you have to specify a prefix

•

not difficult, but you need to know:

Note to self:
remember to first
sort the data

```
proc transpose
          data=have
          out=want (drop=_:)
          prefix=var1_;
    by idnum;
    var var1;
    id date;
run;
```

which are options
which are statements
that you have to "drop" unwanted system variables
that you have to specify a prefix

- and that you have to presort your data

would you be interested in knowing how to obtain the same result with the following code?

it may look like the PROC TRANSPOSE code, but:

No system variables to drop

```
%transpose( data=have,  out=want,  by=idnum,
            var=var1,    id=date,    sort=yes,
            delimiter=_)
```

No need for a prefix (var names automatically included)

No need to differentiate between options and statements as they are all of the form: parameter=value,

No need to presort your data

easier to code (less to type)

runs 10 times faster than PROC TRANSPOSE

or if you needed to flip a more complex SAS dataset from being wide to being wider

i.e., from:

idnum	date	var1	var2
1	31MAR2013	1	SD
1	30JUN2013	2	EF
1	30SEP2013	3	HK
1	31DEC2013	4	HL
2	31MAR2013	5	GH
2	30JUN2013	6	MM
2	30SEP2013	7	JH
2	31DEC2013	8	MS

or if you needed to flip a more complex SAS dataset from being tall to being wide

to:

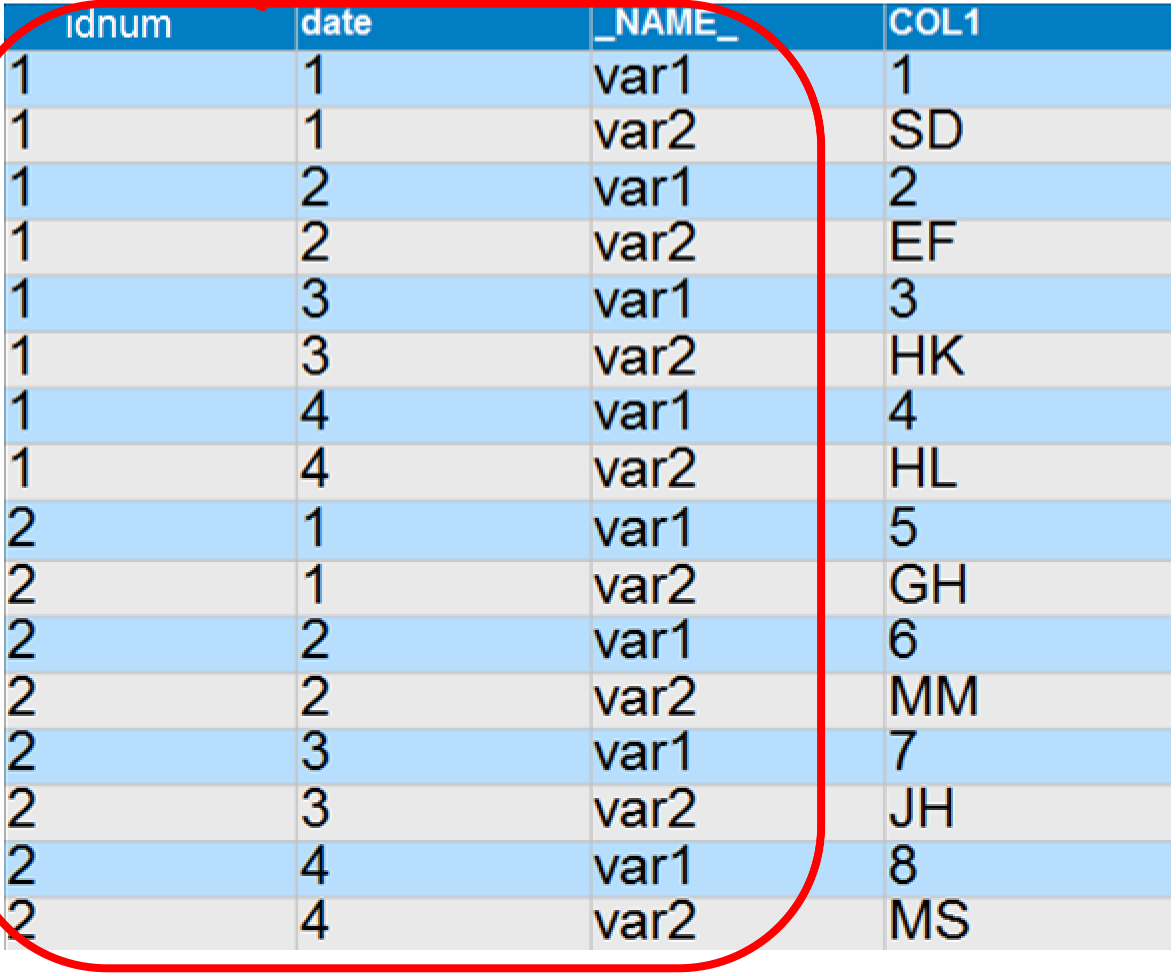
idnum	var1 Qtr1	var2 Qtr1	var1 Qtr2	var2 Qtr2	var1 Qtr3	var2 Qtr3	var1 Qtr4	var2 Qtr4
1	1	SD	2	EF	3	HK	4	HL
2	5	GH	6	MM	7	JH	8	MS

Again, you can use PROC TRANSPOSE but it would require at least two steps

First you have to make the table even taller (i.e., one record for each *by variable* and *var* combination)

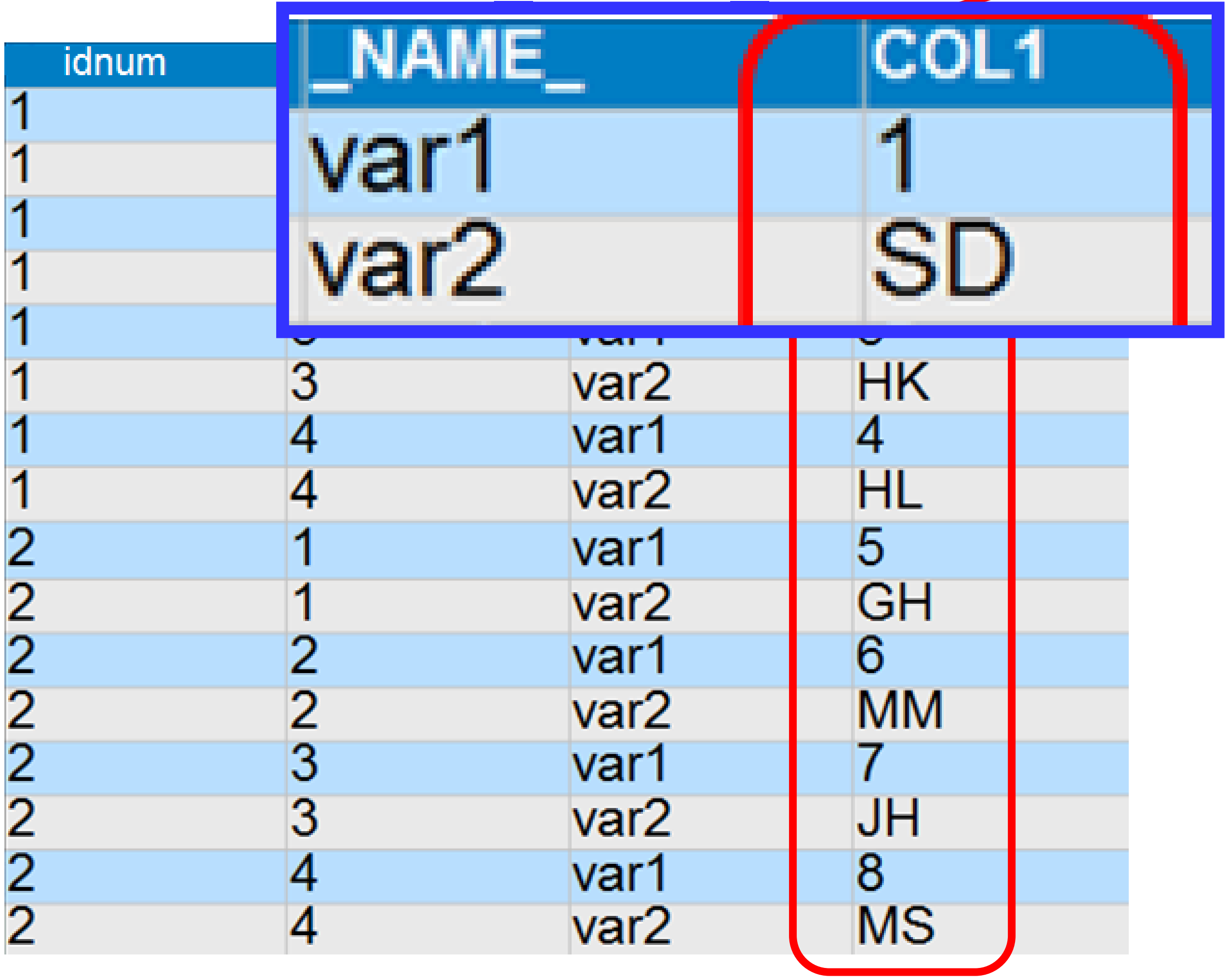
```
proc transpose data=have out=tall ;
  by idnum date;
  var var1-var2;
  format date qtr1.;
run;
```

That will create a taller file (i.e., 1 record for each *by variable* and *var* combination)



idnum	date	_NAME_	COL1
1	1	var1	1
1	1	var2	SD
1	2	var1	2
1	2	var2	EF
1	3	var1	3
1	3	var2	HK
1	4	var1	4
1	4	var2	HL
2	1	var1	5
2	1	var2	GH
2	2	var1	6
2	2	var2	MM
2	3	var1	7
2	3	var2	JH
2	4	var1	8
2	4	var2	MS

That will create a taller file (with var names now in _NAME_ and values in COL1)



NAME	COL1
var1	1
var2	SD

idnum			
1	[illegible]	[illegible]	[illegible]
1	[illegible]	[illegible]	[illegible]
1	[illegible]	[illegible]	[illegible]
1	[illegible]	[illegible]	[illegible]
1	3	var1	3
1	3	var2	HK
1	4	var1	4
1	4	var2	HL
2	1	var1	5
2	1	var2	GH
2	2	var1	6
2	2	var2	MM
2	3	var1	7
2	3	var2	JH
2	4	var1	8
2	4	var2	MS

Then, to make the table wide (i.e., one record for each *by* variable)

you need to run PROC TRANSPOSE a 2nd time

```
proc transpose data=tall out=want (drop=_:)
    delimiter=_Qtr;
 by idnum;
 id _name_ date;
 var col1;
run;
```

Oh, did we mention? There are a couple of problems with the method

result:

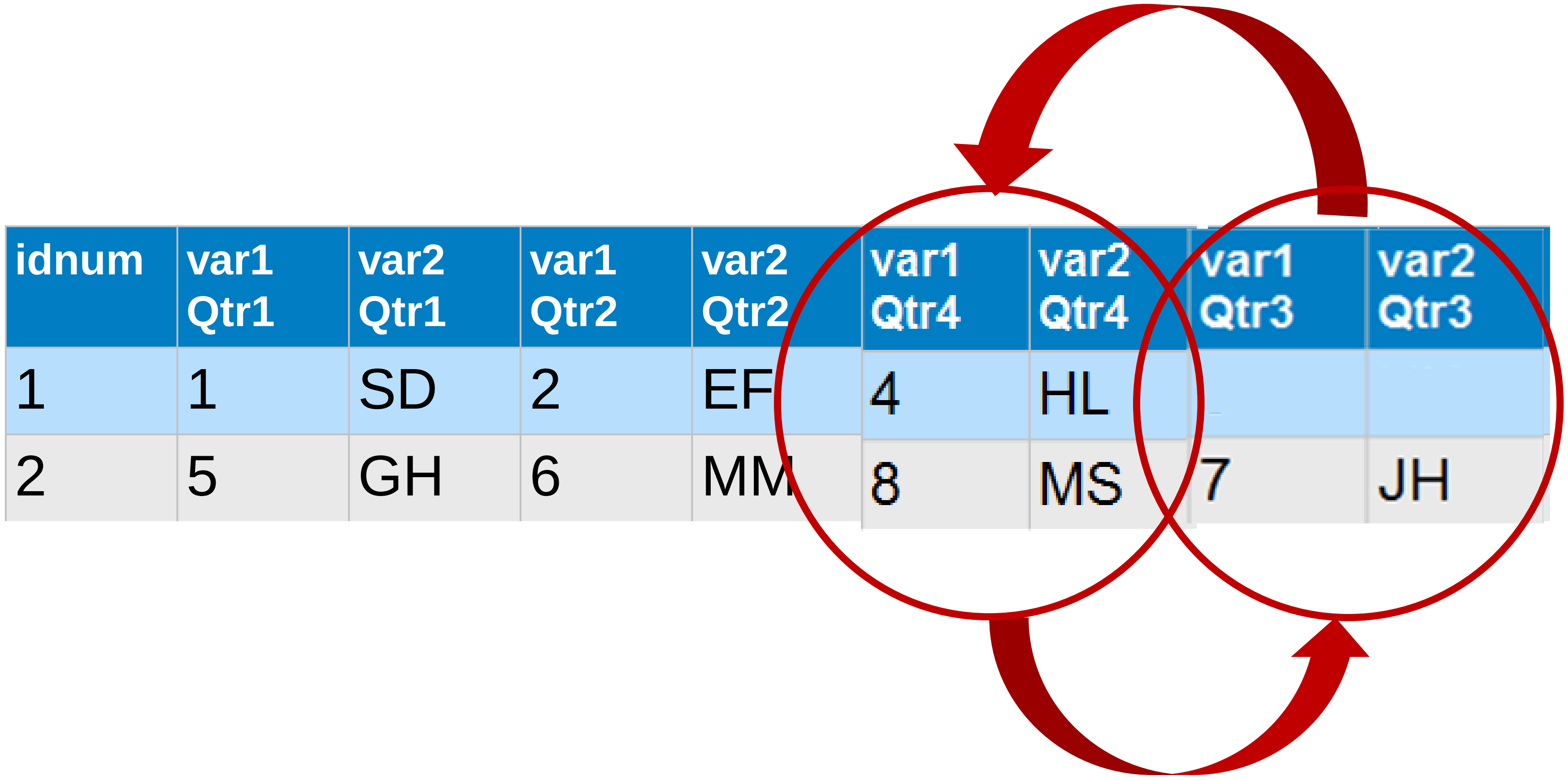
idnum	var1 Qtr1	var2 Qtr1	var1 Qtr2	var2 Qtr2	var1 Qtr3	var2 Qtr3	var1 Qtr4	var2 Qtr4
1	1	SD	2	EF	3	HK	4	HL
2	5	GH	6	MM	7	JH	8	MS

- The numeric variables are now character variables

and if the first idnum was missing data for one date

idnum	date	var1	var2
1	31MAR2013	1	SD
1	30JUN2013	2	EF
1	31DEC2013	4	HL
2	31MAR2013	5	GH
2	30JUN2013	6	MM
2	30SEP2013	7	JH
2	31DEC2013	8	MS

the output variable order will be a bit distorted



idnum	var1 Qtr1	var2 Qtr1	var1 Qtr2	var2 Qtr2	var1 Qtr4	var2 Qtr4	var1 Qtr3	var2 Qtr3
1	1	SD	2	EF	4	HL		
2	5	GH	6	MM	8	MS	7	JH

would you be interested in knowing how to obtain the right result with the following code?

```
%transpose(data=have,  out=need,      by=idnum,
           id=date,       format=qtr1.,  delimiter=_Qtr,
           var=var1-var2, sort=yes)
```

How about if you knew that the macro:

only requires one step

only needs one pass through the data

doesn't produce distorted results

can run more than 50 times faster than PROC TRANSPOSE

why the macro runs faster than PROC TRANSPOSE

- a datastep, using arrays, is simply more efficient than PROC TRANSPOSE
- the macro creates and runs a SAS program that contains such a datastep
- since PROC TRANSPOSE's statements and options necessarily define all of the relevant variables, the macro simply puts those variables in a keep option
- the datastep uses separate arrays for character and numeric variables so, if both character and numeric variables are transposed, they aren't all converted to character as with PROC TRANSPOSE

How the macro works
if we have: dataset *have*

idnum	date	var1	var2
1	31MAR2013	1	SD
1	30JUN2013	2	EF
1	30SEP2013	3	HK
1	31DEC2013	4	HL
2	31MAR2013	5	GH
2	30JUN2013	6	MM
2	30SEP2013	7	JH
2	31DEC2013	8	MS

How the macro works
and we need: dataset *need*

idnum	var1 Qtr1	var2 Qtr1	var1 Qtr2	var2 Qtr2	var1 Qtr3	var2 Qtr3	var1 Qtr4	var2 Qtr4
1	1	SD	2	EF	3	HK	4	HL
2	5	GH	6	MM	7	JH	8	MS

and we submit:

```
• %transpose(data=have,  out=want,      by=idnum,
             id=date,       format=qtr1.,  delimiter=_Qtr,
             var=var1-var2,  sort=yes)
```

How the macro works:

❶ since the *sort* parameter was set to 'yes', the macro will run PROC SORT only keeping the relevant variables

•

How the macro works:

❶ since the *sort* parameter was set to 'yes', the macro will run PROC SORT only keeping the relevant variables

❷ then the macro will run a datastep like the following:

```
data work.want (drop=date ____: var1-var2);
 set work.have (keep=idnum date var2 var1);
 by idnum;
 retain want_chr want_num;
 array have_chr(*)$ var2;  array have_num(*) var1;
 array want_chr(*)$ var2_Qtr1 var2_Qtr2 var2_Qtr3 var2_Qtr4;
 array want_num(*) var1_Qtr1 var1_Qtr2 var1_Qtr3 var1_Qtr4;
 format var1_: 1. var 2_: $2.;
 if first.idnum then call missing(of want_chr(*));
 ___nchar=put(date,labelfmt.)*dim(have_chr);
 do ___i=1 to dim(have_chr);
    want_chr(___nchar+___i)=have_chr(___i);
 end;
 if first.idnum then call missing(of want_num(*));
 ___nnum=put(date,labelfmt.)*dim(have_num);
 do ___i=1 to dim(have_num);
    want_num(___nnum+___i)=have_num(___i);
 end;
 if last.idnum then output;
run;
```

How the macro works:

➊ since the *sort* parameter was set to 'yes', the macro will run PROC SORT only keeping the relevant variables

➋ then the macro will run a datastep like the following:

```
data work.want (drop=date ___: var1-var2);
 set work.h
 by idnum;
 retain wan
 array have
 array want                                          _Qtr4;
 array want                                          _Qtr4;
 format var1_: 1. var 2_: $2.;
 if first.idnum then call missing(of want_chr(*));
 ___nchar=put(date,labelfmt.)*dim(have_chr);
 do ___i=1 to dim(have_chr);
   want_chr(___nchar+___i)=have_chr(___i);
 end;
 if first.idnum then call missing(of want_num(*));
 ___nnum=put(date,labelfmt.)*dim(have_num);
 do ___i=1 to dim(have_num);
   want_num(___nnum+___i)=have_num(___i);
 end;
 if last.idnum then output;
run;
```

labelfmt. is a format, created by the macro, and reflects the ordered names of the transposed variables (i.e., from 1 to n)

and you can use almost all the features that you can with PROC TRANSPOSE

```
%transpose( libname_in=,        libname_out=,
            data=,              out=,
            by=,                prefix=,
            var=,               autovars=,
            id=,                var_first=,
            format=,            delimiter=,
            copy=,              drop=,
            sort=,              sort_options=,
            use_varname=,       preloadfmt=,
            guessingrows=)
```

and you can use almost all the features that you can with PROC TRANSPOSE

- plus some additional ones

```
%transpose( libname_in=,        libname_out=,
            data=,              out=,
            by=,                prefix=,
            var=,               autovars=,
            id=,                var_first=,
            format=,            delimiter=,
            copy=,              drop=,
            sort=,              sort_options=,
            use_varname=,       preloadfmt=,
            guessingrows=)
```

the %transpose macro's features

parameter: libname_in

the name of the SAS library that contains the dataset you want to transpose

```
%transpose( libname_in=,        libname_out=,
            data=,              out=,
            format=,            delimiter=,
            copy=,              drop=,
            sort=,              sort_options=,
            use_varname=,       preloadfmt=,
            guessingrows=)
```

Note: if this parameter is left null, and only a one-level filename is assigned to the *data* parameter, this parameter will be set to *work*

the %transpose macro's features

parameter: libname_out

the name of the SAS library where you want the transposed dataset written

```
%transpose( libname_in=,       libname_out=,
            data=,             out=,
            by=                prefix=
            copy=,             drop=,
            sort=,             sort_options=,
            use_varname=,      preloadfmt=,
            guessingrows=)
```

Note: if this parameter is left null, and only a one-level filename is assigned to the *out* parameter, this parameter will be set to *work*

the %transpose macro's features

parameter: data

the one or two-level name of the file you want to transpose

```
%transpose( libname_in=,        libname_out=,
            data=,              out=,
            by=                 prefix=

            copy=,              drop=,
            sort=,              sort_options=,
            use_varname=,       preloadfmt=,
            guessingrows=)
```

Note: if a two-level filename is supplied, the first level will replace the value that had been assigned to the libname_in parameter.

the %transpose macro's features

parameter: out

the one or two-level name you want assigned to the transposed file

```
%transpose( libname_in=,        libname_out=,
            data=,              out=,
            by=                 prefix=
            copy=,              drop=,
            sort=,              sort_options=,
            use_varname=,       preloadfmt=,
            guessingrows=)
```

Note: if a two-level filename is supplied, the first level will replace the value that had been assigned to the libname_out parameter.

the %transpose macro's features

parameter: by

the variable(s) you want to use to form *by* groups. *By* groups define the record level of the transposed file

```
%transpose( libname_in=,       libname_out=,
            data=,             out=,
            by=,               prefix=,
            var=,              autovars=,
            id=,               var_first=,
            format=,           delimiter=,
            copy=,             drop=,
            sort=,             sort_options=,
            use_varname=,      preloadfmt=,
            guessingrows=)
```

the %transpose macro's features

parameter: prefix

a leading string you want to be the first characters of the transposed variable names

```
%transpose( libname_in=,         libname_out=,
            data=,               out=,
            by=,                 prefix=,
            var=,                autovars=,
            id=,                 var_first=,
            format=,             delimiter=,
            copy=,               drop=,
            sort=,               sort_options=,
            use_varname=,        preloadfmt=,
            guessingrows=)
```

the %transpose macro's features

parameter: var

the variables to be transposed. If null, selection depends on the autovars parameter. Any combination of variable names and lists accepted by a datastep keep option may be used

```
%transpose( libname_in=,        libname_out=,
            data=,              out=,
            by=,                prefix=,
            var=,               autovars=,
            id=,                var_first=,
            format=,            delimiter=,
            copy=,              drop=,
            sort=,              sort_options=,
            use_varname=,       preloadfmt=,
            guessingrows=)
```

the %transpose macro's features
parameter: autovars

determines whether char(acter), num(eric) or all variables should be transposed if the var parameter is null

```
%transpose( libname_in=,          libname_out=,
            data=,                out=,
            by=,                  prefix=,
            var=,                 autovars=,
            id=                   var_first=
```

* * * * * * F E A T U R E * * * * * *

Where PROC TRANSPOSE will only include all numeric variables if there is no var statement,

this parameter lets you indicate if you want all numeric variables, all character variables or simply all variables

the %transpose macro's features
parameter: id

the variable whose values will be concatenated with the var variables selected to be transposed

```
%transpose( libname_in=,        libname_out=,
            data=,              out=,
            by=,                prefix=,
            var=,               autovars=,
            id=,                var_first=,
            format=,            delimiter=,
            copy=,              drop=,
            sort=,              sort_options=,
            use_varname=,       preloadfmt=,
            guessingrows=)
```

the %transpose macro's features

parameter: var_first

determines which is named first in transposed variables:

YES: prefix var *name* delimiter id *value*

```
%transpose( libname_in=,        libname_out=,
            data=,              out=,
            by=,                prefix=,
            var=,               autovars=,
            id=,                var_first=,
            format=,            delimiter=,
            copy=,              drop=,
            sort=,              sort_options=,
            use_varname=,       preloadfmt=,
            guessingrows=)
```

the %transpose macro's features

parameter: var_first

determines which is named first in transposed variables:

YES: prefix var *name* delimiter id *value*

NO: prefix id *value* delimiter var *name*

```
%transpose( libname_in=,       libname_out=,
            data=,             out=,
            by=,               prefix=,
            var=,              autovars=,
            id=,               var_first=,
            format=,           delimiter=,
            copy=,             drop=,
            sort=,             sort_options=,
            use_varname=,      preloadfmt=,
            guessingrows=)
```

the %transpose macro's features

parameter: format

the format you want applied to the id variable

```
%transpose( libname_in=,        libname_out=,
            data=,              out=,
            by=,                prefix=,
            var=,               autovars=,
            id=,                var_first=,
            format=,            delimiter=,
            copy=,              drop=,
            sort=,              sort_options=,
            use_varname=,       preloadfmt=,
            guessingrows=)
```

the %transpose macro's features

parameter: delimiter

the string you want assigned between the concatenated id values and var names

```
%transpose( libname_in=,        libname_out=,
            data=,              out=,
            by=,                prefix=,
            var=,               autovars=,
            id=,                var_first=,
            format=,            delimiter=,
            copy=,              drop=,
            sort=,              sort_options=,
            use_varname=,       preloadfmt=,
            guessingrows=)
```

the %transpose macro's features
parameter: copy

the variables you want copied rather than transposed
Only the last value found for a by record will be copied

```
%transpose( libname_in=,         libname_out=,
            data=,               out=,
            by=,                 prefix=,
            var=,                autovars=,
            id=,                 var_first=,
            format=,             delimiter=,
            copy=,               drop=,
            sort=,               sort_options=,
            use_varname=,        preloadfmt=,
            guessingrows=)
```

the %transpose macro's features

parameter: drop

the variable(s) you want dropped from the transposed file

**** only relevant if you want to drop any of the *by* variables ****

```
%transpose( libname_in=,        libname_out=,
            data=,              out=,
            by=,                prefix=,
            var=,               autovars=,
            id=,                var_first=,
            format=,            delimiter=,
            copy=,              drop=,
            sort=,              sort_options=,
            use_varname=,       preloadfmt=,
            guessingrows=)
```

the %transpose macro's features
parameter: sort

whether the input dataset should be sorted (YES or NO):
for both, only &by, &id, &var and &copy variables will be used
If Yes, the input data will be sorted using the *noequals* option

```
%transpose( libname_in=,        libname_out=,
            data=,              out=,
            by=,                prefix=,
            var=,               autovars=,
            id=,                var_first=,
            format=,            delimiter=,
            copy=,              drop=,
            sort=,              sort_options=,
            use_varname=,       preloadfmt=,
            guessingrows=)
```

How many of you have ever:

- forgotten to run proc sort before running another proc that required sorted data?
- run proc sort but didn't include the options that can make the process more efficient (e.g., *noequals*, *presorted* and *tagsort*)?
- run a proc that only used a few of a file's variables, but didn't include a *keep* dataset option to limit the amount of data that had to be processed?

Compare the performance of the following two sets of *almost* identical code run on a file with 40,000 records and 1,002 variables

```
PROC SORT data=have out=need;
  by idnum date;
run;  took 2.41 seconds CPU time


PROC TRANSPOSE data=need out=want (drop=_:)
                                  prefix=var1_Qtr;
  by idnum;
  var var1;
  id date;
  format date Qtr1.;
run;  took 0.74 seconds CPU time
```

Compare the performance of the following two sets of *almost* identical code run on a file with 40,000 records and 1,002 variables

```
PROC SORT data=have (keep=idnum date var1)
                    out=need noequals;
  by idnum date;
run;  took 0.33 seconds CPU time


PROC TRANSPOSE data=need out=want (drop=_:)
                         prefix=var1_Qtr;
  by idnum;
  var var1;
  id date;
  format date Qtr1.;
run;  took 0.16 seconds CPU time
```

6.39 times faster

Compare the performance of both sets of code with the %transpose macro

```
%transpose( data=have, out=want, var=var1,
            by=idnum,  id=date,    format=Qtr1.,
            sort=yes,    delimiter=_Qtr,
            guessingrows=4)
```

took 0.37 seconds CPU time
i.e., 33.6% faster than the optimized code and
8.5 times faster than the non-optimized code

the %transpose macro's features

parameter: sort

* * * * * * * F E A T U R E * * * * * * *

Where PROC TRANSPOSE will take up unnecessary system time unless you include a keep or drop statement,

the macro will always ensure that only relevant variables are kept

+ if set to YES this parameter will ensure that your data are presorted using the *noequals* sort option and any other options you specify in the *sort_options* parameter

sort=, sort_options=,
use_varname=, preloadfmt=,
guessingrows=)

the %transpose macro's features
parameter: sort_options
whether additional options should be specified if the sort parameter is set to YES

```
%transpose( libname_in=          libname_out=
```

* * * * * * * F E A T U R E * * * * * * *

While the *keep* and *noequals* sort options will always be used, based on your data there are other sort options you may want to specify that could increase efficiency and/or ensure that your data are sorted (e.g., *presorted*, *tagsort* and *force*)

```
            sort=,                sort_options=,
            use_varname=,         preloadfmt=,
            guessingrows=)
```

the %transpose macro's features

parameter: use_varname

if set to NO the var variable name(s) will NOT be assigned to the transposed variable names

```
%transpose( libname_in=,        libname_out=,
            data=,              out=,
            by=,                prefix=,
            var=                autovars=
```

*** * * * * * * N O T E * * * * * * ***

Only needed for simple transpositions where the var variable name is not needed/desired

```
            sort=,              sort_options=,
            use_varname=,       preloadfmt=,
            guessingrows=)
```

the %transpose macro's features

parameter: preloadfmt

lets you specify a SAS dataset you want to use to control the order which id value levels will be output

%transpose(libname_in=, libname_out=,

* * * * * * F E A T U R E * * * * * * *

Lets you specify the id value levels, rather than having the macro analyze your dataset in order to discover them. Can significantly improve the macro's performance and lets you include levels that aren't present in your data.

sort=, sort_options=,
use_varname=, preloadfmt=,
guessingrows=)

* * * * * * an example * * * * * *

```
data have;
  input idnum date var1 $;
  informat date date9.;
  cards;
1 01jan2013     SD
1 01feb2013     EF
1 01mar2013     HK
2 01jan2013     GH
2 01apr2013     MM
2 01may2013     JH
;
```

```
data order;
  input date date9. order;
  cards;
01jan2013     1
01feb2013     2
01mar2013     3
01apr2013     4
01may2013     5
;
```

```
%transpose(data=have, out=want, by=idnum, var=var1,
    id=date, format=yymon7., delimiter=_, preloadfmt=order)
```

the %transpose macro's features

parameter: guessingrows

the number of rows to be read to determine the correct order for the set of transposed variables

%

* * * * * * F E A T U R E * * * * * *

With PROC TRANSPOSE the transposed variables will be in the order they are initially found in the data

this parameter controls the order based on the values found in the first *guessingrows'* records

If this parameter isn't specified, and an order file isn't used, the macro will read ALL of your data records

guessingrows=)

the %transpose macro's features

parameter: all parameters

```
%transpose( libname_in=,         libname_out=,
            data=,               out=,
```

*** * * * * * F E A T U R E * * * * * ***

Since they are all macro *named parameters* you have direct control over their default values

If you set them (in the macro declaration) to commonly used values, they don't have to be specified UNLESS you want to change their value

```
            guessingrows=)
```

Benefits of the approach

- less typing thus fewer errors

Benefits of the approach

- less typing thus fewer errors
- contains some features that would be nice to see available with some SAS procs

Benefits of the approach

- less typing thus fewer errors
- contains some features that would be nice to see available with some SAS procs

for example

named parameters (user controlled defaults)
autovars
automatic optimization (keep, sort and noequals)
preloadfmt
guessingrows

Benefits of the approach

- less typing thus fewer errors
- contains some features that would be nice to see available with some SAS procs
- easier to learn than PROC TRANSPOSE

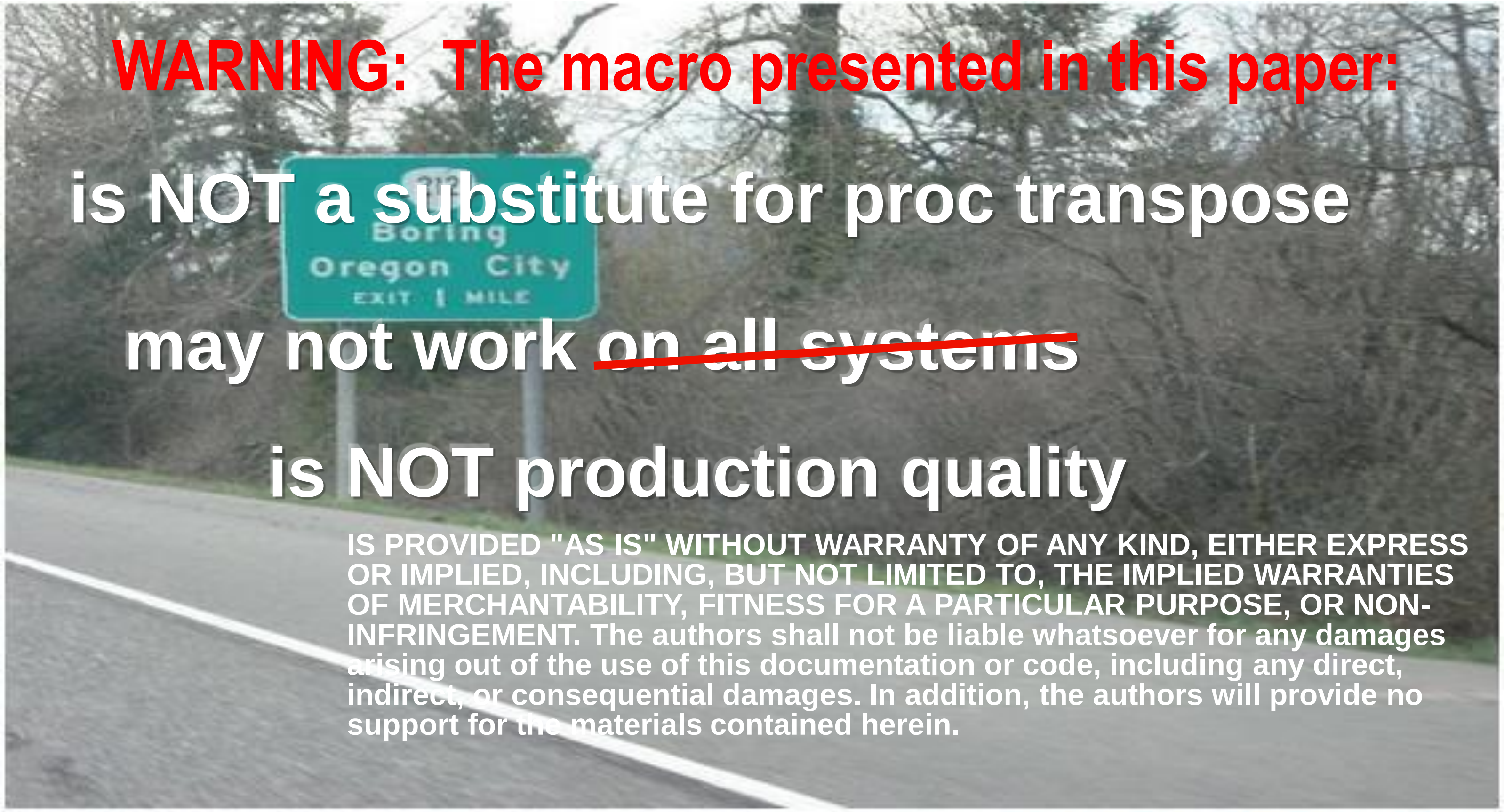
Benefits of the approach

- less typing thus fewer errors
- contains some features that would be nice to see available with some SAS procs
- easier to learn than PROC TRANSPOSE
- runs faster than PROC TRANSPOSE

Benefits of the approach

- less typing thus fewer errors
- contains some features that would be nice to see available with some SAS procs
- easier to learn than PROC TRANSPOSE
- runs faster than PROC TRANSPOSE
- more likely to provide the desired results

our Truth in Advertising commitment



WARNING: The macro presented in this paper:

is NOT a substitute for proc transpose

may not work ~~on all systems~~

is NOT production quality

IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. The authors shall not be liable whatsoever for any damages arising out of the use of this documentation or code, including any direct, indirect, or consequential damages. In addition, the authors will provide no support for the materials contained herein.

Photo Credit: Chris Phan

How the %transpose macro works

First, of course, all of the parameters are declared with any desired default values:

```
%transpose( libname_in=,          libname_out=,
            data=have,            out=want,
            by=,                  prefix=,
            var=,                 autovars=,
            id=,                  var_first=,
            format=,              delimiter=_,
            copy=,                drop=,
            sort=yes,             sort_options=,
            use_varname=,         preloadfmt=,
            guessingrows=1000)
```

Then two-level filenames are parsed and, if necessary, default values assigned to libname_in and libname_out

```
%if %sysfunc(countw(&data.)) eq 2 %then %do;
  %let libname_in=%scan(&data.,1);
  %let data=%scan(&data.,2);
%end;
%else %if %length(&libname_in.) eq 0 %then %do;
  %let libname_in=work;
%end;

%if %sysfunc(countw(&out.)) eq 2 %then %do;
  %let libname_out=%scan(&out.,1);
  %let out=%scan(&out.,2);
%end;
%else %if %length(&libname_out.) eq 0 %then %do;
  %let libname_out=work;
%end;
```

the last (right most) *by* variable is selected to account for users entering more than one *by* variable

```
/*obtain last by variable*/
%if %length(&by.) gt 0 %then %do;
  %let lastby=%scan(&by.,-1);
%end;
%else %do;
  %let lastby=;
%end;
```

Ensure a format is assigned to an *id* variable

```
%if %length(&id.) gt 0 %then %do;
  proc sql noprint;
    select type,length,%sysfunc(strip(format))
      into :tr_macro_type, :tr_macro_len, :tr_macro_format
       from dictionary.columns
         where libname="%sysfunc(upcase(&libname_in.))" and
                  memname="%sysfunc(upcase(&data.))" and
                  upcase(name)="%sysfunc(upcase(&id.))";
  quit;
  %if %length(&format.) eq 0 %then %do;
    %if &tr_macro_format. ne %then %do;
      %let format=&tr_macro_format.;
    %end;
    %else %if "&tr_macro_type." eq "num " %then %do;
      %let format=%sysfunc(catt(best,&tr_macro_len.,.));
    %end;
    %else %do;
      %let format=%sysfunc(catt($,&tr_macro_len.,.));
    %end;
  %end;
%end;
```

Ensure a format is assigned to an *id* variable

```
%if %length(&id.) gt 0 %then %do;
  proc sql noprint;
    select type,length,%sysfunc(strip(format))
      into :tr_macro_type, :tr_macro_len, :tr_macro_format
       from dictionary.columns
         where libname="%sysfunc(upcase(&libname_in.))" and
                  memname="%sysfunc(upcase(&data.))" and
                  upcase(name)="%sysfunc(upcase(&id.))";
  quit;
  %if %length(&format.) eq 0 %then %do;
    %if &tr_macro_format. ne %then %do;
      %let format=&tr_macro_format.;
    %end;
    %else %if "&tr_macro_type." eq "num " %then %do;
      %let format=%sysfunc(catt(best,&tr_macro_len.,.));
    %end;
    %else %do;
      %let format=%sysfunc(catt($,&tr_macro_len.,.));
    %end;
  %end;
%end;
```

Ensure a format is assigned to an *id* variable

```
%if %length(&id.) gt 0 %then %do;
  proc sql noprint;
    select type,length,%sysfunc(strip(format))
      into :tr_macro_type, :tr_macro_len, :tr_macro_format
       from dictionary.columns
         where libname="%sysfunc(upcase(&libname_in.))" and
                  memname="%sysfunc(upcase(&data.))" and
                  upcase(name)="%sysfunc(upcase(&id.))";
  quit;
  %if %length(&format.) eq 0 %then %do;
    %if &tr_macro_format. ne %then %do;
      %let format=&tr_macro_format.;
    %end;
    %else %if "&tr_macro_type." eq "num " %then %do;
      %let format=%sysfunc(catt(best,&tr_macro_len.,.));
    %end;
    %else %do;
      %let format=%sysfunc(catt($,&tr_macro_len.,.));
    %end;
  %end;
%end;
```

Ensure a format is assigned to an *id* variable

```
%if %length(&id.) gt 0 %then %do;
  proc sql noprint;
    select type,length,%sysfunc(strip(format))
      into :tr_macro_type, :tr_macro_len, :tr_macro_format
       from dictionary.columns
        where libname="%sysfunc(upcase(&libname_in.))" and
                 memname="%sysfunc(upcase(&data.))" and
                 upcase(name)="%sysfunc(upcase(&id.))";
  quit;
  %if %length(&format.) eq 0 %then %do;
    %if &tr_macro_format. ne %then %do;
      %let format=&tr_macro_format.;
    %end;
    %else %if "&tr_macro_type." eq "num " %then %do;
      %let format=%sysfunc(catt(best,&tr_macro_len.,.));
    %end;
    %else %do;
      %let format=%sysfunc(catt($,&tr_macro_len.,.));
    %end;
  %end;
%end;
```

Then the copy variables are parsed:

Datastep is used to convert variable list into a macro variable space delimited list

```
%let to_copy=;
 %if %length(&copy.) gt 0 %then %do;
   data _temp;
     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="_TEMP";
     quit;
 %end;
```

Then the copy variables are parsed:

Datastep is used to convert variable list into a macro variable space delimited list

```
%let to_copy=;
 %if %length(&copy.) gt 0 %then %do;
   data _temp;
     set &libname_in..&data. (obs=1 keep=&copy.);
   run;
```

a datastep keep statement will allow any space separated combination of variable names, numbered range lists (e.g. var1-var5), name range lists (e.g. name- -weight), name prefix lists (e.g. var:) and special SAS name lists (e.g. _NUMERIC_)

```
 %
```

Then the copy variables are parsed:

Datastep is used to convert variable list into a macro variable space delimited list

```
%let to_copy=;
 %if %length(&copy.) gt 0 %then %do;
   data _temp;
     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="_TEMP";
     quit;
 %end;
```

If &var is null, the same method is used to populate *&var* macro variable based on the value of *&autovars*

```
%if %length(&var.) eq 0 %then %do;
  data _temp;
    set &libname_in..&data. (obs=1 drop=&by. &id. &copy.);
  run;
  proc sql noprint;
    select name
      into :var separated by " "
       from dictionary.columns
         where libname="WORK" and
                    memname="_TEMP"
 %if %sysfunc(upcase("&autovars.")) eq "CHAR" %then %do;
                   and type="char"
 %end;
 %else %if %sysfunc(upcase("&autovars.")) ne "ALL" %then %do;
                   and type="num"
 %end;
         ;
  quit;
%end;
```

then, guessingrows is checked and variables initialized

```
%if %length(&guessingrows.) eq 0 %then %do;
   %let guessingrows=%sysfunc(constant(EXACTINT));
%end;

%le
%let varlist_char=;
%let vars_num=;
%let varlist_num=;
%let formats_char=;
 %let format_char=;
 %let formats_num=;
 %let format_num=;
```

on Windows XP, running SAS 9.3, that will set guessingrows to equal 9,007,199,254,740,992

Create macro variables containing var names & formats

```
data _temp;
    set &libname_in..&data. (obs=1 keep=&var.);
run;

proc sql noprint;
  select name, case
         when missing(format) then " $"||strip(put(length,5.))||'.'
         else strip(format)
        end
      into :vars_char separated by " ",
              :formats_char separated by "~"
        from dictionary.columns
          where libname="WORK" and
                memname="_TEMP" and
                type="char"
    ;
```

Create macro variables containing var names & formats

proc sql (continued) for Numeric Vars

```
  select name, case
                     when missing(format) then "best12."
                     else strip(format)
                    end
      into :vars_num separated by " ",
              :formats_num separated by "~"
        from dictionary.columns
          where libname="WORK" and
                memname="_TEMP" and
                type="num"
    ;
quit;
```

Then, if the *sort* parameter eq YES, the *&data* file is sorted and output as *work._temp*

```
%if %sysfunc(upcase("&sort.")) eq "YES" %then %do;
   %let notsorted=;
   proc sort data=&libname_in..&data.
               (keep=&by. &id. &vars_char.
                          &vars_num.  &to_copy.)
                out=_temp  &sort_options. noequals;
     by &by.;
   run;
   %let data=_temp;
   %let libname_in=work;
 %end;
 %else %do;
   %let notsorted=notsorted;
 %end;
```

If &sort eq YES, then proc sort is run using *keep* and *noequals* options, along with any options specified in the sort_options parameter

check to see if &preloadfmt was specified and, if so, ensure libname is assigned to the file

```
%if %length(&preloadfmt.) gt 0 %then %do;
   %if %sysfunc(countw(&preloadfmt.)) eq 1 %then %do;
     %let preloadfmt=&libname_in..&preloadfmt.;
   %end;
 %end;
```

Create macro variables containing names and formats of the requested transposed variables uses 3 macro loops

```
proc sql noprint;
%do i=1 %to 2;
  %if &i. eq 1 %then %let i_type=char;
  %else %let i_type=num;
  %if %length(&&vars_&i_type.) gt 0 %then %do;
    select distinct
    %do j=1 %to 2;
      %if &j. eq 1 %then %let j_type=;
      %else %let j_type=format;
      %do k=1 %to %sysfunc(countw(&&vars_&i_type.));
         " &j_type. "||"&prefix."||
```

Create macro variables containing names and formats of the requested transposed variables uses 3 macro loops

One loop to repeat same process for character, then numeric variables

```
proc sql noprint;
%do i=1 %to 2;
 %if &i. eq 1 %then %let i_type=char;
 %else %let i_type=num;
 %if %length(&&vars_&i_type.) gt 0 %then %do;
   select distinct
   %do j=1 %to 2;
    %if &j. eq 1 %then %let j_type=;
    %else %let j_type=format;
    %do k=1 %to %sysfunc(countw(&&vars_&i_type.));
       " &j_type. "||"&prefix."||
```

Create macro variables containing names and formats of the requested transposed variables uses 3 macro loops

```
proc sql noprint;
%do i=1 %to 2;
 %if &i. eq 1 %then %let i_type=char;
 %else %let i_type=num;
 %if %length(&&vars_&i_type.) gt 0 %then %do;
   select distinct
   %do j=1 %to 2;
    %if &j. eq 1 %then %let j_type=;
    %else %let j_type=format;
    %do k=1 %to %sysfunc(countw(&&vars_&i_type.));
         " &j_type. "||"&prefix."||
```

a 2nd loop to repeat same process for variable names, then formats

Create macro variables containing names and formats of the requested transposed variables uses 3 macro loops

```
proc sql noprint;
%do i=1 %to 2;
 %if &i. eq 1 %then %let i_type=char;
 %else %let i_type=num;
 %if %length(&&vars_&i_type.) gt 0 %then %do;
   select distinct
   %do j=1 %to 2;
    %if &j. eq 1 %then %let j_type=;
    %else %let j_type=format;
    %do k=1 %to %sysfunc(countw(&&vars_&i_type.));
         " &j_type. "||"&prefix."||
```

a 3rd loop to repeat same process for all vars

Create macro variables containing names and formats of the requested transposed variables (continued: when var_first is set to NO)

```
%if %sysfunc(upcase("&var_first.")) eq "NO" %then %do;
     strip(put(&id.,&format))||"&delimiter."
  %if &k. lt %sysfunc(countw(&&vars_&i_type.)) %then ||;
  %if %sysfunc(upcase("&use_varname.")) ne "NO" %then %do;
    %if &k. ge %sysfunc(countw(&&vars_&i_type.)) %then ||;
     strip(scan("&&vars_&i_type.",&k.))
    %if &k. lt %sysfunc(countw(&&vars_&i_type.)) %then ||;
  %end;
%end;
```

Create macro variables containing names and formats of the requested transposed variables (continued: when var_first is NOT set to NO)

```
 %else %do;
   %if %sysfunc(upcase("&use_varname.")) ne "NO" %then
      strip(scan("&&vars_&i_type.",&k.))||;
      "&delimiter."||strip(put(&id.,&format))
   %end;
   %if &j. eq 2 %then
      ||" "||strip(scan("&&formats_&i_type.",&k.,"~"))||";";
   %if &k. lt %sysfunc(countw(&&vars_&i_type.)) %then ||;
   %else ,;
 %end;
%end;
     &id.
```

end of 3rd loop

Create macro variables containing names and formats of the requested transposed variables

```
    %if "&tr_macro_type." eq "num " %then &id. format=best12.;
    %else &id.;
     %if %length(
     into :varlist_
            :format_
            :idlist se
     %if %length(&preloadfmt.) gt 0 %then %do;
            ,:idorder separated by " "
      from &preloadfmt.
           order by order
     %end;
     %else %do;
      from &libname_in..&data. (obs=&guessingrows. keep=&id.)
         order by &id.
     %end;
   ;
   %let num_n
   %end;
 %end;
 quit;
```

if &preloadfmt is specified that file is used to obtain variable names and info Otherwise, the info is obtained from &data.

the guessingrows parameter is used so that only the necessary number of records need to be read

Create macro variables containing names and formats of the requested transposed variables (which creates:)

&varlist_char: var2_Qtr1 var2_Qtr2 var2_Qtr3 var2_Qtr4

&varlist_num: var1_Qtr1 var1_Qtr2 var1_Qtr3 var1_Qtr4

&format_char: format var2_Qtr1 $2.; format var2_Qtr2 $2.; format var2_Qtr3 $2.; format var2_Qtr4 $2.;

&format_num: format var1_Qtr1 best12.; format var1_Qtr2 best12.; format var1_Qtr3 best12.; format var1_Qtr4 best12.;

&idlist: Qtr1 Qtr2 Qtr3 Qtr4

&idorder: 1 2 3 4

&num_numlabels: 4

Create format to assign values in correct order

```
%if %length(&preloadfmt.) eq 0 %then %do;
 data _for_format;
   %do i=1 %to &num_numlabels.;
    start=%sysfunc(quote(%scan(&idlist,&i)));
    output;
  %end;
 run;
%end;
data _for_format;
    %if %length(&preloadfmt.) gt 0 %then
    set &preloadfmt. (rename=(&id.=start));
    %else set _for_format;  ;
    %if "&tr_macro_type." eq "num " %then
    retain fmtname "labelfmt" type "N";
    %else retain fmtname "$labelfmt" type "C";   ;
    label= %if %length(&preloadfmt.) eq 0 %then _n_-1;
     %else order-1;   ;
run;
```

Create format to assign values in correct order

```
%if %length(&preloadfmt.) eq 0 %then %do;
 data _for_format;
   %do i=1 %to &num_numlabels.;
    start=%sysfunc(quote(%scan(&idlist,&i)));
    output;
  %end;
 run;
%end;
data _for_form
    %if %length
    set &preloadfmt. (rename=(&id.=start));
    %else set _for_format;  ;
    %if "&tr_macro_type." eq "num " %then
    retain fmtname "labelfmt" type "N";
    %else retain fmtname "$labelfmt" type "C";   ;
    label= %if %length(&preloadfmt.) eq 0 %then _n_-1;
     %else order-1;   ;
run;
```

file is created from &num_numlabels and &idlist

Create format to assign values in correct order

```
%if %length(&preloadfmt.) eq 0 %then %do;
 data _for_format;
   %do i=1 %to
    start=%sysfu
    output;
  %end;
 run;
%end;
data _for_format;
    %if %length(&preloadfmt.) gt 0 %then
    set &preloadfmt. (rename=(&id.=start));
    %else set _for_format;   ;
    %if "&tr_macro_type." eq "num " %then
    retain fmtname "labelfmt" type "N";
    %else retain fmtname "$labelfmt" type "C";   ;
    label= %if %length(&preloadfmt.) eq 0 %then _n_-1;
     %else order-1;   ;
run;
```

a datastep is used to prepare the file for proc format, including setting the ordered position as the format's labels

Create format to assign values in correct order

then the format is created

```
proc format cntlin = _for_format;
run ;
```

finally, a datastep is used to run the job

```
data &libname_out..&out.;
  set &libname_in..&data. (keep=&by. &id.
    %do i=1 %to %sysfunc(countw("&vars_char."));
      %scan(&vars_char.,&i.)
    %end;
    %do i=1 %to %sysfunc(countw("&vars_num."));
      %scan(&vars_num.,&i.)
    %end;
    %do i=1 %to %sysfunc(countw("&to_copy."));
      %scan(&to_copy.,&i.)
    %end;
    );
  by &by. &notsorted.;
  &format_char. &format_num.
```

finally, a datastep is used to run the job

```
data &libname_out..&out.;
  se
   q
   q
   q
   q
   q
   q
  by &by. &notsorted.;
  &format_char. &format_num.
```

which creates

```
data work.want;
  set work.have (keep=idnum date var2 var1);
  by idnum;
  format var2_Qtr1  $8.;
  format var2_Qtr2  $8.;
  format var2_Qtr3  $8.;
  format var2_Qtr4  $8.;
  format var1_Qtr1  best12.;
  format var1_Qtr2  best12.;
  format var1_Qtr3  best12.;
  format var1_Qtr4  best12.;
```

finally, a datastep is used to run the job (Continued)

```
%if %length(&vars_char.) gt 0 %then %do;
   array want_char(*) $
   %do i=1 %to
       %eval(&num_numlabels.*%sysfunc(countw("&vars_char.")));
     %scan(&varlist_char.,&i.)
   %end;
   ;
   array have_char(*) $ &vars_char.;
   retain want_char;
   if first.&lastby. then call missing(of want_char(*));
   ___nchar=put(&id.,labelfmt.)*dim(have_char);
   do ___i=1 to dim(have_char);
     want_char(___nchar+___i)=have_char(___i);
   end;
 %end;
```

finally, a datastep is used to run the job (Continued)

```
%if %length(&vars_char.) gt 0 %then %do;
   array want_char(*) $
  %d
      %                                                                ar.")));
    %
  %er
  ;
   arr
   reta
   if fi
   ___
   do
     want_char(___nchar+___i)=have_char(___i);
   end;
 %end;
```

which creates

```
array want_chr(*) $ var2_Qtr1 var2_Qtr2
                    var2_Qtr3 var2_Qtr4;
array have_chr(*) $ var2;
retain want_chr;
if first.idnum then call missing(of want_chr(*));
___nchar=put(date,labelfmt.)*dim(have_chr);
do ___i=1 to dim(have_chr);
  want_chr(___nchar+___i)=have_chr(___i);
end;
```

finally, a datastep is used to run the job (Continued)

```
%if %length(&vars_num.) gt 0 %then %do;
   array want_num(*)
   %do i=1 %to
       %eval(&num_numlabels.*%sysfunc(countw("&vars_num.")));
     %scan(&varlist_num.,&i.)
   %end;
   ;
   array have_num(*) &vars_num.;
   retain want_num;
   if first.&lastby. then call missing(of want_num(*));
   ___nnum=put(&id.,labelfmt.)*dim(have_num);
   do ___i=1 to dim(have_num);
     want_num(___nnum+___i)=have_num(___i);
   end;
%end;
   drop &id. ___: &var. &drop.;
   if last.&lastby. then output;
 run;
```

finally, a datastep is used to run the job (Continued)

```
%if %length(&vars_num.) gt 0 %then %do;
   array want_num(*)
   %do i=1 %to
      %eval(&num_numlabels.*%sysfunc(countw("&vars_num.")));
     %sc
   %end;
   ;
   array
   retain
   if firs
   ____n
   do __
     wan
   end;
%end;
   drop
   if last.&lastby. then output;
 run;
```

which creates

```
  array want_num(*) var1_Qtr1 var1_Qtr2
                    var1_Qtr3 var1_Qtr4;
  array have_num(*) var1;
  retain want_num;
  if first.idnum then call missing(of want_num(*));
  ____nnum=put(date,labelfmt.)*dim(have_num);
  do ____i=1 to dim(have_num);
     want_num(____nnum+____i)=have_num(____i);
  end;
  drop date ____: var1-var2 ;
  if last.idnum then output;
run;
```

final cleanup

```
 /*Delete all temporary files*/
  proc delete data=work._temp work._for_format;
  run;

%mend transpose;
options NOQUOTELENMAX;
```

an example showing how to run the macro

if you have work.have :

idnum	date	var1	var2
1	31MAR2013	1	SD
1	30JUN2013	2	EF
1	30SEP2013	3	HK
1	31DEC2013	4	HL
2	31MAR2013	5	GH
2	30JUN2013	6	MM
2	30SEP2013	7	JH
2	31DEC2013	8	MS

an example showing how to run the macro

if you have work.have :

idnum	date	var1	var2
1	31MAR2013	1	SD
1	30JUN2013	2	EF
1	30SEP2013	3	HK
1	31DEC2013	4	HL
2	31MAR2013	5	GH
2	30JUN2013	6	MM
2	30SEP2013	7	JH
2	31DEC2013	8	MS

and you need work.need .

idnum	var1 Qtr1	var2 Qtr1	var1 Qtr2	var2 Qtr2	var1 Qtr3	var2 Qtr3	var1 Qtr4	var2 Qtr4
1	1	SD	2	EF	3	HK	4	HL
2	5	GH	6	MM	7	JH	8	MS

an example showing how to run the macro

if you have work.hav

idnum	date	var1	var2
1	31MAR2013	1	SD
1	30JUN2013	2	EF
1	30SEP2013	3	HK
1	01DEC2013	4	HL

run:

```
options NOQUOTELENMAX;
%transpose(data=have,          out=need,
           by=idnum,           id=date,
           format=Qtr1.,       var=var1-var2,
           delimiter=_Qtr,     sort=yes)
```

and you need work.need

idnum	var1 Qtr1	var2 Qtr1	var1 Qtr2	var2 Qtr2	var1 Qtr3	var2 Qtr3	var1 Qtr4	var2 Qtr4
1	1	SD	2	EF	3	HK	4	HL
2	5	GH	6	MM	7	JH	8	MS

an example showing how to run the macro

NOQUOTELENMAX option will be NEEDED if more than 262 characters are needed to hold the collection of transposed variable names

if you have work.have

run:

```
options NOQUOTELENMAX;
%transpose(data=have,          out=need,
           by=idnum,           id=date,
           format=Qtr1.,       var=var1-var2,
           delimiter=_Qtr,     sort=yes)
```

and you need

idnum	var1 Qtr1	var2 Qtr1	var1 Qtr2	var2 Qtr2	var1 Qtr3	var2 Qtr3	var1 Qtr4	var2 Qtr4
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	K	4	HL
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	8	MS

the named parameters ONLY have to be included if needed and AREN'T the default values declared in your macro

Potential Applications

- transposing files more easily
- overcoming system resource constraints in working with large files
- spending less time on data transposition tasks
- using the macro as a template to create similar macros for incorporating sort and keep options for other SAS procs that could benefit from the approach

Presentation Overview

- What the %transpose macro is ✓
- The macro's benefits ✓
- How you would use the macro ✓
- How the macro works ✓
- Potential applications ✓

The macro, paper and Powerpoint can be found at:

http://www.sascommunity.org/wiki/User:Art297

Questions?

Your comments and questions are valued and encouraged

Contact the Authors

Arthur Tabachneck, Ph.D.
President, myQNA, Inc.
Thornhill, ON
art297@rogers.com

Joe Whitehurst
High Impact Technologies
Atlanta, GA
joewhitehurst@gmail.com

Robert Virgile
Robert Virgile Associates, Inc.
Lexington, MA
rvirgile@verizon.net

Xia Ke Shan
Beijing, China
xiakeshan@yahoo.com.cn