PharmaSUG 2017 - Paper TT13

# The Proc Transpose Cookbook

Douglas Zirbel, Wells Fargo and Co.

## ABSTRACT

Proc TRANSPOSE rearranges columns and rows of SAS datasets, but its documentation and behavior can be difficult to comprehend. For common input files, this paper will show a variety of desired output files, plus code and explanations.

## INTRODUCTION

Do you have a SAS® file in one layout of rows and columns – but you need them in a different arrangement? Maybe you have values for Heart Rate in one column with different rows for each patient's 5 visits (vertical) – but you want one row for each patient with all 5 visits' Heart Rates listed as variables (horizontal)? Or vice versa? Or maybe it's financial data, or demographic, or any other kind… and any number of other column/row structures? The TRANSPOSE Procedure is there to solve it: it flips columns and rows of SAS datasets as needed, faster and simpler than you can with a DATA Step. Yet its documentation and behavior are difficult to comprehend.

This paper will show you how to turn SAS File "A" into SAS File "B" with Proc TRANSPOSE. Its SAS File "A" examples are simple, but represent real-world SAS file structures. The transposed File "B"s also represent typical transposition needs. A TRANSPOSE Samples Lookup Table, a Syntax Summary, and the full sample SAS code are located in the Appendix for reference.

## BASICS

Whenever I've had to switch columns and rows in SAS datasets, it has always been under the pressure of a deadline – or more typically, "ASAP". I would look at the SAS documentation page, try to figure it out, do a few tests, and then get something working. Yet I never felt I understood what I was doing, nor was I confident that I could do it again later -- without going through the process all over again.

I needed a cheat sheet. I needed sample code plus before-and-after pictures. It needed to capture 90% of what I'm likely to do, and the principles I need for the rest. The results are in this paper. We'll go through 10 examples of very simplified typical datasets that need to be transposed, and there are reference materials that you can go back to in the future.

You already know that Proc TRANSPOSE changes the shape of your SAS dataset. It can make *values in the rows* for a variable become new *variables* and conversely, it can move *variables* from their horizontal positions into *values in rows*.

We'll start with this example – a small vertical dataset that you want to re-arrange horizontally. Suppose there are only two subjects and four dates which are, for now, just character variables. What you really want is a file that has subject in the first column, but the weights (for the four dates) as their own columns.

**What you have**

| subject | date | weight |
|---|---|---|
| Brittany | jan31 | 145 |
| Ann | jan31 | 153 |
| Brittany | feb28 | 146 |
| Ann | feb28 | 151 |
| Brittany | mar31 | 144 |
| Ann (etc..) | mar31 | 150 |

**What you want**

| subject | jan31 | feb28 | mar31 | apr30 |
|---|---|---|---|---|
| Ann | 153 | 151 | 150 | 152 |
| Brittany | 145 | 146 | 144 | 142 |

This is what we'll be looking at throughout this paper – pictures of what you have, and what you want. You should then be able to translate the examples to your own dataset situation and get what you want.

## VERY SIMPLE PROC TRANSPOSES

TRANSPOSE has a default behavior, like most SAS procs. If you don't tell it what to do, but simply name the input and output datasets, it will turn all *numeric* columns for a row into a single column, and value of numeric columns into new variables. When it's done, it adds a _NAME_ variable to the transposed file so that you can see where the values came from. Finally, unlike some other Procs, TRANSPOSE does not print its output.

**1) What you have**

**1.0 File: work.one_row**

| Obs | count | weight | length | width |
|---|---|---|---|---|
| 1 | 1 | 2 | 4 | 8 |

**What you get**

**1.1 File: work.transp_one_row**

| Obs | _NAME_ | COL1 |
|---|---|---|
| 1 | count | 1 |
| 2 | weight | 2 |
| 3 | length | 4 |
| 4 | width | 8 |

```
proc transpose data=work.one_row
               out=work.transp_one_row;
run;
```

Note that TRANSPOSE also chooses a name for the new column -- *COL1*. And if you had 2 or more rows before transposing, you'd get COL1, COL2... COLn in the output.

If the situation were reversed, where the input file had one column with multiple rows, your TRANSPOSE result would be one row with many columns.

**2) What you have**

**2.0 File: work.one_column**

| Obs | count |
|---|---|
| 1 | 3 |
| 2 | 7 |
| 3 | 4 |
| 4 | 1 |

**What you get**

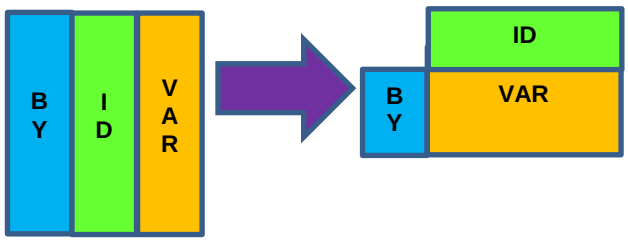
**2.1 File: work.transp_one_column**

| Obs | _NAME_ | COL1 | COL2 | COL3 | COL4 |
|---|---|---|---|---|---|
| 1 | count | 3 | 7 | 4 | 1 |

```
proc transpose data=work.one_column
               out=work.transp_one_column;
run;
```

What happens if you have more than one column in the input? You will get multiple rows in the output, where each row will have the _NAME_ of the original column from the input.

Most of the time we want something more customized though, and TRANSPOSE provides several Proc options and statements. The main statements are VAR, BY, and ID. We'll get into specifics, but for the moment, here is a diagram. We will use this BY, ID, and VAR highlighting in this paper:




**BY** is just like BY in SORT or PRINT, etc;
**ID** column *values* become new col *names*;
**VAR** values are flipped 90° to fit the new structure.

**Figure 1. BY, ID, and VAR relationships**

## COOKBOOK SITUATIONS

**3) What you have**
A by-group (**person**)
A var to transpose: **count**

**3.0 File: work.by_var_and_transp_var**

| Obs | person | count |
|---|---|---|
| 1 | sue | 3 |
| 2 | sue | 7 |
| 3 | ted | 4 |
| 4 | ted | 1 |

**What you want (BY)**
One row per person, **count**s as new columns

**3.1 File: work.transp_by_var_and_transp_var**

| Obs | person | COL1 | COL2 |
|---|---|---|---|
| 1 | sue | 3 | 7 |
| 2 | ted | 4 | 1 |

```
proc sort data=work.by_var_and_transp_var; /** You need to SORT first!! **/
    by person;
run;

proc transpose data=work.by_var_and_transp_var
               out=work.transp_by_var_and_transp_var (drop=_NAME_);
    by person;
    var count;
run;
```

The BY variable should be a discrete, or categorical, variable – that is, with a limited number of values for the dataset. Why? Because your by-group is your main grouping of rows and you want a smaller, manageable number of them.

The _NAME_ of the variable that was transposed doesn't really help us in the output so we drop it.

**4) What you have**
A var w/values for new col names (**week**)
Vars to transpose: **count, weight**

**4.0 File: work.id_var_and_transp_vars**

| Obs | week | count | weight |
|---|---|---|---|
| 1 | 1 | 10 | 15 |
| 2 | 2 | 20 | 25 |
| 3 | 3 | 30 | 35 |
| 4 | 4 | 40 | 45 |

**What you want (ID)**
One row per transposed variable, and one new column for each ID value (**week**)

**4.1 File: work.transp_id_var_and_transp_vars**

| Obs | _NAME_ | _1 | _2 | _3 | _4 |
|---|---|---|---|---|---|
| 1 | count | 10 | 20 | 30 | 40 |
| 2 | weight | 15 | 25 | 35 | 45 |

```
proc transpose data=work.id_var_and_transp_vars
               out=work.transp_id_var_and_transp_vars;
    id week;
    var count weight;
run;
```

Note that the new columns are named with the **week** number, but automatically prefixed with "_". Note also, the ID variable (like BY) should be discrete/categorical. Why? Because the ID values will become your new column names – and you may not want too many new columns. BY and ID vars are most often character-datatype vars, although they certainly can be numeric.

**5) What you have**
A var w/values for new col names (**week**)
Vars to transpose: **count, weight** (for *dup ID values*, see Appendix Samples 5a)

**What you want (ID and prefix)**
One row per transposed variable, and one new column for each ID value (**week**), ***and better-looking***

**5.0 File: work.id_var_and_transp_vars**

| Obs | week | count | weight |
|---|---|---|---|
| 1 | 1 | 10 | 15 |
| 2 | 2 | 20 | 25 |
| 3 | 3 | 30 | 35 |
| 4 | 4 | 40 | 45 |

**5.1 File: work.transp_id_var_and_transp_vars**

| Obs | measurement | week1 | week2 | week3 | week4 |
|---|---|---|---|---|---|
| 1 | count | 10 | 20 | 30 | 40 |
| 2 | weight | 15 | 25 | 35 | 45 |

```
proc transpose data=work.id_var_and_transp_vars
               out=work.transp_id_var_and_transp_vars (rename=(_NAME_=measurement))
               prefix=week;
    id week;
    var count weight;
run;
```

Important: notice that **week**'s values are unique. That is, there are not two "week" rows with the same value. That (duplicate ID values) would lead to two new columns both named "week2"... can't happen. If this situation occurs in your input data, you could use Proc MEANS/SUMMARY to get average **count**s and **weight**s before running TRANSPOSE – if average values are acceptable to your customer. This is shown in Appendix Samples and Code sections "5a". And for an alternative solution, see example 10.

**6) What you have**
*2 vars w/values to be new col names*: **year, week**
A var to transpose: **weight**

**6.0 File: work.two_id_vars**

| Obs | year | week | weight |
|---|---|---|---|
| 1 | 2016 | 1 | 170 |
| 2 | 2017 | 1 | 172 |
| 3 | 2016 | 4 | 171 |
| 4 | 2017 | 4 | 163 |

**What you want (multiple IDs)**
One row for the transposed var with columns named using **year+week**

**6.1 File: work.transp_two_id_vars**

| Obs | Resolution | yrweek_ 2016_1 | yrweek_ 2016_4 | yrweek_ 2017_1 | yrweek_ 2017_4 |
|---|---|---|---|---|---|
| 1 | weight | 170 | 171 | 172 | 163 |

```
proc transpose data=work.two_id_vars  /** but do a SORT BY year week first **/
               out=work.transp_two_id_vars /**(rename=(_NAME_=resolution)) **/
               name=Resolution
               prefix=yrweek_
               delimiter=_;
    id year week;
    var weight;
run;
```

Note that we can use the name= option to provide a new name for _NAME_, instead of using (rename=.

**7) What you have**
A by-group (**person**)
An ID var w/values to be new col names: **bird**
A var to transpose: **count**

**7.0 File: work.by_var_and_id_var**

| Obs | person | bird | count |
|---|---|---|---|
| 1 | SUE | Sparrow | 30 |
| 2 | SUE | Wren | 32 |
| 3 | TED | Sparrow | 34 |
| 4 | TED | Wren | 39 |
| 5 | TED | Robin | 23 |

**What you want (BY and ID)**
One row per person, and turn an input column's (**bird**) values into the new COLs

**7.1 File: work.transp_by_var_and_id_var**

| Obs | person | Sparrow | Wren | Robin |
|---|---|---|---|---|
| 1 | SUE | 30 | 32 | . |
| 2 | TED | 34 | 39 | 23 |

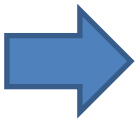
```
proc sort data=work.by_var_and_id_var;
    by person;
run;

proc transpose data=work.by_var_and_id_var
    out=work.transp_by_var_and_id_var (drop=_NAME_);
    by person;
    var count;
    id bird;
run;
```

## THE DOUBLE-TRANSPOSE SITUATION

Do you want to combine ID values with existing column names? You can't do this with a single TRANSPOSE! Note: there are numerous alternative solutions to this. Josh Horstman's SAS Global Forum paper 1266-2014, *Five Ways to Flip-Flop Your Data* presents 5. He illustrates a relevant example:

**CHOLESTEROL_IN**

| SUBJECT | VISIT | LDL | HDL |
|---|---|---|---|
| 1 | 1 | 115 | 33 |
| 1 | 2 | 112 | 43 |
| 2 | 1 | 136 | 51 |
| 2 | 2 | 121 | 50 |
| 3 | 1 | 99 | 57 |
| 3 | 2 | 100 | 59 |



**CHOLESTEROL_OUT**

| SUBJECT | LDL_1 | LDL_2 | HDL_1 | HDL_2 |
|---|---|---|---|---|
| 1 | 115 | 112 | 33 | 43 |
| 2 | 136 | 121 | 51 | 50 |
| 3 | 99 | 100 | 57 | 59 |

**8) What you have** (SAS 9.2 and later)
A by-group (**person**)
An ID var (**week**) w/2+ values to be part of new var names,
one or more vars to transpose:
**wren_count**, **hawk_count**

**8.0 File: work.combine_id_and_vars**

| Obs | person | week | wren_ count | hawk_ count |
|---|---|---|---|---|
| 1 | SUE | 1 | 17 | 3 |
| 2 | SUE | 2 | 23 | 2 |
| 3 | TED | 1 | 5 | 0 |
| 4 | TED | 2 | 14 | 2 |

**What you want (BY and 2 or more *sets* of ID columns)**
One row per BY-group (**person**), and a set of columns for each transposed var (**wren_count, hawk_count**) named with values of the ID var (**week**) + transp vars' names

(Output of second Transpose)

**8.2 File: work.combine_id_and_vars_step2**

| Obs | person | wren_ count_1 | hawk_ count_1 | wren_ count_2 | hawk_ count_2 |
|---|---|---|---|---|---|
| 1 | SUE | 17 | 3 | 23 | 2 |
| 2 | TED | 5 | 0 | 14 | 2 |

```
proc transpose data=work.combine_id_and_vars        /** first SORT BY person week **/
               out=work.combine_id_and_vars_step1;
    by person week;
    var wren_count hawk_count;
run;
```

Output of first Transpose:

**8.1 File: work.combine_id_and_vars_step1**

| Obs | person | week | _NAME_ | COL1 |
|---|---|---|---|---|
| 1 | SUE | 1 | wren_count | 17 |
| 2 | SUE | 1 | hawk_count | 3 |
| 3 | SUE | 2 | wren_count | 23 |

| | | | | |
|---|---|---|---|---|
| 4 | SUE | 2 | hawk_count | 2 |
| 5 | TED | 1 | wren_count | 5 |
| 6 | TED | 1 | hawk_count | 0 |
| 7 | TED | 2 | wren_count | 14 |
| 8 | TED | 2 | hawk_count | 2 |

```
proc transpose data=work.combine_id_and_vars_step1
               out=work.combine_id_and_vars_step2 (drop=_NAME_)
               delimiter=_;
    by person;
    var COL1;        /** tranposing this to multiple columns **/
    id _NAME_ week; /** combining _NAME_ and week values for new column names **/
run;                                Output of this second Transpose is seen in Print 8.2 above
```

## A DOUBLE-TRANSPOSE ALTERNATIVE

**8a) What you have** (SAS pre-9.2)
A by-group (**person**)
An ID var (**week**) w/2+ values to be part of new var names,
one or more vars to transpose:
**wren_count**, **hawk_count**

**8a.0 File: work.by_var_and_2_id_vals**

| Obs | person | week | wren_ count | hawk_ count |
|---|---|---|---|---|
| 1 | SUE | 1 | 17 | 3 |
| 2 | SUE | 2 | 23 | 2 |
| 3 | TED | 1 | 5 | 0 |
| 4 | TED | 2 | 14 | 2 |

**What you want (BY and 2 or more *sets* of ID columns)**
One row per BY-group (**person**), and a set of columns for each transposed var (**wren_count, hawk_count**) named with values of the ID var (**week**) + transp vars' names

**8a.3 File: work.by_var_and_2_id_vals_step2**

| Obs | person | wren_ count_1 | hawk_ count_1 | wren_ count_2 | hawk_ count_2 |
|---|---|---|---|---|---|
| 1 | SUE | 17 | 3 | 23 | 2 |
| 2 | TED | 5 | 0 | 14 | 2 |

```
proc transpose data=work.combine_id_and_vars
               out=work.combine_id_and_vars_step1a (drop=_NAME_)
               prefix=wren_count_;
    by person;
    id week;
    var wren_count;
run;
```

Output of first Transpose:

**8a.1 File: work.combine_id_and_vars_step1a**

| Obs | person | wren_ count_1 | wren_ count_2 |
|---|---|---|---|
| 1 | SUE | 17 | 23 |
| 2 | TED | 5 | 14 |

```
proc transpose data=work.combine_id_and_vars
               out=work.combine_id_and_vars_step2a (drop=_NAME_)
               prefix=hawk_count_;
    by person;
    id week;
    var hawk_count;
run;
```

Output of second Transpose:

**8a.2 File: work.combine_id_and_vars_step2a**

| Obs | person | hawk_ count_1 | hawk_ count_2 |
|---|---|---|---|
| 1 | SUE | 3 | 2 |
| 2 | TED | 0 | 2 |

```
data work.combine_id_and_vars_step3a;
    merge work.combine_id_and_vars_step1a
          work.combine_id_and_vars_step2a;
    by person;
run;
```

Output of this merge (work.combine_id_and_vars_step2a) is identical to Print **8.2** above

## UNUSUAL SITUATIONS: COPY STATEMENT

**9) What you have**

A var to leave un-transposed: **count**

A var to transpose: **count**

**9.0 File: work.copy_and_transp_var**

| Obs | count |
|---|---|
| 1 | 10 |
| 2 | 20 |
| 3 | 30 |
| 4 | 40 |

**What you want (# of output rows = # of input rows)**

a) Transposed var (**count**), and

b) *same number of output rows* as input rows

**9.1 File: work.transp_copy_and_transp_var**

| Obs | count | _NAME_ | COL1 | COL2 | COL3 | COL4 |
|---|---|---|---|---|---|---|
| 1 | 10 | count | 10 | 20 | 30 | 40 |
| 2 | 20 | | . | . | . | . |
| 3 | 30 | | . | . | . | . |
| 4 | 40 | | . | . | . | . |

```
proc transpose data=work.copy_and_transp_var
               out=work.transp_copy_and_transp_var;
    copy count;
    var count;
run;
```

Note that the transposed vars are put only on the first row. If there is a BY group/statement, the transposed vars go on the first row of each BY group row. COPY does not need to be the same as VAR; could be another variable.

## UNUSUAL SITUATIONS: *LET* OPTION (Naming Transposed Variables When ID Has Dup Values):

**10) What you have**

An ID var to be new col names but has duplicate values: **week**

A var to transpose: **wren_count**

**10.0File: work.dup_id_values_LET_option**

| Obs | week | time | wren_ count |
|---|---|---|---|
| 1 | 1 | 8:00 | 17 |
| 2 | 1 | 17:00 | 15 |
| 3 | 2 | 8:00 | 23 |
| 4 | 2 | 17:00 | 20 |
| 5 | 3 | 8:00 | 5 |
| 6 | 3 | 17:00 | 2 |
| 7 | 4 | 8:00 | 2 |
| 8 | 4 | 17:00 | 0 |

**What you want (Only 1 of multiple ID values from input)**

Take the last value of each ID group to be transposed

**10.1 File: work. transp_dup_id_values_LET_option**

| Obs | _NAME_ | week1 | week2 | week3 | week4 |
|---|---|---|---|---|---|
| 1 | wren_count | 15 | 20 | 2 | 0 |

```
proc sort data=work.dup_id_values_LET_option;
    by week time;
run;

proc transpose data=work.dup_id_values_LET_option
                out=work.transp_dup_id_values_LET_option
                prefix=week
                LET;
    id week;
    var wren_count;
run;
```

Because LET takes the *last set* of values for the ID to transpose, make sure that you SORT prior to TRANSPOSE so that you get the right values (in last position) for each by BY group.

## CONCLUSION

Proc TRANSPOSE is an easy way to rearrange rows and columns – if you have examples to refer to. Use those in this paper, and if you have other, interesting TRANSPOSE situations, feel free to send them to me.

## REFERENCES


Horstman, Joshua M, 2014, "Five Ways to Flip-Flop Your Data", *SAS Global Forum Proceedings 2014*

Stuelpner, Janet, 2006, "The TRANSPOSE Procedure or How To Turn It Around", *Proceedings of SAS Users Group International 31*

Tilanus, Erik W., 2007, "Turning the data around: PROC TRANSPOSE and alternative approaches", *SAS Global Forum Proceedings 2007*


## ACKNOWLEDGMENTS


Thanks to Josh Horstman for permission to include a portion of his paper.


## CONTACT INFORMATION

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


Douglas Zirbel
Wells Fargo & Co.
doug_zirbel@hotmail.com or douglas.w.zirbel@wellsfargo.com

## APPENDIX

### SAMPLES LOOKUP TABLE

| | What you have | What you want | What you do |
|---|---|---|---|
| 1 | One column, many rows | Many columns, one row (simple) | Basic/default Transpose |
| 2 | Many columns, one row | Many rows, one column (simple) | Basic/default Transpose |
| | Many columns, many rows | Many rows, many columns (simple) | Basic/default Transpose |
| 3 | A by-group (**person**) var<br>A var to transpose: **count** | One row per person, **count**s as new columns | BY person;<br>VAR count; |
| 4 | Var w/values for new col names (**week**)<br>Vars to transpose: **count, weight** | One row/transposed variable, one new column for each ID value (**week**) | ID week;<br>VAR count weight; |
| 5 | A var w/values for new col names (**week**)<br>Vars to transpose: **count, weight** | One row per transposed variable, and one new column for each ID value (**week**), ***and better-looking*** | prefix=week<br>ID week;<br>VAR count weight; |
| 5a | ID var with dup values: **week**<br>Var to transpose: **count** | One row per transposed var, one new var for each ID value...**but no dup-ID ERRORs**<br>(this works if mean values are acceptable by your customer) | Summarize dup IDs with Proc MEANS first, *then transpose that output*<br>**proc means** data=..NWAY;<br>class week; /*(ID)*/<br>var count;<br>output out=...<br>mean(count)=count;<br>run; |
| *6* | *2 vars w/values to be new col names*: **year, week**<br>A var to transpose: **weight** | One row for the transposed var with columns named using **year+week** | prefix=yrweek_<br>delimiter=_<br>ID year week;<br>VAR weight; |
| 7 | A by-group (**person**)<br>An ID var w/values to be new col names: **bird**<br>A var to transpose: **count** | One row per person, and turn an input column's (**bird**) values into the new COLs | BY person;<br>VAR count;<br>ID bird; |
| 8 | A by-group (**person**)<br>An ID var (**week**) w/2+ values to be part of new var names,<br>one or more vars to transpose:<br>**wren_count**, **hawk_count** | One row per BY-group (**person**), and a set of columns for each transposed var (**wren_count, hawk_count**) named with values of the ID var (**week**) + transp vars' names | ***Double transpose***:<br>BY person week;<br>VAR wren hawk;<br>***then***<br>BY person;<br>VAR COL1;<br>ID _NAME_ week; |
| 8a | (Above, alternative method) | (Above, alternative method) | ***Double transpose***:<br>BY person;<br>ID week;<br>VAR wren_count;<br>***then***<br>BY person;<br>ID week;<br>VAR hawk_count;<br>***Then merge*** |
| 9 | A var to leave un-transposed: **count**<br>A var to transpose: **count** | **What you want (# of output rows = # of input rows)**<br>a) Transposed var (**count**), and<br>b) *same number of output rows* as input rows | COPY count;<br>VAR count; |
| 10 | An ID var to be new col names but has duplicate values: **week**<br>A var to transpose: **wren_count** | Take the last value of each ID group to be transposed | ***Sort first by ID***<br>LET option<br>ID week;<br>VAR wren_count; |

## TRANSPOSE SYNTAX

| PROC TRANSPOSE | Proc name (required); note: unlike some Procs, TRANSPOSE does not print anything |
|---|---|
| **Proc options** | |
| data= | Input dataset (required, otherwise it picks up _last_ dataset) |
| out= | Transposed output file. (required, otherwise it writes output files as work.data#, e.g. work.data1, etc) |
| obs/where/keep/rename… | Many ordinary dataset options are also available after the **data=** or **out=** |
| prefix= | a prefix to use for constructing names for transposed variables in the output data set. For example, if PREFIX=QTCB, then the names of the variables are QTCB1, QTCB2, …, QTCBn; otherwise, transposed columns are COL1, COL2, … COLn |
| suffix= | a suffix to use in creating names for transposed vars in the output dataset |
| name= | the name for the variable in the output data set that *contains the name of the variable that is being transposed* to create the current observation; if you don't use this, the column will automatically be named _**NAME**_ |
| label= | a name for the variable in the output data set that *contains the label of the variable that is being transposed* to create the current observation; like **name=**, if the variable being transposed has a variable label and you don't use this option, the column will be automatically named _**LABEL**_ |
| delimiter= | Provides a delimiter character (e.g. _) to separate new column names – *when the ID statement has more than one variable* (new with SAS 9.2) |
| let | An odd option. It allows duplicate values of an ID variable (used to name the new column(s)). PROC TRANSPOSE transposes the observation that contains the *last occurrence* of a particular ID value within the data set or BY group. You might want this – but be sure to sort your data so that the last occurrence value is the one you want. Try it to see how it works. |
| **Proc Statements** | |
| VAR | Names the variable(s) to be transposed – can be character or numeric; if you don't have a VAR statement, all numeric vars will be transposed |
| ID | A variable whose values will become the new transposed columns – often a numeric value and often used with the **prefix=** option. If it is numeric and no prefix= is given the number will be prefixed with an underscore when it becomes a var name |
| IDLABEL | Can only be used with the ID statement – creates variable labels for the transposed vars by using the value of another column that is related to the ID column. For example, if *studentnum* is unique and you use **ID studentnum;** then **IDLABEL student_last_name;** could provide the student's last names as the column labels to be used with e.g. **proc print data=work.xyz label;**. |
| BY | The BY var(s) is not transposed (flipped) – it is often found in the first column(s); data must be first SORTed by the same BY var(s) before using BY in Proc TRANSPOSE. For each BY-group, the proc creates one row for each var it transposes: for example, if 3 "horizontal" numeric vars for a single BY-group row are transposed, there will be 3 rows with that BY var value in the output (and vice versa):<br>(# of output rows) = (# of BY-groups) * (# of transposed vars) |
| COPY | Another unusual feature. Copies one or more vars directly from the input data set to the output data set without transposing them – the best of both worlds, i.e., you get transposed vars + the original rows; the number of rows in the output dataset is equal to the number of rows in the input dataset, and the first row for each by-group contains transposed values. Try it to see how it works. |
| RUN | A Good SAS Practice (GSP) |
| **Other statements** | |
| attrib | Associates attributes (format, informat, label, length) with variables by changing the descriptor information of the SAS dataset |
| format | Associates formats with variables |
| label | Assigns descriptive labels to variables |
| where | Selects observations from SAS datasets that meet a particular condition |

## CODE USED IN EXAMPLES

```
*******************************************************;
*   CURRENT PROGRAM SUMMARY SECTION ;
*******************************************************;
* Pgm name  : $Id:
Proc_Transpose_Cookbook_Pharmasug_2017_TT13.sas 649 2017-
03-31 18:39:55Z n550513 $ ;
* Curr Descr: Samples for Pharmasug 2017 paper TT13;
*******************************************************;
* Declare print macro, obs= is optional, otherwise max;
*******************************************************;
%macro proc_print(run_yn=y,  data=,  obs=,  example_num=);
    %if %upcase(&run_yn) ne N %then %do;
        %if &obs eq %str( ) %then %do;
            %let obs = max;
            title "&example_num File: &data";
        %end;
        %else %do;
            title "&example_num File: &data (obs=&obs)";
        %end;
        proc print data=&data (obs=&obs)  width=min
                   heading=h;
        run;
    %end;
%mend proc_print;

*******************************;
* 1) Transp 1 row;
*******************************;
data work.one_row;
    infile cards;
    input count weight length width;
    cards;
1 2 4 8
;
run;
%proc_print(data=work.one_row, example_num=1.0);

proc transpose data=work.one_row
               out=work.transp_one_row;
run;
%proc_print(data=work.transp_one_row, example_num=1.1);

*******************************;
* 2) Transp 1 column;
*******************************;
data work.one_column;
    infile cards;
    input count;
    cards;
3
7
4
1
;
run;
%proc_print(data=work.one_column, example_num=2.0);

proc transpose data=work.one_column
               out=work.transp_one_column;
run;
%proc_print(data=work.transp_one_column, example_num=2.1);

*******************************;
* 3) By var + transp var;
*******************************;
data work.by_var_and_transp_var;
    infile cards;
    input person $ count;
    cards;
sue 3
sue 7
ted 4
ted 1
;
run;
%proc_print(data=work.by_var_and_transp_var,
example_num=3.0);

proc sort data=work.by_var_and_transp_var; /** You need to
SORT first!! **/
    by person;
run;

proc transpose data=work.by_var_and_transp_var
               out=work.transp_by_var_and_transp_var
(drop=_NAME_);
    by person;
    var count;
run;
%proc_print(data=work.transp_by_var_and_transp_var,
example_num=3.1);

*******************************;
* 4) ID var + transp var;
*******************************;
data work.id_var_and_transp_vars;
    infile cards;
    input week count weight;
    cards;
1 10 15
2 20 25
3 30 35
4 40 45
;
run;
%proc_print(data=work.id_var_and_transp_vars,
example_num=4.0);

proc transpose data=work.id_var_and_transp_vars
               out=work.transp_id_var_and_transp_vars;
    id week;
    var count weight;
run;
%proc_print(data=work.transp_id_var_and_transp_vars,
example_num=4.1);

*******************************;
* 5) ID var + transp var w/prefix;
*******************************;
data work.id_var_and_transp_vars;
    infile cards;
    input week count weight;
    cards;
1 10 15
2 20 25
3 30 35
4 40 45
;
run;
%proc_print(data=work.id_var_and_transp_vars,
example_num=5.0);

proc transpose data=work.id_var_and_transp_vars
               out=work.transp_id_var_and_transp_vars
(rename=(_NAME_=measurement))
               prefix=week;
    id week;
    var count weight;
run;
%proc_print(data=work.transp_id_var_and_transp_vars,
example_num=5.1);

*******************************;
* 5a) dup ID values           ;
*******************************;
data work.dup_id_values;
    infile cards;
    input week count;
    cards;
1 10
2 20
2 24
3 30
4 40
;
run;
%proc_print(data=work.dup_id_values, example_num=5a.0);

proc means data=work.dup_id_values NWAY NOPRINT;
    class week;
    var count;
    output out=work.means_dup_id_values (drop=_type_
_freq_) mean(count)=count;
run;
%proc_print(data=work.means_dup_id_values,
example_num=5a.1);


proc transpose data=work.means_dup_id_values
               out=work.transp_means_dup_id_values
(rename=(_NAME_=measurement))
               prefix=week;
    id week;
    var count;
run;
%proc_print(data=work.transp_means_dup_id_values,
example_num=5a.2);


*****************************************;
* 6) 2 or more id vars for the new columns;
*****************************************;
data work.two_id_vars;
    infile cards;
    input year week weight ;
    cards;
```

```
2016 1 170
2017 1 172
2016 4 171
2017 4 163
;
run;
%proc_print(data=work.two_id_vars, example_num=6.0);

proc sort data=work.two_id_vars;
    by year week;
run;

proc transpose data=work.two_id_vars
               out=work.transp_two_id_vars
/**(rename=(_NAME_=resolution)) **/
               name=Resolution
               prefix=year_week_
               delimiter=_;
    id year week;
    var weight;
run;
%proc_print(data=work.transp_two_id_vars, example_num=6.1);

****************************************;
* 7) A by var and an id var;
****************************************;
data work.by_var_and_id_var;
    infile cards;
    input person $ bird $ count;
    cards;
SUE Sparrow 30
SUE Wren 32
TED Sparrow 34
TED Wren 39
TED Robin 23
;
run;
%proc_print(data=work.by_var_and_id_var, example_num=7.0);

proc sort data=work.by_var_and_id_var;
    by person;
run;

proc transpose data=work.by_var_and_id_var
    out=work.transp_by_var_and_id_var (drop=_NAME_);
    by person;
    var count;
    id bird;
run;
%proc_print(data=work.transp_by_var_and_id_var,
example_num=7.1);

****************************************;
* 8) A by var and 2 or more id vars;
* (Double-transpose);
****************************************;
data work.combine_id_and_vars;
    infile cards;
    input person $ week wren_count hawk_count;
    cards;
SUE 1 17 3
SUE 2 23 2
TED 1 5 0
TED 2 14 2
;
run;
%proc_print(data=work.combine_id_and_vars,
example_num=8.0);

proc sort data=work.combine_id_and_vars;
    by person week;
run;
proc transpose data=work.combine_id_and_vars
               out=work.combine_id_and_vars_step1;
    by person week;
    var wren_count hawk_count;
run;
%proc_print(data=work.combine_id_and_vars_step1,
example_num=8.1);

proc transpose data=work.combine_id_and_vars_step1
               out=work.combine_id_and_vars_step2
                   (drop=_NAME_)
               delimiter=_;
    by person;
    var COL1; /** tranposing this to multiple columns **/
    id _NAME_ week; /** combining _NAME_ and week for new
column names **/
run;
%proc_print(data=work.combine_id_and_vars_step2,
example_num=8.2);

****************************************;
* 8a) A by var and 2 or more id vars;
* (Double-transpose) - another way (merge);
****************************************;
proc transpose data=work.combine_id_and_vars
               out=work.combine_id_and_vars_step1a
                   (drop=_NAME_)
               prefix=wren_count_;
    by person;
    id week;
    var wren_count;
run;
%proc_print(data=work.combine_id_and_vars_step1a,
example_num=8a.0);

proc transpose data=work.combine_id_and_vars
               out=work.combine_id_and_vars_step2a
                       (drop=_NAME_)
               prefix=hawk_count_;
    by person;
    id week;
    var hawk_count;
run;
%proc_print(data=work.combine_id_and_vars_step2a,
example_num=8a.1);

data work.combine_id_and_vars_step3a;
    merge work.combine_id_and_vars_step1a
          work.combine_id_and_vars_step2a;
    by person;
run;
%proc_print(data=work.combine_id_and_vars_step3a,
example_num=8a.2);
****************************************;
* 9) Unusual - where you want same number  ;
* of output rows as input rows (COPY stmt);
****************************************;
data work.copy_and_transp_var ;
    infile cards;
    input count;
    cards;
10
20
30
40
;
run;
%proc_print(data=work.copy_and_transp_var,
example_num=9.0);

proc transpose data=work.copy_and_transp_var
               out=work.transp_copy_and_transp_var;
    copy count;
    var count;
run;
%proc_print(data=work.transp_copy_and_transp_var,
example_num=9.1);
****************************************;
* 10) Unusual - where you keep only last row;
* for duplicate ID values (LET option);
****************************************;
data work.dup_id_values_LET_option;
    infile cards;
    input week time wren_count;
    informat time time5.;
    format time time5.;
    cards;
1 08:00 17
2 08:00 23
3 08:00 5
4 08:00 2
1 17:00 15
2 17:00 20
3 17:00 2
4 17:00 0
;
run;
proc sort data=work.dup_id_values_LET_option;
    by week time;
run;
%proc_print(data=work.dup_id_values_LET_option,
example_num=10.0);

proc transpose data=work.dup_id_values_LET_option
               out=work.transp_dup_id_values_LET_option
               prefix=week
               LET;
    id week;
    var wren_count;
run;
%proc_print(data=work.transp_dup_id_values_LET_option,
example_num=10.1);
```