Long-to-Wide: PROC TRANSPOSE vs Arrays vs PROC SUMMARY
Mike Zdeb, University at Albany School of Public Health, Rensselaer, NY

ABSTRACT
When data for the same entity (person, place, thing) are stored across multiple observations, it is a common task to place all the data for a given entity into a single observation. The task is sometimes referred to as converting a data set from long-to-wide and the complexity differs depending on whether one or more variables are involved. This paper compares three methods of long-to-wide conversion, two common (PROC TRANSPOSE and arrays) and one not commonly used (PROC SUMMARY). It also shows how PROC SQL can be used to find information needed for both the array and PROC SUMMARY methods. The good and bad points of each method are discussed in both one and many variable situations and the discussion is intended for an audience with a skill level ranging from beginner to intermediate. All the techniques require only base SAS®. (Note some text in this paper and some examples also are used in a previous paper by the author on reshaping and merging data ... see references).

INTRODUCTION
One of the many strengths of SAS software is the number of tools it offers for data organization, or perhaps reorganization is a better term. Prior to using any of the SAS procedures for data analysis or the ODS tools for data presentation, it is not uncommon to spend a lot of time and effort putting data in a form that makes it amenable to analysis and presentation. Sometimes, if you want to conduct an analysis across observations in a data set, you can use SAS procedures. Or, if you want to conduct an analysis within observations, you can use SAS functions. However, your data may be arranged in such a way that the only way to complete a given task is to first rearrange the data set. Rearranging in this context means converting variables into observations or observations into variables. The examples in this paper focus only on converting observations into variables. For example, your data might comprise sets of observations for different individuals who you are following over time to measure the effect of a weight loss regimen. Each observation represents a measurement taken on a different date. If you want to see the change in weight for each person over the study period, your beginning and end points are in different observations and you cannot simply subtract one value from another. However, if you could rearrange your data set to have the weights from all the observations for a given individual in one observation as variables (weight1, weight2, etc.), calculating the change in weight is quite easy.

A series of examples will be used to show three different approaches for turning observations into variables. You will see that each method has both strengths and weaknesses.

PROC TRANSPOSE
You have a SAS data set with observations that contain an account number, a month (in the form of 1 for January, 2 for February, etc.), and a dollar value indicating the amount deposited into money market account in the given month. You would like to create a report showing the monthly deposits for each person in your data set (as indicated by the account number). You think that it would be easier to create the report if the data set had one observation per person with all the deposits in that observation. The following data step creates the data set shown on the right.

* EXAMPLE 1;
 data deposits;
 input account :$2. month deposit @@;
 datalines;
 01 2 100 01 4 50 01 6 200
 02 1 50 02 3 100
 03 1 50 03 2 50 03 3 50 03 4 50 03 5 50 03 6 50
 ;

data set DEPOSITS, example 1
The minimum amount of information needed to run PROC TRANSPOSE is the name of a data set to be rearranged. SAS will create a new data set and give it a name ... DATAx, where x is an integer that is one (DATA1) if this is the first data set you have created in a session with a SAS-assigned name (then DATA2, DATA3, etc.). Rather than have SAS assign a data set name, you can use an OUT= option to name the new data set.

* EXAMPLE 2;
* SAS-supplied name for new data set;
    proc transpose data=deposits;
    run;

* user-supplied name for new data set;
    proc transpose data=deposits out=accounts;
    run;

Here is the LOG file when only an input data set is specified ...

55   proc transpose data=deposits;
56   run;
NOTE: There were 11 observations read from the data set WORK.DEPOSITS.
NOTE: The data set WORK.DATA1 has 2 observations and 12 variables.

The second portion of the code in example 2 produces the data set shown below. There are few things that you should notice: there were 11 observations and now you have 11 variables, automatically named with the prefix "COL" and a numeric suffix of 1 through 11 (observations-to-variables); only values for the two numeric variables in data set DEPOSITS are in data set ACCOUNTS and they are now observations (variables-to-observations); there is a SAS-supplied variable _NAME_ then identifies the variable that is the source of the data within each observation.

<table>
<thead>
<tr>
<th>Obs</th>
<th><em>NAME</em></th>
<th>COL1</th>
<th>COL2</th>
<th>COL3</th>
<th>COL4</th>
<th>COL5</th>
<th>COL6</th>
<th>COL7</th>
<th>COL8</th>
<th>COL9</th>
<th>COL10</th>
<th>COL11</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>month</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>deposit</td>
<td>100</td>
<td>50</td>
<td>200</td>
<td>50</td>
<td>100</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td></td>
</tr>
</tbody>
</table>

data set ACCOUNTS, example 2

You should also ask yourself a few questions: how can I transpose character variables if only numeric variables are transposed by default; how can I change the names of the new variables to something other than COL1, COL2, etc.; how can I get one observation per account number. Assuming that your data are sorted by account number, you can use a BY statement to rearrange the original data one account number at a time. You can use a PREFIX option to change the names of the new variables, in this case from "COL" to "DEP ..."

* EXAMPLE 3;
    proc transpose data=deposits out=accounts prefix=dep;
    by account;
    run;

The data set shown on the right is closer to the desired data set, but there are two observations per account rather than one. You only want the deposits, not the months. The new variables are now named DEP1-DEP6 rather than COL1-COL6. However, notice that numeric suffix for variable DEP has nothing to do with the month in which the deposit was made for accounts 01 and 02. How do you get one observation per account and how are the deposits assigned to the correct month?
The VAR statement can be used to specify the variables that are to be transposed (DEPOSIT). An ID statement specifies a variable (MONTH) whose value controls the variables to which values of DEPOSIT are assigned. The variable _NAME_ is not needed as is dropped using a data set option.

* EXAMPLE 4;
  proc transpose data=deposits out=accounts (drop=_name_) prefix=dep;
  by account;
  var deposit;
  id month;
  run;

The data set shown above on the left is the result of the SAS code in example 4. If you look at the numeric suffix on the "DEP" variables, you can see that the values match the months in which money was deposited into each account. The order of the "DEP" variables is controlled by the order in which PROC TRANSPOSE encounters values of the ID variable (MONTH). The first account (01) made deposits months 2, 4, and 6 so once PROC TRANSPOSE has processed the data for that account, variables DEP2, DEP4, and DEP6 are created. The second account (02) made deposits in months 1 and 3 so variables DEP1 and DEP3 are added to the data set. Finally, the last account (03) adds on more month the variable list, DEP5. The data set shown on the above right displays the deposits in month order and it is a little easier to see that all the deposits have been assigned to the correct months.

One way to remember how the various statements work in using PROC TRANSPOSE to convert observations to variables is to think of the original data as a list and the rearranged data as a spreadsheet. The list contains a variable whose values that you want to place in the cells of the spreadsheet (in the current example, that is DEPOSIT) and that variable is used in the VAR statement. The list also contains a variables that control the row and column (in the current example ACCOUNT and MONTH) for placing the value of the variable in the VAR statement. Row control is done by the value of the variable in the BY statement while column control is done by the variable in the ID statement. If column placement does not matter, no ID statement is necessary and columns with rows will be just filled from left to right.

If an ID statement is used and values of the ID variable are numeric, it is common to use a PREFIX option as shown in examples 3 and 4 to control the naming of the new variables. If no PREFIX option is used with a numeric ID variable, PROC TRANSPOSE adds an underscore as a prefix to each number, converting the numbers to valid SAS variable names.

If an ID statement is used and the values of the ID variable are character and would be allowable variable names, no PREFIX option is necessary. If the data set DEPOSITS had months with values of three-letter month names, the transposed data set would look as shown on the right. When the ID variable is character but with values that are not allowable variable names (for example, with embedded spaces), PROC TRANSPOSE adjusts the variable values to form allowable variable names (in the case of embedded spaces, they are replaced by underscores).
Before leaving PROC TRANSPOSE, we will add a variable to the data set DEPOSITS to show an example where alternative methods might be better for rearranging the data set. The new variable is TYPE and it indicates whether the deposit was made by check (TYPE=1) or cash (TYPE=2). Once again you would like to rearrange the data set so there is one observation for each account, but that observation should contain information as to both the amount and type of deposit. What happens if you add a second variable to the BY statement in PROC TRANSPOSE (example 2 gives an hint as to what will happen) ...

* EXAMPLE 5;
proc transpose data=deposits out=accounts
prefix=month;
by account;
var deposit type;
id month;
run;

In example 5 the PREFIX option now specifies the text "MONTH" and the DROP data step option has been removed. The VAR statement has two variables, DEPOSIT and TYPE. The output from PROC TRANSPOSE shown below has all the information from the original data set but the for each account that information is spread across two observations.

<table>
<thead>
<tr>
<th>Obs</th>
<th>account</th>
<th>deposit</th>
<th>month2</th>
<th>month4</th>
<th>month6</th>
<th>month1</th>
<th>month3</th>
<th>month5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01</td>
<td>deposit</td>
<td>100</td>
<td>50</td>
<td>200</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>2</td>
<td>01</td>
<td>type</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>3</td>
<td>02</td>
<td>deposit</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>50</td>
<td>100</td>
<td>.</td>
</tr>
<tr>
<td>4</td>
<td>02</td>
<td>type</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>2</td>
<td>2</td>
<td>.</td>
</tr>
<tr>
<td>5</td>
<td>03</td>
<td>deposit</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>6</td>
<td>03</td>
<td>type</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

data set ACCOUNTS, two variables in the VAR statement (DEPOSIT and TYPE)

There is a way to get all the information into one observation but it requires a data step to restructure the original data prior to PROC TRANSPOSE. When PROC TRANSPOSE is used, rather than having two variables in the BY statement, two variables will be specified in the ID statement and that is only possible as of version SAS V9.3. The following data step rearranges the data.

* EXAMPLE 6;
data new_deposits (keep=account new1 new2 month);
set deposits;
new1 = 'AMT' ; new2 = deposit; output;
new1 = 'TYP' ; new2 = type; output;
run;

The first six observations (all the data for account 01) in the new data set are shown on the right. Notice that two new variables: NEW1 indicates whether the observation is a deposit amount (AMT) or a deposit type (TYP); NEW2 contains the value of either the deposit or the deposit type.

<table>
<thead>
<tr>
<th>Obs</th>
<th>account</th>
<th>month</th>
<th>new1</th>
<th>new2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01</td>
<td>2</td>
<td>AMT</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>01</td>
<td>2</td>
<td>TYP</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>01</td>
<td>4</td>
<td>AMT</td>
<td>50</td>
</tr>
<tr>
<td>4</td>
<td>01</td>
<td>4</td>
<td>TYP</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>01</td>
<td>6</td>
<td>AMT</td>
<td>200</td>
</tr>
<tr>
<td>6</td>
<td>01</td>
<td>6</td>
<td>TYP</td>
<td>2</td>
</tr>
</tbody>
</table>

data set NEW_DEPOSITS, example 6
Given the rearranged data and the two new variables, PROC TRANSPOSE can produce the desired data set (but only in SAS V9.2 and beyond using the newly introduced ability to specify multiple ID variables).

* EXAMPLE 7;
```
proc transpose data=new_deposits out=accounts (drop=_name_);
by account;
var new2;
id new1 month;
run;
```

The data set shown below has all the variable values in the correct months (note: the order of the variables shown below is not the order of the variables in data set ACCOUNTS but was printed in that order using a VAR statement in PROC PRINT). The values of the variables used in the ID statement have been concatenated to create the names of the new variables. That requires some "thinking ahead" when rearranging the data prior to PROC TRANSPOSE in that the prefixes of the new variable names should give you some indication as to variable content. Also, the order of the variables in the ID statement is important since the variable values are concatenated in that order. If there were many more variables to rearrange prior to PROC TRANSPOSE, the rearranging done with the SAS code in example 6 might be better done with some creative use of an array, some DO loops, and the VNAME function. There is an example in the appendix A.

**ARRAYS**

When using PROC TRANSPOSE to rearrange data, you must use statements and options provided by the procedure. As shown in examples 6 and 7, you can be a bit creative using a data step prior creating the final data set, but the there's a limited amount of creativity allowed when using any procedure. There are no rules when using arrays to rearrange data since the rearranging is dependent on data step programming and that set of skills varies greatly from user-to-user. The examples in this section create the data sets produced in examples 4 and 7, but this time using arrays. Example 8 uses the data set DEPOSITS shown on page 1.

* EXAMPLE 8;
```
data accounts (keep=account dep1-dep6);
array dep(6);
do until (last.account);
  set deposits;
  by account;
  dep(month) = deposit;
end;
run;
```

The new data set will contain only the variables ACCOUNT and DEP1 through DEP6, the same variables in the data set produced in example 4. Notice that you have to have some knowledge as to the largest value of the variable MONTH in data set. In order to write the KEEP data set option, you had to know that the maximum value of MONTH in data set DEPOSITS is 6, something you did not have to know when using PROC TRANSPOSE. That value is also required in the ARRAY statement. A DOW-loop (see REFERENCES) is used to process the observations in the data set one account at a time. The value of the variable DEPOSIT is placed in the correct position in the array DEP using the value of the variable MONTH. As soon as the loop completes (when all the observations for a given account have been read), an observation is written to data set ACCOUNTS. Just as when PROC TRANSPOSE was used with a BY statement, the use of a BY variable within the data step loop also assumes the data set is sorted in ascending order of values of variable ACCOUNT.
What if there was no variable named MONTH and all you wanted to do is place all the deposits for a given account in a single observation without regard to placement in a specific variable location. The next example creates a data set similar to that produced in example 8, but values of the variable DEPOSIT are just added to the array from left to right.

* EXAMPLE 9;
\[
data\ accounts (keep=account dep1-dep6);
array dep(6);
do j = 1 by 1 until (last.account);
  set deposits;
  by account;
  dep(j) = deposit;
end;
run;
\]

The data set on the right looks similar to that produced in example 8, but the numeric suffix on the variables DEP1 through DEP6 does not represent any particular month. The SAS code in example 9 is similar to that used in example 8 with two exceptions: a index variable (J) is used to count passes through the loop \( \Box \); the value of the index variable is used to place the value of the variable DEPOSIT is an array location \( \ast \).

Creating a data set with two transposed variables (DEPOSIT and TYPE) required a data step (example 6) and two ID variables in PROC TRANSPOSE (example 7). The SAS code to transpose two variables using arrays is not much different than that used in example 8 to transpose one variable.

* EXAMPLE 10;
\[
proc sql noprint;
select max(month) into :mm separated by '' from deposits; \( \Box \)
quit;

data accounts (keep=account amt1-amt&mm typ1-typ&mm);
array amt(&mm);
array typ(&mm);
do until (last.account);
  set deposits;
  by account;
  amt(month) = deposit;
  typ(month) = type;
end;
run;
\]

Once again, some knowledge as to the maximum value of the variable MONTH is required to write the SAS data step. Rather than just looking at the data to find that value, PROC SQL is used to determine the maximum value of MONTH and to place that value in a macro variable (&MM) ... the addition of the "SEPARATED BY" clause removes leading spaces from the macro variable. The data step uses the macro variable in the KEEP data set option \( \Box \) and in the ARRAY statements \( \ast \). Within the DOW-loop, values are assigned to the two arrays to produce the data set shown below.

<table>
<thead>
<tr>
<th>Obs</th>
<th>account</th>
<th>dep1</th>
<th>dep2</th>
<th>dep3</th>
<th>dep4</th>
<th>dep5</th>
<th>dep6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01</td>
<td>100</td>
<td>50</td>
<td>200</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>50</td>
<td>100</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
</tbody>
</table>

data set ACCOUNTS, example 9 ... created with an ARRAY

\[
\begin{array}{cccccccccc}
\hline
\text{Obs} & \text{account} & \text{amt1} & \text{amt2} & \text{amt3} & \text{amt4} & \text{amt5} & \text{amt6} & \text{typ1} & \text{typ2} & \text{typ3} & \text{typ4} & \text{typ5} & \text{typ6} \\
\hline
1 & 01 & 100 & . & 50 & . & 200 & . & 1 & . & 1 & . & 2 \\
2 & 02 & . & 100 & . & . & . & 2 & . & 2 & . & . & . \\
3 & 03 & 50 & 50 & 50 & 50 & 50 & 50 & 1 & 1 & 2 & 1 & 1 \\
\hline
\end{array}
\]
data set ACCOUNTS, example 10 ... created with ARRAYS
PROC SQL is used in example 10 to find the maximum value of MONTH in data set DEPOSITS. What if there were no variable MONTH (as shown on the right) and all you wanted to do was produce a data set with all the amounts and types of deposits for each account in one observation. Now you must know the maximum number of observations in an account in data set DEPOSITS. You can still use PROC SQL to determine that number and store it in a macro variable for use in a subsequent data step.

* EXAMPLE 11;
proc sql noprint;
  select max(count) into :mm separated by '' from (
  select count(*) as count from deposits group by account);
quit;

data accounts (keep=account amt1-amt&mm typ1-typ&mm);
array amt(&mm);
array typ(&mm);
do j = 1 by 1 until (last.account);
  set deposits;
  by account;
  amt(j) = deposit;
  typ(j) = type;
end;
run;

PROC SQL is used again to create a macro variable used in the data step ①. However, the maximum number of observations in any account is found in two stages ②. The SELECT statement within the parentheses first finds the maximum number of observations within each account and then the first SELECT statement finds the “maximum of the maximums” and stores that value in the macro variable &MM (once again, the addition of the “SEPARATED BY” clause removes leading spaces from the macro variable). The data step in example 11 produces the table shown below. The value of the macro variable &MM is 5 and that number was used in the KEEP data set option and in the ARRAY statements in example 11.

<table>
<thead>
<tr>
<th>Obs</th>
<th>account</th>
<th>deposit</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>01</td>
<td>50</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>01</td>
<td>200</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>02</td>
<td>50</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>02</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>03</td>
<td>25</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>03</td>
<td>50</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>03</td>
<td>50</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>03</td>
<td>100</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>03</td>
<td>50</td>
<td>1</td>
</tr>
</tbody>
</table>

data set DEPOSITS, example 11

PROC SUMMARY
Rearranging of data is most commonly done with the two methods demonstrated thus far, either PROC TRANSPOSE or a data step that often contains one or more arrays. There is an alternative to these methods, the IDGROUP option in PROC SUMMARY. That option was introduced in SAS V8 but the method is little used since the task of rearranging data using the IDGROUP option is not made explicit in the documentation for the procedure and there are few papers that discuss the method.

The key phrase in the documentation indicating that one might be able to use the IDGROUP option to rearrange a data set from long-to-wide is as follows ...

\[ n \] specifies the number of extreme values for each variable in id-variable-list to include in the OUT= data set. PROC MEANS creates n new variables and uses the suffix .n to create the variable names, where n is a sequential integer from 1 to n. By default, PROC MEANS determines one extreme value for each level of each requested type.
If you consider all the values within a group (we have been using accounts as groups) as extreme and specify a value of N equal to the maximum number of observations within any group in your data, you can use PROC SUMMARY to produce an output data set that is similar to those created with PROC TRANSPOSE and/or a data step using arrays. Some examples will make this clearer (as would the last paper listed on the references).

The data set shown on the right is similar to those used in previous examples, but there are two differences: the data set is not in ACCOUNT order; there are some missing values for both DEPOSIT and TYPE. Once again you would like to put all the information for a given account into one observation.

First, we will only consider one variable, DEPOSIT ...

* EXAMPLE 12;
PROC SQL noprint;
select max(count) into :mm separated by '' from
(select count(*) as count from deposits group by
account);
quit;

PROC SUMMARY data=deposits nway;
class account;
output out=accounts (drop=_type_ _freq_) idgroup(out[&mm](deposit)=dep);
run;

PROC SQL creates a macro variable (&MM) that contains the value of the maximum number of accounts in any particular account group (in this case, 4 values in account 03). The NWAY option is used in PROC SUMMARY since we later use a CLASS statement and we want to limit output to only combinations of values of variables used in the CLASS statement. A CLASS statement instructs PROC SUMMARY to group results by values of the variable ACCOUNT, but the CLASS statement does not require that the data set be sorted by ACCOUNT (as was required with by-group processing done in both PROC TRANSPOSE and data step with arrays).

The OUTPUT statement is where the reshaping instructions occur given that IDGROUP is an OUTPUT statement option. The OUT= option specifies the name of the data set to be created and a DROP data set option removes the SAS-supplied variables _TYPE_ and _FREQ_ from the data set. Notice that no VAR statement is used and no statistics are requested in the OUTPUT statement. That normally causes PROC SUMMARY to produce a data set that contains counts (_FREQ_) of the number of observations in each level of variables in the CLASS statement (similar to running PROC FREQ). Within the IDGROUP option, OUT[n] argument specifies how many extreme values to write to the output data set. Since you want all the values for each account, the value of [n] after OUT is the maximum size of an account group (that value was found with PROC SQL and stored in the macro variable &MM). The second argument is a list of variables in parentheses to be rearranged followed by an “=” and the prefix for the names of the new variables to be created. If no new name is specified, the variable name is used as the prefix (in this case, new variables would be DEPOSIT_1, DEPOSIT_2, etc.). Note that the maximum number of observations within a group that can be transposed with PROC SUMMARY is 100.

<table>
<thead>
<tr>
<th>Obs</th>
<th>account</th>
<th>deposit</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>50</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>25</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>01</td>
<td>50</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>02</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>03</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>01</td>
<td>200</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>03</td>
<td>50</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>03</td>
<td>100</td>
<td>2</td>
</tr>
</tbody>
</table>

data set DEPOSITS, example 12

<table>
<thead>
<tr>
<th>Obs</th>
<th>account</th>
<th>dep_1</th>
<th>dep_2</th>
<th>dep_3</th>
<th>dep_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01</td>
<td>100</td>
<td>50</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>50</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>25</td>
<td></td>
<td>50</td>
<td>100</td>
</tr>
</tbody>
</table>

data set ACCOUNTS, example 12 ... created with PROC SUMMARY and an IDGROUP option

-8-
Not much code revision is needed to rearrange two variables, DEPOSIT and TYPE ...

* EXAMPLE 13;
proc sql noprint;
  select max(count) into :mm separated by '' from
  (select count(*) as count from deposits group by account);
quit;

proc summary data=deposits nway;
  class account;
  output out=accounts (drop=_type_ _freq_) idgroup(out[&mm](deposit type)=dep typ);
run;

<table>
<thead>
<tr>
<th>Obs</th>
<th>account</th>
<th>dep_1</th>
<th>dep_2</th>
<th>dep_3</th>
<th>dep_4</th>
<th>typ_1</th>
<th>typ_2</th>
<th>typ_3</th>
<th>typ_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01</td>
<td>100</td>
<td>50</td>
<td>200</td>
<td>.</td>
<td>1</td>
<td>.</td>
<td>2</td>
<td>.</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>50</td>
<td>100</td>
<td>.</td>
<td>.</td>
<td>2</td>
<td>1</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>25</td>
<td>.</td>
<td>50</td>
<td>100</td>
<td>1</td>
<td>2</td>
<td>.</td>
<td>2</td>
</tr>
</tbody>
</table>

data set ACCOUNTS, example 13 ... created with PROC SUMMARY and an IDGROUP option

The only difference between example 12 and 13 is the addition of a second variable to the list in the IDGROUP option plus a second prefix after the "=" ①. The data set created with PROC SUMMARY is almost identical to that produced in example 11 except for the addition of an underscore in the variable names (also, the values of DEPOSIT and TYPE that are rearranged are a bit different). Another example is shown in appendix B, transposing four variables.

Other IDGROUP options allow you to transpose data in ways that might be more difficult using PROC TRANSPOSE or arrays. For example, consider the data set DEPOSITS shown on page one that has three variables: ACCOUNT, MONTH, DEPOSIT. You want to create a new data set with one observation per account that contains the account number and the amount of just the first and last deposit, with first and last deposits being those associated with the minimum and maximum months that the deposits were made for any given account. That task is quite easy using an IDGROUP ...

* EXAMPLE 14
proc summary data=deposits nway;
  class account; ①
  output out=accounts (drop=_:) ②
    idgroup(min(month) out(deposit)=dep_first) ③
    idgroup(max(month) out(deposit)=dep_last); ④
run;

<table>
<thead>
<tr>
<th>Obs</th>
<th>account</th>
<th>dep_first</th>
<th>dep_last</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01</td>
<td>100</td>
<td>200</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>50</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>50</td>
<td>50</td>
</tr>
</tbody>
</table>

data set ACCOUNTS, example 14 ... created with PROC SUMMARY and two IDGROUP options

Date are analyzed within groups based on the value of the variable ACCOUNT ①. A colon is used as a wildcard, specifying the all variables that start with an underscore ("_") are dropped from the output data set ②. The IDGROUP option now contains a MIN option specifying that the observation with the minimum value of the variable MONTH be used ③. When that observation is found within an account, the value of the variable DEPOSIT is moved to a new variable named DEP_FIRST. A second use of an IDGROUP option does a similar operation when the maximum value of the variable of MONTH within an account is found ④. This example shows that multiple IDGROUP options can be used in one use of PROC SUMMARY.

Another data set named DEPOSITS is shown at the top right of page 4 and that data set has an additional variable named TYPE. Now, your task is the same as in example 14, but you also want to know the type of deposit associated with DEP_FIRST and DEP_LAST ...
* EXAMPLE 15;
proc summary data=deposits nway;
class account;
output out=accounts (drop=_:)
  idgroup(min(month) out(deposit type)=) ①
  idgroup(max(month) out(deposit type)=)/
    autoname; ②
run;

The only differences between examples 14 and 15 are: two variables are specified in the OUT option within the IDGROUP option ①; the AUTONAME option is used rather than naming the variables to be present in the output data set ②. Notice that the variables associated with the minimum month have no numeric suffix while those associated with the maximum month have a numeric suffix of "2".

Look at the data set CLINICAL shown on the first page of appendix A. There are multiple observations for each value of ID, but no variable with a date that would indicate what set of values are first and what set are last. However, if you can assume that the data are in chronological order and your task is the extract the information from the first and last observation within each ID group ...  

* EXAMPLE 16;
proc summary data=clinical nway;
class id;
output out=clin_first_last (drop=_:)
  idgroup(out(chol sbp dbp)=) ①
  idgroup(last out(chol sbp dbp)=) / autoname; ②
run;

Once again, two IDGROUP statements are used. The first just specifies the variables to be transposed and added to data set CLIN_FIRST_LAST ①. Since no options other than OUT are used, PROC SUMMARY uses the first observation within groups for variables in the CLASS statement, in this case the first observation for any given value of variable ID. The second IDGROUP uses the LAST option to request variable values from the last observation for any given ID ②.

Look once again at data set CLINICAL in appendix A and you can see that the table on the right contains variable values form the first and last observation within group of common IDs.
SUMMARY
SAS offers multiple options for converting a data set from long-to-wide. The three methods presented in this paper each offer both advantages and disadvantages depending on the complexity of the data rearranging. A few of the advantages and disadvantages can be summarized as follows:

PROC TRANSPOSE ... Advantages
1/ When only a single variable is transposed, the SAS code needed is usually quite simple.
2/ The use of an ID statement (and multiple ID variables starting in version 9.3) allows control of the placement of variable values into new variables in the transposed data set.
3/ There is no need to know the number of new variables to be created by transposing the data. PROC TRANSPOSE creates as many new variables as needed to accommodate variable values.

PROC TRANSPOSE ... Disadvantages
1/ Transposing more than one variable into a single new observation is difficult (impossible?) with one use of PROC TRANSPOSE.
2/ As with any procedure, customization of the data rearranging is limited to creative use of procedure options.
3/ If a BY statement is used, data must be sorted (or at least grouped) by values of the BY-variable(s).

ARRAYS ... Advantages
1/ Anything that can be done with PROC TRANSPOSE can be done using a data step with one (or more) arrays.
2/ The use of an array subscripts allows control of the placement of variable values into new variables in the transposed data set.
3/ Transposing more than one variable is not much more difficult than transposing only one variable.

ARRAYS ... Disadvantages
1/ Requires data step programming that is sometimes more complicated than using PROC TRANSPOSE and just specifying procedure options.
2/ Some knowledge of the number of new variables to be created is required prior to use of a data step in order to set the size of one or more arrays (the number of new variables being created). This can be done with PROC SQL prior to the data step with the array(s).
3/ If a BY-group processing is used within a data step, data must be sorted (or at least grouped) by values of the BY-variable(s).

PROC SUMMARY ... Advantages
1/ Very little SAS code need to transpose one or more variables.
2/ Data need not be sorted since the CLASS statement in PROC SUMMARY that groups observations does not require a sorted data set.
3/ IDGROUP options such as MIN, MAX, and LAST allow selection of observations to be transposed.

PROC SUMMARY ... Disadvantages
1/ Bit of a "learning curve" to understand PROC SUMMARY option (IDGROUP) most folk have never used.
2/ Some knowledge of the number of new variables to be created is required prior to use of a data step in order to set the number of new variables being created. This can be done with PROC SQL prior to the using PROC SUMMARY.
3/ There is little control as to assigning values to specific variables as can be done with one or more ID variables in PROC TRANSPOSE or with array subscripts in a data step.

REFERENCES
An Introduction to Reshaping (TRANSPOSE) and Combining (MATCH-MERGE) SAS® Data Sets
http://www.nesug.org/proceedings/nesug06/hw/hw09.pdf

The DOW-Loop Unrolled

Transposing Data Using PROC SUMMARY’S IDGROUP Option
ACKNOWLEDGMENTS
I would like to thank John King who taught me about using PROC SUMMARY to rearrange data long-to-wide and was the co-author (and main idea person) for the PROC SUMMARY paper cited in the references.

TRADEMARK CITATION
SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are registered trademarks or trademarks of their respective companies.

CONTACT INFORMATION
The author can be contacted using e-mail... Mike Zdeb msz03@albany.edu
APPENDIX A: REARRANGING DATA PRIOR TO PROC TRANSPOSE

The data set shown below shows the first 10 of 35 observations in a data set with multiple observations for each patient (identified by variable ID). There are measurements for cholesterol (CHOL), systolic blood pressure (SBP), diastolic blood pressure (DBP), and heart rate (HR). In example 7 data are rearranged prior to PROC TRANSPOSE with two ID variables. The following data step shows to use an array to rearrange the data shown below prior to PROC TRANSPOSE. The resulting data set is shown on the right. Notice that there are now 20 observations for ID '01' versus the only 5 in the original data set ... you have made the data set even longer prior to the long-to-wide operation in PROC TRANSPOSE. Also, notice that two ID variables are used in PROC TRANSPOSE and that is only possible starting with version 9.3.

data new_clinical (keep=id new:);
array x(4) chol sbp dbp hr;
do new1 = 1 by 1 until(last.id);
   set clinical;
   by id;
do j = 1 to 4;
   new2 = x(j);
   new3 = vname(x(j));
   output;
end;
end;
run;

proc transpose data=new_clinical out=patients
(drop=_name_);
by id;
var new2;
id new3 new1;
run;

<table>
<thead>
<tr>
<th>Obs</th>
<th>id</th>
<th>chol</th>
<th>sbp</th>
<th>dbp</th>
<th>hr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01</td>
<td>400</td>
<td>160</td>
<td>90</td>
<td>88</td>
</tr>
<tr>
<td>2</td>
<td>01</td>
<td>350</td>
<td>156</td>
<td>88</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>01</td>
<td>350</td>
<td>140</td>
<td>82</td>
<td>76</td>
</tr>
<tr>
<td>4</td>
<td>01</td>
<td>300</td>
<td>138</td>
<td>78</td>
<td>78</td>
</tr>
<tr>
<td>5</td>
<td>01</td>
<td>305</td>
<td>142</td>
<td>82</td>
<td>84</td>
</tr>
<tr>
<td>6</td>
<td>02</td>
<td>390</td>
<td>180</td>
<td>100</td>
<td>82</td>
</tr>
<tr>
<td>7</td>
<td>02</td>
<td>320</td>
<td>178</td>
<td>88</td>
<td>86</td>
</tr>
<tr>
<td>8</td>
<td>02</td>
<td>325</td>
<td>172</td>
<td>82</td>
<td>78</td>
</tr>
<tr>
<td>9</td>
<td>02</td>
<td>304</td>
<td>166</td>
<td>78</td>
<td>99</td>
</tr>
<tr>
<td>10</td>
<td>02</td>
<td>299</td>
<td>150</td>
<td>80</td>
<td>80</td>
</tr>
</tbody>
</table>

data set CLINICAL

<table>
<thead>
<tr>
<th>Obs</th>
<th>new1</th>
<th>id</th>
<th>new2</th>
<th>new3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>01</td>
<td>400</td>
<td>chol</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>01</td>
<td>160</td>
<td>sbp</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>01</td>
<td>90</td>
<td>dbp</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>01</td>
<td>88</td>
<td>hr</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>01</td>
<td>350</td>
<td>chol</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>01</td>
<td>156</td>
<td>sbp</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>01</td>
<td>88</td>
<td>dbp</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>01</td>
<td>80</td>
<td>hr</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>01</td>
<td>350</td>
<td>chol</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>01</td>
<td>140</td>
<td>sbp</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>01</td>
<td>82</td>
<td>dbp</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>01</td>
<td>76</td>
<td>hr</td>
</tr>
<tr>
<td>13</td>
<td>4</td>
<td>01</td>
<td>300</td>
<td>chol</td>
</tr>
<tr>
<td>14</td>
<td>4</td>
<td>01</td>
<td>138</td>
<td>sbp</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>01</td>
<td>78</td>
<td>dbp</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>01</td>
<td>78</td>
<td>hr</td>
</tr>
<tr>
<td>17</td>
<td>5</td>
<td>01</td>
<td>305</td>
<td>chol</td>
</tr>
<tr>
<td>18</td>
<td>5</td>
<td>01</td>
<td>142</td>
<td>sbp</td>
</tr>
<tr>
<td>19</td>
<td>5</td>
<td>01</td>
<td>82</td>
<td>dbp</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>01</td>
<td>84</td>
<td>hr</td>
</tr>
<tr>
<td>21</td>
<td>1</td>
<td>02</td>
<td>390</td>
<td>chol</td>
</tr>
<tr>
<td>22</td>
<td>1</td>
<td>02</td>
<td>180</td>
<td>sbp</td>
</tr>
<tr>
<td>23</td>
<td>1</td>
<td>02</td>
<td>100</td>
<td>dbp</td>
</tr>
<tr>
<td>24</td>
<td>1</td>
<td>02</td>
<td>82</td>
<td>hr</td>
</tr>
<tr>
<td>25</td>
<td>2</td>
<td>02</td>
<td>320</td>
<td>chol</td>
</tr>
</tbody>
</table>

data set NEW_CLINICAL
PROC TRANSPOSE produced the data set shown below.

| Obs | id | chol1 | chol2 | chol3 | chol4 | chol5 | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | dbp1 | dbp2 | dbp3 | dbp4 | dbp5 | hr1 | hr2 | hr3 | hr4 | hr5 |
|-----|----|-------|-------|-------|-------|-------|------|------|------|------|------|------|------|------|------|-----|-----|-----|-----|-----|
| 01  | 01 | 400   | 350   | 350   | 300   | 305   | 160  | 156  | 140  | 138  | 142  | 90   | 88   | 82   | 78   | 82  | 88  | 80  | 76  | 78  | 84 |
| 02  | 02 | 390   | 320   | 325   | 304   | 299   | 180  | 178  | 172  | 166  | 150  | 100  | 88   | 82   | 78   | 80  | 82  | 86  | 76  | 78  | 80 |
| 03  | 03 | 387   | 377   | 380   | 400   | 390   | 190  | 188  | 182  | 186  | 182  | 110  | 98   | 88   | 92   | 90  | 90  | 84  | 80  | 82  | 78 |
| 04  | 04 | 380   | 370   | 355   | 306   | 279   | 120  | 122  | 128  | 130  | 126  | 78   | 76   | 68   | 72   | 74  | 56  | 58  | 60  | 68  | 62 |
| 05  | 05 | 399   | 379   | 375   | 365   | 321   | 128  | 128  | 132  | 130  | 132  | 62   | 66   | 70   | 76   | 78  | 60  | 62  | 58  | 66  | 68 |
| 06  | 06 | 387   | 379   | 375   | 365   | 321   | 128  | 128  | 132  | 130  | 132  | 62   | 66   | 70   | 76   | 78  | 60  | 62  | 58  | 66  | 68 |
| 07  | 07 | 376   | 379   | 389   | 388   | 400   | 118  | 124  | 120  | 124  | 128  | 68   | 72   | 68   | 72   | 80  | 54  | 70  | 62  | 60  | 66 |

APPENDIX B: TRANSPOSING FOUR VARIABLES WITH PROC SUMMARY
The task performed in appendix A can also be accomplished using PROC SUMMARY and an IDGROUP. No data rearranging is required and the original data set (CLINICAL) can be used ...

* use PROC SQL to find the maximum number of observations for any person (ID);
proc sql noprint;
select max(count) into :mm separated by ' ' from
(select count(*) as count from clinical group by id);
quit;

proc summary data=clinical nway;
class id;
output out=patients (drop=_type_ _freq_) idgroup(out[&mm](chol sbp dbp hr)=);
run;

PROC SUMMARY produced the data set shown below.

| Obs | id | chol1 | chol2 | chol3 | chol4 | chol5 | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | dbp1 | dbp2 | dbp3 | dbp4 | dbp5 | hr1 | hr2 | hr3 | hr4 | hr5 |
|-----|----|-------|-------|-------|-------|-------|------|------|------|------|------|------|------|------|------|-----|-----|-----|-----|-----|
| 01  | 01 | 400   | 350   | 350   | 300   | 305   | 160  | 156  | 140  | 138  | 142  | 90   | 88   | 82   | 78   | 82  | 88  | 80  | 76  | 78  | 84 |
| 02  | 02 | 390   | 320   | 325   | 304   | 299   | 180  | 178  | 172  | 166  | 150  | 100  | 88   | 82   | 78   | 80  | 82  | 86  | 76  | 78  | 80 |
| 03  | 03 | 387   | 377   | 380   | 400   | 390   | 190  | 188  | 182  | 186  | 182  | 110  | 98   | 88   | 92   | 90  | 90  | 84  | 80  | 82  | 78 |
| 04  | 04 | 380   | 370   | 355   | 306   | 279   | 120  | 122  | 128  | 130  | 126  | 78   | 76   | 68   | 72   | 74  | 56  | 58  | 60  | 68  | 62 |
| 05  | 05 | 399   | 379   | 375   | 365   | 321   | 128  | 128  | 132  | 130  | 132  | 62   | 66   | 70   | 76   | 78  | 60  | 62  | 58  | 66  | 68 |
| 06  | 06 | 387   | 379   | 375   | 365   | 321   | 128  | 128  | 132  | 130  | 132  | 62   | 66   | 70   | 76   | 78  | 60  | 62  | 58  | 66  | 68 |
| 07  | 07 | 376   | 379   | 389   | 388   | 400   | 118  | 124  | 120  | 124  | 128  | 68   | 72   | 68   | 72   | 80  | 54  | 70  | 62  | 60  | 66 |