(10) REARRANGING DATA

If you want to conduct an analysis across observations in a data set, you can use SAS procedures. If you want to conduct an analysis within observations, you can use SAS functions. However, there are occasions when neither a procedure nor a function will suffice. 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 sense means converting variables into observations or observations into variables. Rather than discuss methods of rearranging and showing the situations where they are applicable, two different problems are presented and methods for solving the problems are shown.

...ONE-TO-MANY

The first problem is one in which you have observations that contain multiple occurrences of a medical diagnosis. Each observation represents one person and your task is to create a table that shows how often each diagnosis occurs within the data set. The following creates the data set to be used...

data many_dx;
  infile datalines missover;
  length id $2 dx1-dx5 $3;
  input id dx1-dx5;
  datalines;
  01 647 038
  02 428 518 416
  03 642 674 431 648
  04 415 280 997 427 453
  05 641 668 648 666 641
  06 670 041 728 427 518
  07 864 866
  08 648 801
  09 666 666 286 286
  10 659 560 348 507
  
  run;

One approach is to use PROC FREQ on each of the five diagnoses and add the results of the five tables...

proc freq data=many_dx;
  tables dx1-dx5;
  run;

Another approach is to rearrange the data so a new data set to be analyzed contains only one variable, diag, with one observation for each diagnosis in the original data set, i.e. create many observations from one observation (one-2-many).

...Example 10.1...

data all_dx (keep=diag);
  set many_dx;
  diag = dx1; if diag ne '' then output;
  diag = dx2; if diag ne '' then output;
  diag = dx3; if diag ne '' then output;
  diag = dx4; if diag ne '' then output;
  diag = dx5; if diag ne '' then output;
  run;

proc freq data=all_dx;
  table diag;
  run;
A new data set is created that contains only one variable, DIAG.

The value for each diagnosis (dx1 through dx5) in the original data set is placed in the variable diag. If the value is not missing, an observation is output to the new data set.

There is yet another method for rearranging data, PROC TRANSPOSE. This procedure can be used to turn variables into observations, and to turn observations into variables. In each example shown thus far, variables (diagnoses) within one observation have been turned into one variable (diag) occurring across many observations.

---Example 10.4---

data many_dx;
  infile datalines missover;
  length id $2 dx1-dx5 $3;
  input id dx1-dx5;
  datalines;
  01 647 038
  02 428 518 416 710
  03 642
  04 415 280
  05 641 668 648 666 641
;   run;

proc transpose data=many_dx out=diags;
  var dx1-dx5;
  run;

proc print data=diags;
  run;
PROC TRANSPOSE is used to rearrange the data in data set many_dx, resulting in data set diags. A VAR statement specifies the variables to be transposed (rearranged). The data set created by PROC TRANSPOSE has data in observations that were in variables in the original data set.

Though the data have been rearranged, the result is not quite what was desired, a data set with one variable (diag). Using a by statement in PROC TRANSPOSE will make the resulting data set look more like those produced with data steps and arrays in previous examples.

...Example 10.5...
proc transpose data=many_dx out=diags;
var dx1-dx5;
by id;
run;
proc print data=diags;
run;

A BY statement is added to PROC TRANSPOSE specifying that the variables specified in the VAR statement should be transposed within each value of the ID variable. The resulting data set is much closer to the desired data set.

Notice that in both examples 10.4 and 10.5, PROC TRANSPOSE created its own names for the variables in the output data set, COL followed by a numeric suffix. In example 10.4, the variable named COL1 is the one that should be named DIAG. The procedure adds a new variable to the data set, _NAME_, and its value across the observations is the names of the variables occurring in the VAR statement. In example 10.5, the BY variable is also added to the data set, and there are the same number of observations within each different value of by variable, even if some have missing data in the
variable COL1. More changes to PROC TRANSPOSE can produce a data set exactly like that created in examples 10.1 and 10.2.

...Example 10.6...
proc transpose
  data=many_dx
  out=diags (keep=col1 rename=(col1=diag) where=(diag ne ' ')) ;
  var dx1-dx5;
  by id;
  run;
proc print data=diags;
run;

Several data set options are used on the output data set. First, a KEEP option specifies that only the variable COL1 is to be added to the data set DIAGS. Next, a RENAME option changes the variable name COL1 to DIAG. Finally, a WHERE option tells PROC TRANSPOSE to only output observations in which the value of the variable DIAG is not missing.

The data set is exactly what is desired, containing only one variable named DIAG with no observations having missing data.

...MANY-TO-ONE
The second problem is one in which the there is a data set containing the information on the number of births to mothers who took drugs during their pregnancy. There are three variables in the data set: the year the birth occurred (YEAR); whether the mother took illegal drugs, therapeutic or prescribed drugs, or no drugs (DRUGS); the number of births in the observation (BIRTHS). The task is to put all the data for a given year into one observation and then compute the percentage of births to mothers who took either illegal and/or therapeutic+prescribed drugs. In the data set, the variable DRUGS has the following values: 1, no drugs taken; 2, therapeutic+prescribed drugs taken; 3, illegal drugs taken. As in the first ONE-TO-MANY example, a data step can be used to rearrange the data.

...Example 10.7...
data births1;
  length year $4 drugs $1;
  input year drugs births @@;
datalines;
;run;
data births2;
  retain d1-d3;
  set births1;
  data births3;
  length year $4 drugs $1;
  input year drugs births @@;
datalines;
;run;
by year;
if drugs eq '1' then d1 = births;
else
if drugs eq '2' then d2 = births;
else
if drugs eq '3' then d3 = births;
if last.year then output;
run;

data births2 (keep=year total pct1 pct2);
set births2;
total = sum (of d1-d3);
pct1 = 100 * d2 / total;
pct2 = 100 * d3 / total;
label
year  = 'YEAR OF BIRTH'
total = 'NUMBER OF BIRTHS'
pct1  = '% THERAPEUTIC OR PRESCRIBED'
pct2  = '% ILLEGAL'
;
run;

proc print data=births2 noobs label;
var year total pct1 pct2;
format total comma8. pct1 pct2 6.1;
run;

<table>
<thead>
<tr>
<th>YEAR OF BIRTH</th>
<th>NUMBER OF BIRTHS</th>
<th>THERAPEUTIC OR PRESCRIBED</th>
<th>% ILLEGAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1993</td>
<td>282,377</td>
<td>3.0</td>
<td>1.7</td>
</tr>
<tr>
<td>1994</td>
<td>277,945</td>
<td>4.4</td>
<td>1.5</td>
</tr>
<tr>
<td>1995</td>
<td>271,042</td>
<td>6.2</td>
<td>1.4</td>
</tr>
<tr>
<td>1996</td>
<td>263,611</td>
<td>7.2</td>
<td>1.5</td>
</tr>
<tr>
<td>1997</td>
<td>256,976</td>
<td>8.4</td>
<td>1.5</td>
</tr>
<tr>
<td>1998</td>
<td>257,748</td>
<td>9.0</td>
<td>1.4</td>
</tr>
<tr>
<td>1999</td>
<td>255,147</td>
<td>9.4</td>
<td>1.3</td>
</tr>
<tr>
<td>2000</td>
<td>254,230</td>
<td>9.9</td>
<td>1.3</td>
</tr>
</tbody>
</table>

1. A data set is created with one observation for every year-drug combination.
2. The retain statement is used to prevent values of the variables D0, D1, and D2 from being set to missing during execution of the data step.
3. The original data are read with a SET statement plus a BY statement.
4. The value of the variable DRUGS is checked and the value of the variable births is assigned to the appropriate variable.
5. An observation is written to a new data set when the last observation is by group is read.
6. Another data step is used to compute percentages.
7. The output from PROC PRINT shows the percentage of births to mothers who took the two different classes of drugs within each year.

One feature of the original data set should be noted. There are three observations within each year, one observation for no, therapeutic+prescribed, and illegal drugs. If any of the years did not have all three such observations, the data step that creates data set BIRTHS2 would have to be modified as follow.

data births2;
retain d1-d3;
set births1;
by year;
if first.year then do;
d1 = 0; d2 = 0; d3 = 0;
end;
if drugs eq '1' then d1 = births;
else
if drugs eq '2' then d2 = births;
else
if drugs eq '3' then d3 = births;
if last.year then output;
run;
The new statements that begin with IF FIRST.YEAR... ensure that the values of D0, D1, and D2 are set to zero when each new year is encountered in the data set BIRTHS1 and that data assigned to any given year actually belong to that year. In other words, no data ends up in a new year that have been retained from the previous year. The data step that creates data set BIRTHS2 could be modified to use an array.

Example 10.9...

PROC TRANSPOSE is used to rearrange the data in data set BIRTHS1 to create data set BIRTHS2. The VAR statement specifies the variable whose values will appear in the new data set. The ID statement specifies the variable that controls the variable that will contain the value of the variable in the VAR statement. The BY variable specifies the variable that controls the observation into which variables are placed. The output from PROC PRINT shows that the data set BIRTHS2 needs new variable names, and that a new variable _NAME_ was added to the data set.

Example 10.10...

PROC TRANSPOSE uses an underscore plus the values of the ID variable as a suffix to create the variable names in the output data set. A PROC TRANSPOSE option and a data set option can be to alter the content of data set BIRTHS2.
1 A data set option is used to drop the variable _NAME_ from the data set BIRTHS2. The PREFIX= option replaces the default prefix on an underscore with a D.
2 The data set is exactly what is desired, correct variable names and no extra variables.

(11) COMBINING SAS DATA SETS

There are a number of ways to combine SAS data sets:

- **concatenate** - stack data sets (place one after another)
- **interleave** - stack data sets, but form the result in order by one or more variables present in the data sets

  NOTE: both concatenation and interleaving result in a data set that has as many observations as the sum of all the observations in the data sets being combined

- **one-to-one merge** - combine observation one in data set one with observation one in data set two, observation two-with-two, three-with-three, etc.

  NOTE: the data set resulting from one-to-one merge will have as many observations as the number of observations in the largest data set involved in the merge

- **matched-merge** - combine observations in data sets based upon the value of one or more variables

  NOTE: the number of observations in the resulting data set depends on the content of the data sets being combined and on the matching options being used

- **update** - change the values of the variables in certain observations in a data set based upon the values of variables in another data set

  NOTE: as with matched-merge, the number of observations in the resulting data set depends upon the content of the data sets being combined and on the matching options being used

---

**CONCATENATION**

There are a number of methods that can be used to concatenate data sets. The easiest, but probably the least efficient way, is to use the SET statement.

**Example 11.1...**

data jan;
  retain month 'JAN';
  input ssn weight;
  datalines;
  123456789 150
  987654321 120
  001001234 180
;  
run;

* data for subjects in study in February;

data feb;
  retain month 'FEB';
  input ssn weight;
  datalines;
  123456789 148
  001001234 115
  987654321 122
  888888888 190
;  
run;
* make a new data set out of the two old data sets;  
data jan_feb;  
set jan feb;  
run;  

proc print data=jan_feb noobs;  
run;  

month     ssn       weight  
JAN     123456789      150  
JAN     987654321      120  
JAN     001001234      180  
FEB     123456789      148  
FEB     001001234      115  
FEB     987654321      122  
FEB     888888888      190  

1 Data set JAN is created with three variables: month, ssn, weight.  
2 Data set FEB is created with the same three variables.  
3 The two data sets are combined with a SET statement.  
4 The output from PROC PRINT shows the two data sets combined 'tot-to-bottom'.

You can concatenate as many data sets as you want with a SET statement. This method forces SAS to read every observation in each of the data sets being combined. Concatenation using the SET statement is very robust, i.e. there are not many conditions that will cause it to fail. However, concatenation using SET is sensitive to the TYPE of variables in the data sets. You can't have a variable with the same name across data sets defined as NUMERIC in one data set and as CHARACTER in another. If SSN had been NUMERIC in data set JAN and CHARACTER in data set FEB, the data sets could not be combined and the following error would appear in the LOG when the data sets were used in the same SET statement...

ERROR: Variable SSN has been defined as both character and numeric.

Each data set need not have the same number of variables. Variables not present in one of the data sets will have their values set to missing in the combined data set. If data set FEB contained a new variable not present in data set JAN, e.g. ZIP, all values of ZIP in observations that are contributed to data set JAN_FEB by data set JAN would be set to missing. If variables have different LENGTHS in the data sets being combined, the LENGTHS of variables in the resulting data set will be those in the first data set in the SET statement in which the variables are present.

...Example 11.2...

data jan;  
input (name weight) ($8. 3.);  
datalines;  
ZDEB  180  
SMITH  200  
;  
run;  

data feb;  
input (name weight zip) ($10. 3. $5.);  
datalines;  
SMITH  21012202  
ZDEB   18012203  
WASHINGTON19000001  
;  
run;
data jan_feb;
set jan (in=j) 3
    feb
    ;
    if j then month='JAN';
else month='FEB';
run;

proc contents data=jan_feb;
run;

proc print data=jan_feb noobs;
run;

#    Variable    Type    Len    Pos
-----------------------------------
 4    month       Char      3     21
 1    name        Char      8      8
 2    weight      Num       8      0
 3    zip         Char      5     16

month    name        weight     zip
JAN     ZDEB          180
JAN     SMITH         200
FEB     SMITH         210     12202
FEB     ZDEB          180     12203
FEB     WASHINGT      190     00001

1  Data set JAN is created with two variables:  name, weight.  Name has a length of 8.
2  Data set FEB is created with an additional variable, zip, and a name has a length of 10.
3  The data sets are combined using SET and an IN= data set option.
4  The IN= variable is used to assign a month to each observation.
5  PROC CONTENTS run using the new data set shows that the variable name has a length of 8.
6  Values for the variable zip are missing in observations contributed by data set JAN.  One name in an
observation from data set FEB has been truncated.

The variable NAME had a length of 8 in data set JAN.  Since data set JAN occurs first in the SET
statement, the length of variable NAME in the combined data set JAN_FEB is also 8 and the names
from data set FEB are truncated.  A LENGTH statement in the data step that combines JAN and FEB
would override this problem, or just change the order of the data sets in the SET statement...

length name $10;
-or-
set feb jan;

The values for variable ZIP in observations contributed to JAN_FEB by data set JAN are missing and the
length of ZIP is 5, taken from information contributed by data set FEB.

Rather than creating data sets with a month variable as in example 11.1, neither data set JAN nor FEB
had a month variable.  When the data sets were combined, a IN= data set option was used to determine
if an observation had been provided to the new data set by data set JAN or FEB.  The IN= option creates
a new variable (in this case J) that only exists for the duration of the data ste and that can have only two
values, 0 or 1.  In this example, if an observation is contributed by data set JAN, then the variable J will
have a value of 1, otherwise it will be 0.  The statement beginning with 'IF J' will only be true when J=1,
or when an observation from data set JAN is read.

<PROC APPEND section deleted>
...INTERLEAVING

Interleaving is 'sorted concatenation' and is done using a SET statement and a BY variable(s).

...Example 11.6...

```
data jan;
  retain month 'JAN';
  input (ssn weight) ($9. 3.);
  datalines;
  123456789150
  987654321120
  001001234180
;  
run;

data feb;
  retain month 'FEB';
  input (ssn weight) ($9. 3.);
  datalines;
  123456789148
  001001234115
  987654321122
  888888888190
;  
run;

proc sort data=jan;
  by ssn;
run;
proc sort data=feb;
  by ssn;
run;

data jan_feb;
  set jan feb;
  by ssn;
run;

proc print data=jan_feb noobs;
run;
```

1 The data sets from example 11.1 are used again.
2 Both data sets are sorted by variable SSN.
3 The data sets are combined with a SET statement, plus a BY statement.
4 The results from PROC PRINT show the data set in order by SSN.

There is one consideration about sorting the new, larger combined data set in lieu of sorting the two smaller data sets then interleaving. The time to sort a data set is not always linearly related to the size of the data set, i.e. it might not take just twice as long to sort a data set that has twice as many observations than another. If the combined data set is very large, it may be more EFFICIENT to first sort the smaller data sets then use SET and BY to interleave.
...ONE-TO-ONE MERGE
Concatenation and interleaving combine data sets 'top-to-bottom' - observations in one data set are added to observations in another data set. Merging combines data sets 'side-to-side' - variables in one data set are added to variables in another data set. A one-to-one merge combines observation #1 in data set one with observation #1 in data set two, #2 with #2, etc. SAS 'sees' the data set mentioned on the right LAST. If you have variables with the same names in data sets that are being merged, SAS will keep the values of the variables from the right-most named data set in the MERGE statement. A one-to-one merge is only useful when you are sure that the order of the observations in the data sets being combined is appropriate, i.e. observation #1 in data set one should be combined with #1 in data set two, etc.

...Example 11.7...

data jan;
input ssn weight zip : $5.;
datalines;
001001234 180 12203
123456789 150 13502
987654321 120 12345
;
r;
da data feb;
input ssn weight;
datalines;
001001234 115
123456789 148
987654321 122
;
r;
data jan_feb;
merge jan feb;
r;
proc print data=jan_feb noobs;
format ssn ssn.;
r;

ssn        weight     zip
001-00-1234      115     12203
123-45-6789      148     13502
987-65-4321      122     12345

1 Data set JAN is created with three variables: ssn, weight, zip.
2 Data set FEB is created with only two variables: ssn, weight.
3 The data sets are combined using MERGE
4 The resulting data set has three observations and three variables.

The WEIGHT in data set JAN_FEB is from data set FEB, the right-most data set in the merge statement (the one SAS 'sees' last). The SSN is also from data set FEB, but since it is the same as in data set JAN, that is not as obvious. The variable ZIP is contributed by data set JAN (there is no variable ZIP in data set FEB to overwrite the values of ZIP).

If there are variables with the same name in the data sets, e.g. WEIGHT in JAN and FEB, you can use the RENAME data set option to avoid the replacement of values of variables.
...Example 11.8...

```sas
data jan_feb;
merge jan (rename=(weight=wt1)) feb (rename=(weight=wt2));
diff_wt = wt2-wt1;
run;
```

```sas
proc print data=jan_feb noobs;
var ssn zip wt1 wt2 diff_wt;
format ssn ssn. ;
run;
```

1  The two data sets from example 11.7 are combined with MERGE, but the variable with a name common to both data sets is renamed.
2  An additional statement is added to find the change in weight from one month to the next.
3  The results from PROC PRINT show that you can now see all the data.

Since the merge is performed within a data step, you can use additional statements. You are not limited to just performing the merge. In the example, the renamed variables are used to compute another variable, the difference in weight between January and February.

What would happen if we added one more observation to just data set JAN?

...Example 11.9...

```sas
data jan;
input ssn weight zip : $5.;
datalines;
001001234 180 12203
123456789 150 13502
888888888 222 14001
987654321 120 12345
;
run;
```

```sas
data feb;
input ssn weight;
datalines;
001001234 115
123456789 148
987654321 122
;
run;
```

```sas
data jan_feb;
merge jan (rename=(weight=wt1)) feb (rename=(weight=wt2));
avg_wt = mean(of wt1-wt2);
run;
```

```sas
proc print data=jan_feb;
var ssn zip wt1 wt2 avg_wt;
run;
```

1  The two data sets from example 11.7 are combined with MERGE, but the variable with a name common to both data sets is renamed.
2  An additional statement is added to find the change in weight from one month to the next.
3  The results from PROC PRINT show that you can now see all the data.
4  What would happen if we added one more observation to just data set JAN?
1 A new observation is added to data set JAN. There is no matching observation (same SSN) in FEB.
2 The two data sets are combined with MERGE.
3 One-to-one merging was done. The third observation in data set JAN was paired with the third observation in data set FEB.
4 There were only three observations in data set FEB. All the values for variables in the fourth observation of JAN_FEB are taken from data set JAN.

The first two observations in JAN_FEB are identical to those found in example 11.8. However, observations three and four in JAN_FEB are not correct. Observation three from JAN is combined with observation three from FEB. Since the variable SSN is in both data sets, the SSN from FEB (the right-most data set in the merge statement) ends up in data set JAN_FEB. Though observations having two different SSNs are combined, SAS still computes AVG_WT. Observation four in data set JAN_FEB is observation four from data set JAN. It was not combined with any data from FEB since FEB was all out of observations for one-to-one matching. The variable WT2 has a value of missing, i.e. there is no weight for February. Since a function was used to compute the average weight, the missing value for WT1 did not result in missing value for AVG_WT.

The moral to this example is that a one-to-one merge should not be used unless you know all the details about the data sets being combined. If you can be sure that both data sets are in the proper order and that there are not any extra observations in any of the data sets being combined, one-to-one merge is safe.

**MATCHED MERGE**

Matched merging is 'side-to-side' combination of data sets in which observations are combined on the basis of the value(s) of one or more variables, i.e. BY variable(s). Assume that the data sets JAN and FEB from example 11.9 are used.

**Example 11.10...**

```sas
data jan_feb;
merge jan (rename=(weight=wt1)) feb (rename=(weight=wt2));
by ssn;
avg_wt = mean(of wt1-wt2);
run;
```

```sas
proc print data=jan_feb;
var ssn zip wt1 wt2 avg_wt;
run;
```

<table>
<thead>
<tr>
<th>ssn</th>
<th>zip</th>
<th>wt1</th>
<th>wt2</th>
<th>avg_wt</th>
</tr>
</thead>
<tbody>
<tr>
<td>001-00-1234</td>
<td>12203</td>
<td>180</td>
<td>115</td>
<td>147.5</td>
</tr>
<tr>
<td>123-45-6789</td>
<td>13502</td>
<td>150</td>
<td>148</td>
<td>149.0</td>
</tr>
<tr>
<td>888-88-8888</td>
<td>14001</td>
<td>222</td>
<td>.</td>
<td>222.0</td>
</tr>
<tr>
<td>987-65-4321</td>
<td>12345</td>
<td>120</td>
<td>122</td>
<td>121.0</td>
</tr>
</tbody>
</table>

1 A BY statement is added to the data step merge. All data set involved in the merge must be sorted according to values of the variable(s) in the by statement.
2 Observation three from data set JAN is not matched to an observation in data set FEB.
3 The correct match is made for the observation having SSN 987654321.

Since a by-variable is used, all data sets in the MERGE statement must be sorted in order of that by-variable. In this example, both JAN and FEB are in SSN-order so no sorts are done. Data set JAN_FEB is now 'correct' in that it contains matched records, plus the extra unmatched observation from data set JAN.

Match-merging can be applied to a problem posed earlier in these notes. Example 8.9 contains data on patients, with multiple observations for most patients. One question to answer with these data is how the value of cholesterol changed from first visit to last visit.
**Example 11.11...**

```sas
data manyids;
  input id : $2. visit : mmddyy8. chol;
  format visit mmddyy8.;
  datalines;
  01 01/05/89   400 1
  01 05/18/90   350
  01 11/11/90   305
  01 02/18/91   260
  02 12/25/91   200
  03 01/01/90   387
  03 02/02/91   380
  04 05/15/91   380
  04 08/20/91   370
  04 03/23/92   355
  04 07/05/92   261
; run;
```

```sas
proc sort data=manyids;
  by id visit;
run;
```

```sas
data first_visit last_visit;
  set manyids;
  by id;
  if not (first.id and last.id);
  if first.id then output first_visit;
  else
    if last.id the output last_visit;
run;
```

```sas
data first_last;
  merge first_visit (rename=(visit=visit1 chol=chol1))
    last_visit (rename=(visit=visit2 chol=chol2));
  by id;
  diff_days = visit2 - visit1;
  diff_chol = chol2 - chol1;
run;
```

```sas
proc print data=first_last noobs;
  var id visit1 visit2 diff_days chol1 chol2 diff_chol;
run;
```

1. This data step is the same as example 8.9, creating a data set with multiple observations for all patients except for ID=02.
2. The data set is sorted in ID order and in date order within each patient.
3. Two data sets are created, one with data from the first visit for each patient, another with data from the last visit.
4. The original data are read with a SET plus BY, allowing the use of a FIRST. and LAST. variable, ID.
5. The observations are screened to see if there are any patients who only had one visit. or an observation that is both the FIRST and LAST observation in a group of IDs. All single visit patients are excluded.
6. FIRST and LAST variables are used to write observations to either of two data sets.
7. The new data sets are merge by ID. Variables with names common to both data sets are renamed. The number of days in the study and the change in cholesterol are computed.
8. The final, matched data set is the same as that produced in example 8.9, with ID=02 excluded.
The results of a matched merge are predictable when there is only one observation with a specific value of a BY variable in one or all of the data sets that are being combined. In example 11.10, there were no repeated values of the variable SSN in either data set JAN or FEB. Another name for this combination of data sets is one-to-one match merge. If there are multiple observations with the same value of the by-variable in one of the data sets being merged, the term one-to-many match merge (or many-to-one match merge) is used.

...Example 11.12...

1 The first data set contains demographic data.
2 The second data set contains medical data.
3 Demographic and medical data are combined using merge with a by variable.
4 Each SMITH (MANY) in data set demographic was combined with SMITH (ONE) in data set medical.

In a matched merge, the BY statement causes SAS to look for possible matches based on the value of the by-variable(s). Each SMITH in DEMOGRAPHIC 'found' SMITH in MEDICAL. There is one extra bit of information that will allow us to determine which of the matched SMITH observations is correct. What if you changed the matching criteria to: names must match; age must be within one year.
...Example 11.13...

data both;
merge
demographic (rename=(age=age1))
medical (rename=(age=age2)) ;
by name;
if  abs(age1 - age2) le 1;
run;

proc print data=both noobs;
run;

name     age1    age2     zip      rate    cholesterol
ADAMS     20      20     12203      89         2003
BROWN     21      21     10001      60         140
SMITH     33      34     12012      71         150

1 The data sets are merged and the age is renamed in both data sets.
2 The ages in the two data sets are compared. ABS(...) is the absolute value function. This statement
is the same as writing... if -1 <= (age1 - age2) <= 1;
3 Data set BOTH now only has the observation for SMITH in which the values for the variable ae were
within one year in the two data sets.

Use of match-merge is more complicated when there are observations with repeated values of the by-
variable(s) used to match records in both data sets. This type of merge is referred to as a many-to-many
match merge. Whether you get the correct result whenever you perform this type of merge depends on
the both the data sets being used, the order of the data sets, and the matching criteria. You may or may
not get the correct result.

...Example 11.14...
data dataset1;
input name $ age;
datalines;
ADAMS  20
BROWN  21
BROWN  30
JONES  45
JONES  46
JONES  47
LAWRENCE 10
LAWRENCE 14
LAWRENCE 16
SMITH  50
WALTERS 29
;
run;

data dataset2;
input name $ age hr;
label hr = 'heart rate';
datalines;
ADAMS  21  89
BROWN  15  60
BROWN  21  75
BROWN  40  80
JONES  48  60
KELLY  57  90
LAWRENCE 16  60
LAWRENCE 20  84
SMITH  30  71
SMITH  50  55
;
run;
data both;
merge
dataset1 (rename=(age=age1))
dataset2 (rename=(age=age2))
;
by name;
if abs(age1 - age2) le 1;
run;

proc print data=both noobs label;
var name age1 age2 hr;
run;

from the LOG file for the merge data step....

NOTE: MERGE statement has more than one data set with repeats of BY values.

<table>
<thead>
<tr>
<th>name</th>
<th>age1</th>
<th>age2</th>
<th>rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADAMS</td>
<td>20</td>
<td>21</td>
<td>89</td>
</tr>
<tr>
<td>JONES</td>
<td>47</td>
<td>48</td>
<td>60</td>
</tr>
<tr>
<td>KELLY</td>
<td>.</td>
<td>57</td>
<td>90</td>
</tr>
<tr>
<td>SMITH</td>
<td>50</td>
<td>50</td>
<td>55</td>
</tr>
<tr>
<td>WALTERS</td>
<td>29</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

1 The first data set contains only the variables name and age and there are repeated names.
2 The second data set adds the variable heart rate, and once again names are repeated.
3 The data sets are merged by name.
4 Only those observations in which the age is within one year are output.
5 A NOTE in the LOG file gives you a warning about the structure of the data sets.
6 Data set both has missed some observations that met the matching criteria.

Three observations in data set BOTH meet the matching criteria, same NAME in both data sets and ages within one year of each other. Two other observations are present since no observations with corresponding names are found for either. The DEFAULT behavior of a matched merge is to put matched records in the resulting data set, PLUS all unmatched records from all the data sets being merged.

How did the match-merge form data set BOTH? When there are repeated values of BY-variables in both data sets, SAS will combine the first observation in a BY-group in one data set with the first observation in a by-group in the other data set, the second with the second, etc. This is what occurred during the merge of the observations in DATASET1 and DATASET2 in example 11.14.
Here is a look at how merge created data set BOTH...

<table>
<thead>
<tr>
<th>OBS</th>
<th>DATASET1</th>
<th>DATASET2</th>
<th>AGE WITHIN ONE YEAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ADAMS/20</td>
<td>ADAMS/21</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>BROWN/21</td>
<td>BROWN/15</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>BROWN/30</td>
<td>BROWN/21</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BROWN/40</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>JONES/45</td>
<td>JONES/48</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>JONES/46</td>
<td></td>
<td>NO</td>
</tr>
<tr>
<td>2</td>
<td>JONES/47</td>
<td></td>
<td>YES</td>
</tr>
<tr>
<td>3</td>
<td>NO MATCH</td>
<td>KELLY/57</td>
<td>UNKNOWN</td>
</tr>
<tr>
<td></td>
<td>LAWRENCE/10</td>
<td>LAWRENCE/16</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>LAWRENCE/14</td>
<td>LAWRENCE/20</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>LAWRENCE/16</td>
<td></td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>SMITH/50</td>
<td>SMITH/30</td>
<td>NO</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>SMITH/50</td>
<td>YES</td>
</tr>
<tr>
<td>5</td>
<td>WALTERS/29</td>
<td>NO MATCH</td>
<td>UNKNOWN</td>
</tr>
</tbody>
</table>

This illustration shows that one match-merge may be a combination of all three matching types: one-to-one; one-to-many / many-to-one; many-to-many. Within the larger merge, those observations involved in either a one-to-one or a one-to-many / many-to-one match presented no problems. The matching criteria produced the correct result in data set BOTH for ADAMS, JONES, and SMITH. However, there were two many-to-many matches within the larger merge, one for BROWN and another for LAWRENCE.

Were any possible matches missed for these names? The match-merge missed combining the observations in both data sets with NAME=BROWN and AGE=21, and with NAME=LAWRENCE and AGE=16. These observations that do meet the matching criteria never ‘saw’ each other. During the merge process, the observations that would have met the matching criteria were never combined.

The moral to this example is that you never know if you will get the 'correct' result in a many-to-many match merge. In this example, with a small number of observations you can see that you got the wrong answer. If you are working with large numbers of observations, you will not have the opportunity to look at every possible mistake.
...PROC SQL <NOT REQUIRED>
The only way to be sure you get the correct result in a situation with observations with repeated BY
values in both data sets (or a many-to-many match) is to have each observation in data set one paired
with every observation in data set two that share a common BY-variable. There are a few ways in which
you can get the correct answer using this 'all possible pairs' matching. One way is with elaborate data
step programming. Another is to use a SAS procedure, PROC SQL (Structured Query Language). One
of the tasks that can be accomplished with PROC SQL is the merging of data sets. The SAS code
needed to properly match the two data sets used earlier by name and age range is as follows.

...Example 11.15...
proc sql;
create table both as
    select dataset1.name, dataset1.age as age1, dataset2.age as age2, hr
    from dataset1, dataset2
    where dataset1.name eq dataset2.name and
        abs(age1-age2) le 1;
quit;
proc print data=both noobs;
run;

1 PROC SQL is used to combine two data sets.
2 Data sets in PROC SQL are referred to as TABLES. The procedure creates data set BOTH.
3 Variables to be present in the data set being created must be selected from the data sets being
   combined with a SELECT statement. Those variables with common names must be referred to with
   a combination of the data set name and the variable name, e.g. dataset1.name. Variables can be
   renamed by specifying AS and a new name, sometimes called an alias. Since HR (heart rate) only
   occurs in one data set, no data set name is needed. Since we only need one name in the data set
   BOTH, the name is selected from only one data set, i.e. dataset1.
4 The two data sets being combined are specified in a FROM statement.
5 The matching criteria are listed in a WHERE statement.
6 Data set both now has all the records that meet the matching criteria and no unmatched records.

Why does PROC SQL work when a match-merge did not? In theory, PROC SQL produces a cartesian
product of all the observations in the two data sets. That is a shortcut way of saying that each
observation in dataset1 is at some point in the matching process paired with each observation in
dataset2. Once the observations are paired, the matching criteria are checked. Another way to think
about this is that there are 11 observations in dataset1 and 10 observations in dataset2, resulting in 110
possible pairs of observations (the product of multiplying 11 by 10). Each of these 110 pairs are
examined to see if they meet the matching criteria.

Notice that there are ONLY MATCHED records in the combined data set. The match-merge put both
matched and unmatched records in data set BOTH. PROC SQL as used in example 11.15 will only put
matched records in data set BOTH. With either method, match-merge or PROC SQL, there are ways to
control the types of observations that are placed in the combined data set.

PROC SQL has its own syntax, and yes, its own manual. It is very powerful and worth learning.
Combining data sets is very small part of the tasks performed with PROC SQL. The bit of knowledge to
remember for now about PROC SQL is that is one of several ways to successfully accomplish many-to-
many match-merging with SAS.
..MORE ABOUT MATCHED MERGE

There is still more to learn about combining data sets using match-merge. In the match-merge examples, there were instances when observations in one data set did not match any observations in the other data set. What if you only wanted to keep paired observations in the data set that results from a merge, i.e. observations with contributions from both data sets. The data from example 11.9 will be used again.

...Example 11.16..

```
data jan;
  input ssn weight zip : $5.;
datalines;
  001001234 180 12203
  123456789 150 13502
  888888888 222 14001
  987654321 120 12345
;
run;

data feb;
  input ssn weight;
datalines;
  001001234 115
  123456789 148
  987654321 122
;
run;

data jan_feb;
  merge jan (rename=(weight=wt1) in=d1) 1
         feb (rename=(weight=wt2) in=d2);
  by ssn;
  if d1 and d2; 2
  avg_wt = mean(of wt1-wt2);
run;

proc print data=jan_feb noobs;
  var ssn zip wt1 wt2 avg_wt;
run;
```

ssn         zip     wt1    wt2    avg_wt
001-00-1234    12203    180    115     147.5
123-45-6789    13502    150    148     149.0
987-65-4321    14001    222    122     172.0

1 A new option is added to both data sets in the merge statement, IN=. Two new variables are created with the IN= option, D1 and D2.
2 Variables D1 and D2 are checked. If an observation contributes variables to a matched observation, the value of a variable created with IN= will be 1, otherwise it will have a value of 0.
3 Data set JAN_FEB has only matched observations.

An IN option is used on each data set involved in the merge. The IN option variables, in this case D1 and D2, can be assigned any valid SAS names and these variables only exist for the duration of the data step (similar to what you have already learned about first. and last. variables). In this example, they will not appear in data set JAN_FEB. The values of D1 and D2 are contingent upon whether the data set with which they are associated contributes variables to an observation formed by the MERGE statement.

<table>
<thead>
<tr>
<th>D1</th>
<th>D2</th>
<th>CONDITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>a matched observation - variables from both JAN and FEB</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>an unmatched observation - variables from JAN ONLY</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>an unmatched observation - variables from FEB ONLY</td>
</tr>
</tbody>
</table>

When SAS evaluates the result of the statement...
if d1 and d2;

it is only true when D1=1 and D2=1. Only matched observations are kept. If we want to keep all
observations from JAN whether or not they are matched to an observation in FEB, we can rewrite the
data step as...

data jan_feb;
merge
jan (rename=(weight=wt1) in=jan)
feb (rename=(weight=wt2));
if jan;
avg_wt = mean(of wt1-wt2);
run;

The only IN= variable added to the data step is associated with data set JAN. The subsetting if
statement only checks to see if an observation contains information from an observation in the JAN data
set.

If there were extra observations in data set FEB, the changes necessary to keep all observations from
FEB whether or not they match an observation in JAN should be obvious, but just in case...

data jan_feb;
merge
jan (rename=(weight=wt1))
feb (rename=(weight=wt2) in=feb);
if feb;
avg_wt = mean(of wt1-wt2);
run;

The IN= variable(s) can also be used to create multiple data sets during a data step merge. What if all
the matched observations were to be put in one data set, and unmatched observations put in another.

...Example 11.17...

data jan;
input ssn weight zip : $5.;
datalines;
001001234 180 12203
123456789 150 13502
888888888 222 14001
987654321 120 12345
;
run;

data feb;
input ssn weight;
datalines;
777777777 101
001001234 115
123456789 148
987654321 122
000111222 300
;
run;

proc sort data=jan;
by ssn;
run;

proc sort data=feb;
by ssn;
run;
data jan_feb
only_jan (keep=ssn wt1 zip)
only_feb (keep=ssn wt2);
format ssn ssn.;
merge
jan (rename=(weight=wt1) in=jan)
feb (rename=(weight=wt2) in=feb);
by ssn;
avg_wt = mean(of wt1-wt2);
if jan and feb then output jan_feb;
else
  if jan then output only_jan;
  else output only_feb;
run;

title 'JAN_FEB';
proc print data=jan_feb noobs;
  var ssn zip wt1 wt2 avg_wt;
run;

title 'ONLY_JAN';
proc print data=only_jan noobs;
run;

title 'ONLY_FEB';
proc print data=only_feb noobs;
run;

JAN_FEB
  ssn         zip     wt1    wt2    avg_wt
  001-00-1234    12203    180    115     147.5
  123-45-6789    13502    150    148     149.0
  987-65-4321    12345    120    122     121.0

ONLY_JAN
  ssn        wt1     zip
  888-88-8888    222    14001

ONLY_FEB
  ssn        wt2
  000-11-1222    300
  777-77-7777    101

1  Data set JAN has one observation that does not match an observation in data set FEB.
2  Two observations have been added to data set FEB that do not match any observations in data set JAN.
3  Neither data set is sorted by SSN, so they are both sorted now prior to the MERGE BY SSN.
4  Three data sets are created: one with matched observations; one with unmatched observations from data set JAN; one with unmatched observations from data set FEB. Notice that KEEP data set options are used for the data sets that contain unmatched observations.
5  An IN= variable is created for each data set.
6  The IN= variables are used to determine the destination for each observation.
7  The data sets are all correct.

The default action of a data step match-merge is to create a data set with both matched and unmatched observations. You can limit the observations placed in a data set and/or create multiple data sets using a combination of IN= variables and subsetting if statements.