7

Combining Data Sets

7.1 Vertically Combining Data Sets

When the data that you work with come from different sources, you need to combine the data before you generate your report(s) or perform statistical analysis. In a situation where observations are from a different source, the data need to be combined vertically. On the other hand, when variables are from a different source, the data need to be combined horizontally. In SAS®, two methods of combining data are concatenating and interleaving data.

7.1.1 Concatenating Data Sets

Concatenating data sets refers to creating a single data set by combining two or more data sets, one after another. The syntax for concatenating data sets is as follows:

```
SET data-set(s);
```

When concatenating data sets, SAS reads all the observations from the first data set and then continues to read all the observations from the second data set, and so on. The number of observations in the combined data set is the sum of the observations from the input data sets. If the input data sets have different variables, the observations from a given data set are set to missing for variables that exist only in other data sets. For example, Program 7.1 concatenates RECORD1 and RECORD2 data sets.

RECORD1:

	Name	Course	Score
1	John	MATH	90
2	John	MATH	85
3	Mary	MATH	.
4	Tom	MATH	92

RECORD2:

	Name	Course	Grade
1	Joe	ENGLISH	96
2	John	ENGLISH	89
3	Mary	ENGLISH	78
4	Tom	ENGLISH	.
5	Dave	ENGLISH	98

Program 7.1:

```
data ex7_1;
    set record1 record2;
run;

title 'Concatenating record1 and record2';
proc print data = ex7_1;
run;
```

Output from Program 7.1:

```
                Concatenating record1 and record2
               Obs  Name   Course   Score   Grade
                1   John    MATH     90       .
                2   John    MATH     85       .
                3   Mary    MATH      .       .
                4   Tom     MATH     92       .
                5   Joe     ENGL      .      96
                6   John    ENGL      .      89
                7   Mary    ENGL      .      78
                8   Tom     ENGL      .       .
                9   Dave    ENGL      .      98
```

Log from Program 7.1:

```
629 data ex7_1;
630    set record1 record2;
631 run;

WARNING: Multiple lengths were specified for the variable Course
         by input data set(s). This may cause truncation of data.
NOTE: There were 4 observations read from the data set
      WORK.RECORD1.
NOTE: There were 5 observations read from the data set
      WORK.RECORD2.
NOTE: The data set WORK.EX7_1 has 9 observations and 4
      variables.
NOTE: DATA statement used (Total process time):
      real time      0.01 seconds
      cpu time       0.00 seconds
```

When concatenating data sets, if common variables from different input data sets have a different data *type*, SAS will stop processing and issue an error message in the log. However, when common variables have different *length, label, format,* or *informat* attributes, SAS will use the attributes from the first data set that contains the variable with that attribute and issue a warning message in the log. In Program 7.1, because the length of the COURSE in the RECORD1 data set is 4, the length of the variable COURSE is also 4 in the combined data set. Thus, "ENGLISH" is truncated to "ENGL".

During the compilation phase when data sets are concatenated, the program data vector (PDV) is created based on the descriptor information for all the data sets in the SET statement. The contents in the PDV contain all variables from each of the input data sets, as well as variables, if any, that are being created in the DATA step.

During the execution phase, SAS reads each observation from the first data set into the PDV, executes additional statements if there are any in the DATA step, and finally writes the contents from the PDV to the output data set. As expected with a SET statement, variables from the input data set are *not* set to missing each time the implicit loop of the DATA step iterates. Also as expected, when new variables are created, they will be set to missing per iteration of the DATA step. However, because multiple data sets are being concatenated, an extra step occurs when SAS reaches the end-of-file marker in the first data set. At that time, all the values in the PDV (except for automatic variables) are set to missing. Then SAS begins the same process with observations from the second data set, and so on.

Because the variables GRADE and SCORE are used separately to record testing scores for English and Math, you can create a single SCORE variable from them with the RENAME= data set option:

```
RENAME=(old-name-1=new-name-1 <...old-name-n=new-name-n>)
```

In the RENAME= data set option, *old-name* is the variable you want to rename, and *new-name* is the newly given name of the variable. Program 7.2 utilizes the RENAME= option to rename GRADE to SCORE in the RECORD2 data set. In addition, the variable COURSE is set to length of 7 by the LENGTH statement.

Program 7.2:

```
data ex7_2;
    length Course $ 7;
    set record1 record2(rename = (grade = score));
run;

title 'Renaming the GRADE variable before concatenating the
data';
proc print data = ex7_2;
run;
```

Output from Program 7.2:

```
Renaming the GRADE variable before concatenating the data
             Obs    Course       Name      Score
              1      MATH        John       90
              2      MATH        John       85
              3      MATH        Mary        .
              4      MATH        Tom        92
              5      ENGLISH     Joe        96
              6      ENGLISH     John       89
              7      ENGLISH     Mary       78
              8      ENGLISH     Tom         .
              9      ENGLISH     Dave       98
```

7.1.2 Interleaving Data Sets

Interleaving data sets utilizes BY-group processing with the SET statement to combine two or more data sets vertically:

```
SET data-set(s);
BY variable(s);
```

The number of observations in the resulting data set is the sum of the observations from all of the input data sets. The observations in the resulting data sets are arranged by the values of the BY variable(s). Within each BY group, the order of the observation is arranged by the order of the input data sets. Similar to concatenating data sets, if the input data sets have different variables, the observations from the data set are set to missing for variables that exist only in other data sets.

Before interleaving data sets, the input data sets must be sorted by the same variable(s) that you use in the BY statement. Program 7.3 begins with sorting RECORD1 and RECORD2 data sets by the NAME variable and then interleaves the sorted data sets in the DATA step.

Program 7.3:

```
proc sort data = record1 out = record1_sort;
    by Name;
run;

proc sort data = record2 out = record2_sort;
    by Name;
run;

data ex7_3;
    length Course $ 7;
    set record1_sort record2_sort;
    by Name;
run;
```

```
title 'Interleaving record1 and record2';
proc print data = ex7_3;
run;
```

Output from Program 7.3:

Interleaving record1 and record2

Obs	Course	Name	Score	Grade
1	ENGLISH	Dave	.	98
2	ENGLISH	Joe	.	96
3	MATH	John	90	.
4	MATH	John	85	.
5	ENGLISH	John	.	89
6	MATH	Mary	.	.
7	ENGLISH	Mary	.	78
8	MATH	Tom	92	.
9	ENGLISH	Tom	.	.

During the compilation phase when interleaving data sets, SAS creates FIRST.VARIABLE and LAST.VARIABLE in addition to the variables from the input data sets in the PDV.

During the execution phase, SAS reads all the observations from the first BY group from the data set that contains the observations in the first BY group. If the BY group appears in more than one data set, SAS will read the observations from the data sets in the order in which they are listed in the SET statement. The SET statement sets the variables in the PDV to missing each time SAS begins to read a new data set or when the BY group changes. This process continues until SAS has read all the observations from all the input data sets.

7.2 Horizontally Combining Data Sets

Combining data sets horizontally is required when *variables* come from different sources. Horizontal data merges are a common occurrence. For example, to obtain a complete record for a patient in a health-care study, demographic profiles, medical information, and survey questions often have to be combined. This section covers different methods of combining data sets horizontally, such as one-to-one reading, one-to-one merging, match-merging, and updating data sets.

7.2.1 One-to-One Reading

One-to-one reading utilizes multiple SET statements to combine observations from two or more input data sets independently, forming one observation that contains all of the variables from each contributing data set.

Observations are combined based on their relative position in each data set. The first observation in the first data set, for example, is combined with the first observation in the remaining data sets. The number of observations in the combined data set is equal to the number of observations in the smallest input data set. If the input data sets share common variables, only the values read in from the last data set are kept. Here's the syntax for reading two input data sets using one-to-one reading:

```
SET data-set-1;
SET data-set-2;
```

Program 7.4 utilizes one-to-one reading to combine data sets RECORD1 and RECORD2. Notice that the number of observations in the combined data set is 4, which is equal to the number of observations in the smallest input data set (RECORD1). Because variables NAME and COURSE appear in both RECORD1 and RECORD2, only the values for NAME and COURSE in RECORD2 are written to the output data set.

Program 7.4:

```
data ex7_4;
    set record1;
    set record2;
run;

title 'Use One-to-one reading to combine record1 and record2';
proc print data = ex7_4;
run;
```

Output from Program 7.4:

```
    Use One-to-one reading to combine record1 and record2
       Obs    Name   Course   Score    Grade
        1     Joe     ENGL      90       96
        2     John    ENGL      85       89
        3     Mary    ENGL       .       78
        4     Tom     ENGL      92        .
```

During the compilation phase, SAS reads the descriptor information of the data sets in all the SET statements and creates a PDV that contains all the variables from all the input data sets plus variables, if any, that are being created in the DATA step.

During the execution phase, SAS reads the first observation from the first data set into the PDV. Then the second SET statement reads the first observation from the second data set into the PDV. If, at this point, both data sets share common variables, associated values from the first data set will be overwritten by values from the second data set. After executing any additional SAS statements in the DATA step, SAS writes the contents from the PDV to the output data set.

At the beginning of the second iteration, variables in the PDV are automatically retained, except for the variables being created in the DATA step. SAS continues reading observations from one data set to another until it reaches the end-of-file marker of the data set that contains the smallest number of observations.

The output data set from Program 7.4 shows how one-to-one reading works. However, the results are incorrect because of the shared variables, NAME and COURSE. John's MATH *score* (90), for example, is combined with Joe's ENGL *grade* (96). Furthermore, the fifth observation for Dave in RECORD2 never makes it to the output.

Although one-to-one reading is not advised for combining the RECORD1 and RECORD2 data sets in Program 7.4, the merge-type works well when an IF statement is used in conjunction with a SET statement. For example, Program 7.5 merges the *mean* for SCORE calculated from RECORD1 with detail records for the *same* variable from the *same* data set. To create EX7_5, detail records for SCORE from RECORD1 are read in first. Then SAS uses an IF statement along with a second SET statement to obtain MEAN_SCORE from RECORD1_MEAN, which was created by exercising PROC MEANS beforehand. Using SET and IF statements together in this fashion ensures that SAS will not encounter an end-of-file marker that would abruptly terminate the data step. Consequently, the single MEAN_SCORE can be associated with each observation in RECORD1.

Program 7.5:

```
proc means data = record1 noprint;
    var score;
    output out = record1_mean(keep = mean_score)
    mean = mean_score;
run;

data ex7_5;
    set record1;
    if _n_ = 1 then set record1_mean;
run;

title 'Use One-to-one reading to merge the mean score with
record1';
proc print data = ex7_5;
run;
```

Output from Program 7.5:

```
    Use One-to-one reading to merge the mean score with record1

                                          mean_
      Obs    Name    Course    Score     score

       1     John     MATH       90        89
       2     John     MATH       85        89
       3     Mary     MATH        .        89
       4     Tom      MATH       92        89
```

7.2.2 One-to-One Merging

One-to-one merging uses the MERGE statement, without an accompanying BY statement, to read observations from two or more input data sets into one observation in a combined data set. The result from a one-to-one *merging* is similar to what you get with one-to-one *reading* except that one-to-one merging processes *all* observations in *all* data sets listed in the MERGE statement. The syntax for one-to-one merging is as follows:

```
MERGE data-set(s);
```

Program 7.6 uses one-to-one merging to combine RECORD1 and RECORD2. Other than the last observation for Dave in EX7_6, results match output generated in Program 7.4.

Program 7.6:

```
data ex7_6;
    merge record1 record2;
run;

title 'Use One-to-one merging to combine record1 and record2';
proc print data = ex7_6;
run;
```

Output from Program 7.6:

```
Use One-to-one merging to combine record1 and record2
   Obs    Name   Course   Score   Grade
    1     Joe     ENGL      90      96
    2     John    ENGL      85      89
    3     Mary    ENGL       .      78
    4     Tom     ENGL      92       .
    5     Dave    ENGL       .      98
```

MERGE works similarly to SET in that SAS reads the descriptor information of the input data sets during the compilation phase. SAS then creates the PDV that contains all the variables from all the data sets, along with new variables created in the DATA step.

During the execution phase with MERGE, observations from each data set are processed in the same order as they were with one-to-one reading. If two data sets have the same variables, the values from the second data set will replace the values that are read from the first data set. However, when SAS encounters the end-of-file marker from the data set with the fewest number of observations, variables in all the data sets involved with the merge will be set to missing in the PDV. Then SAS continues to read in data sets that have more observations until all observations from all data sets have been processed.

7.2.3 Match-Merging

Match-merging combines observations from two or more SAS data sets into a single observation according to the values of one or more common variables. The number of observations in the combined data set equals the sum of the largest number of observations in each BY group among all the input data sets. If the input data sets contain common variables that are not used as BY variables, values from the initial data sets listed in the MERGE statement will be overwritten by values contained in the last data set in the list when the observations being processed share the same BY values. To perform match-merging, you need to use the MERGE statement with the BY statement in the DATA step:

```
MERGE data-set(s);
BY variable(s);
```

All the input data sets must be previously sorted by the variable(s) listed in the BY statement.

The data sets being combined through match-merging can be related by one-to-one, one-to-many, or many-to-many matching of the values of one or more variables. One-to-one matching refers to a single observation in one data set relating to a single observation from another based on the value of one or more common variables. In the case of two or more common variables, one-to-one matching means that each combination of values occurs only once in each data set. One-to-many matching means a single observation in one data set is associated with multiple observations from another based on the value of one or more common variables. Many-to-many matching refers to multiple observations from each input data set can be related based on values of one or more common variables. Readers should be cautious in performing many-to-many match-merging. The MERGE statement does not produce a Cartesian product (all possible combinations of observations between the input data sets) on a many-to-many match-merge. In order to get a Cartesian product, you can use ("a" or "the") SQL procedure, which is not covered in this book.

Program 7.7 merges RECORD1_SORT and RECORD2_SORT by the common variable NAME. Both data sets have been previously sorted by NAME in Program 7.3. The variable COURSE is also a common variable in both input data sets; however, COURSE is not used as a BY variable. Thus, the values read in from RECORD2_SORT replace the values read from RECORD1_SORT. As an aside, recall that "ENGLISH" is truncated to "ENGL" with 4 bytes, because "MATH" precedes "ENGLISH" in the input data.

Notice that "John" has two observations in RECORD1_SORT but only one observation in RECORD2_SORT. Only the first value for COURSE for subject "John" is replaced with "ENGL" in WORK.EX7_7, but the second value for COURSE carries John's "MATH" observation over

from RECORD1_SORT. Because SCORE is found only in RECORD1_SORT, both values (90 and 85) in the combined data set for "John" originate in RECORD1_SORT. On the other hand, GRADE exists only in RECORD2_SORT, but because there is only one observation for "John" in RECORD2_SORT, the value for the GRADE variable (89) is carried down to the second observation of "John". Data sets RECORD1_SORT and RECORD2_SORT are displayed below so that you can follow along in this discussion.

RECORD1_SORT:

	Name	Course	Score
1	**John**	**MATH**	**90**
2	**John**	**MATH**	**85**
3	Mary	MATH	.
4	Tom	MATH	92

RECORD2_SORT:

	Name	Course	Grade
1	Dave	ENGLISH	98
2	Joe	ENGLISH	96
3	**John**	**ENGLISH**	**89**
4	Mary	ENGLISH	78
5	Tom	ENGLISH	.

Program 7.7:

```
data ex7_7;
    merge record1_sort record2_sort;
    by Name;
run;

title 'Use match-merge to combine record1 and record2';
proc print data = ex7_7;
run;
```

Output from Program 7.7:

```
Use match-merge to combine record1 and record2
Obs    Name   Course   Score   Grade
 1     Dave    ENGL      .       98
 2     Joe     ENGL      .       96
 3     John    ENGL     90       89
 4     John    MATH     85       89
 5     Mary    ENGL      .       78
 6     Tom     ENGL     92        .
```

During the compilation phase, SAS reads the descriptor information of the input data sets listed in the MERGE statement. SAS then creates the PDV that contains all the variables from all the data sets along with new variables created in the DATA step. In addition, SAS creates FIRST. VARIABLE and LAST.VARIABLE for each variable listed in the BY statement in the PDV.

During the execution phase, SAS examines the first BY group in each input data set listed in the MERGE statement to determine which BY group should appear first in the output data set. Then the DATA step starts to read observations from the first BY group from each data set. When reading the first BY group, the DATA step reads the first observations into the PDV in the order in which they appear in the MERGE statement. If a data set does not have observations in that BY group, the PDV contains missing values for the variables unique to that data set. After reading the first observation from the last data set, SAS executes other statements in the DATA step (if there are any). At the end of the DATA step, SAS writes the contents of the PDV to the output data set and returns to the beginning of the next iteration. SAS retains the values of all variables in the PDV except those variables that were created by the DATA step; SAS sets those values to missing. SAS continues to merge observations until it writes all observations from the first BY group to the output data set. When SAS has read all observations in a BY group from all data sets, it sets all variables in the PDV (except those created by SAS) to missing.

After reading all the observations from all the data sets from the first BY group, SAS begins to examine the next BY group in each data set to determine which BY group should appear next in the new data set. SAS starts to process the next BY group in each data set. The merging process continues until the DATA step reads all observations from all BY groups in all data sets.

Program 7.7 was not a logical approach to merge RECORD1 and RECORD2 because the COURSE variable does not convey any meaningful information for the combined data set. Program 7.8 improves on Program 7.7 by utilizing the DROP= and RENAME= data set options.

Program 7.8:

```
data ex7_8;
    merge record1_sort(drop = course
                       rename = (score = Math_score))
          record2_sort(drop = course
                       rename = (grade = English_score)) ;
    by Name;
run;

title 'An improved approach to merge record1 and record2';
proc print data = ex7_8;
run;
```

Output from Program 7.8:

An improved approach to merge record1 and record2

Obs	Name	Math_ score	English_ score
1	Dave	.	98
2	Joe	.	96
3	John	90	89
4	John	85	89
5	Mary	.	78
6	Tom	92	.

By default, SAS combines all the observations from all the input data sets during a match-merge. You can also exclude any unmatched observations by using the IN= data set option, which has the following form:

```
IN=variable
```

The *variable* in the IN= data set option is a temporary variable that is available in the PDV but is not included in the output data set. The value of the *variable* is either 1 or 0. The *variable* equals 1 if the input data set contributes to the current observation in the PDV; otherwise, its value equals 0. In addition to the MERGE statement, the IN= data set option can be used with the SET, MODIFY, and UPDATE statements.

Program 7.9 excludes the unmatched observations that were read from RECORD1_SORT and RECORD2_SORT. Two temporary variables, IN_RECORD1 and IN_RECORD2, are created by using the IN= data set option. IN_RECORD1 equaling 1 indicates the current observation in the PDV is read from the RECORD1_SORT data set. Similarly, IN_RECORD2 equaling 1 indicates the current observation in the PDV is read from RECORD2_SORT. A subsetting IF statement is used to include observations that are read from both RECORD1_SORT and RECORD2_SORT data sets.

Program 7.9:

```
data ex7_9;
    merge record1_sort(drop = course
                       rename = (score = Math_score)
                       in = in_record1)
          record2_sort(drop = course
                       rename = (grade = English_score)
                       in = in_record2);
     by Name;
     if in_record1 and in_record2;/*A SUBSETTING IF
     STATEMENT */
run;
```

```
title 'Excluding unmatched observations';
proc print data = ex7_9;
run;
```

Output from Program 7.9:

```
Excluding unmatched observations
                Math_   English_
Obs    Name     score     score
 1     John       90        89
 2     John       85        89
 3     Mary        .        78
 4     Tom        92         .
```

7.2.4 Updating Data Sets

You can use the UPDATE statement to update a *master* data set with a *transaction* data set. The *master* data set refers to the data set that contains the original information. The *transaction* data set refers to the data set that contains newly collected information. The UPDATE statement uses observations from the *transaction* data set to change the values of the matched observations in the *master* data set. The number of observations in the resulting data set is the sum of the observations in the *master* data set and number of unmatched observations in the *transaction* data set. To update a data set, you must use the BY statement after the UPDATE statement. The syntax for updating data sets is as follows:

```
UPDATE master-data-set transaction-data-set;
BY variable(s);
```

The *master-data-set* in the UPDATE statement names the SAS data set that contains the original information, while *transaction-data-set* is the name of the SAS data set that contains new information. When the *transaction* data set contains duplicate values of the BY variable, only the last values copied to the PDV are written to the output data set. If the *master* data set contains duplicate values of the BY variable, only the first observation in the *master* data set is updated and SAS will write a warning message in the log.

Updating data sets is similar to match-merging with the MERGE statement. However, unlike MERGE, missing values in the *transaction* data set do not replace the existing values in the *master* data set. Thus, if you would like to update only some observations but not all observations of a specific variable, you can set the observations you do not want to change to missing in the *transaction* data set.

For example, some of the test scores in the RECORD2_SORT data set need to be modified. In Program 7.10, the *transaction* data set, RECORD3_SORT, contains newly collected scores that are stored in the GRADE

variable. Program 7.10 updates RECORD2_SORT with values from GRADE in RECORD3_SORT. Notice that the values for GRADE for Joe and John are missing in RECORD3_SORT. Thus, in the final updated data set, the values for Joe and John originate in the *master* data set (RECORD2_SORT).
RECORD3:

	Name	Grade
1	Joe	.
2	John	.
3	Mary	82
4	Tom	90
5	Dave	97

Program 7.10:

```
proc sort data = record3 out = record3_sort;
    by Name;
run;

data ex7_10;
    update record2_sort record3_sort;
    by name;
run;

title 'Updating record2 data set';
proc print data = ex7_10;
run;
```

Output from Program 7.10:

```
Updating record2 data set
Obs   Name    Course    Grade
 1    Dave    ENGLISH     97
 2    Joe     ENGLISH     96
 3    John    ENGLISH     89
 4    Mary    ENGLISH     82
 5    Tom     ENGLISH     90
```

Exercises

Exercise 7.1. Consider the following data set, GROUP_SCORE.SAS7BDAT. Use one-to-one reading to add one variable, MAXSCORE, that contains the maximum score of the SCORE variable.

	Group	Score
1	A	6
2	A	9
3	B	2
4	A	8
5	A	8
6	A	9
7	A	6
8	A	6
9	B	1
10	B	2

Exercise 7.2. Based on the same data set in Exercise 7.1, GROUP_SCORE. SAS7BDAT, use the match-merge method to add one variable, MEANSCORE, that contains the mean of the SCORE variable for each group.