Combining, Combining, Combining, Splitting, Splitting, Splitting
Emmy Pahmer, MDS Pharma Services, St. Laurent, Québec

ABSTRACT
Combining and Splitting data sets are among the most commonly performed data management tasks and as with everything in SAS® there are many ways to go about it. This paper is for new SAS users and will show a progression of examples using the data step and proc SQL, discussing the way each of them functions. Some of the advantages and disadvantages of each will be presented.

INTRODUCTION
The code is provided to offer examples of the syntax for many situations. Some general principles will be considered such as terminology, different types of merges/joins and subsetting. We consider how often we read the data, if sorting is required, how much coding is involved, and quality control checking. The differences between the data step and SQL will be discussed. The code is presented first followed by comments.

DATA STEP VS. SQL
Merging looks at sorted observations from the source datasets, checks if the observations match and if they do, they get combined and output. SQL creates a full Cartesian join (everything by everything) then selects the records that correspond to the criteria. For more information on this, many good papers have been written that will mentioned at the end.

SPLITTING, SPLITTING, SPLITTING
The objective is to divide a dataset into subgroups by the values of a variable. Here is our first sample dataset:

<table>
<thead>
<tr>
<th>Age</th>
<th>Who</th>
<th>Sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Annie</td>
<td>F</td>
</tr>
<tr>
<td>15</td>
<td>Bill</td>
<td>M</td>
</tr>
<tr>
<td>20</td>
<td>Chandra</td>
<td>F</td>
</tr>
<tr>
<td>25</td>
<td>David</td>
<td>M</td>
</tr>
<tr>
<td>30</td>
<td>Eleanor</td>
<td>G</td>
</tr>
</tbody>
</table>

1) USING IF TO SUBSET AND OUTPUT – TWO DATA STEPS
When first starting out you might use two steps to separate this dataset into two groups. First selecting the males.

```sas
data males;
  set everyone;
  if sex = 'M';
  * implicit output;
run;
```

Then the females.

```sas
data females;
  set everyone;
  if sex = 'F';
run;
```
Results:

MALES
Age | Who | Sex
-- | -- | --
15  | Bill | M
25  | David | M

FEMALES
Age | Who | Sex
-- | -- | --
10  | Annie | F
20  | Chandra | F

This produces the correct results but is reading the data twice and is not efficient, therefore not recommended. Also we lost Eleanor’s observation which has an invalid value for the Sex variable.

2) USING IF TO SUBSET AND OUTPUT – ONE DATA STEP

data males females;
  set everyone;
  if sex = 'F' then output females;
  else if sex = 'M' then output males;
run;

Here we also use the data step but we output to two different datasets in the Data statement. The results are also correct but it is more efficient.

data males females;
  set everyone;
  if sex = 'F' then output females;
  else if sex = 'M' then output males;
  else put "Neither F nor M - check " _all_; *or output to another dataset ;
run;

We can add quality control code.

3) USING WHERE DATASET OPTION TO SUBSET IN THE DATA STATEMENT – TWO DATA STEPS

data female (where= (sex = 'F'));
  set everyone ;
run;

data male (where= (sex = 'M'));
  set everyone ;
run;

Data that we don’t need to keep is being read and using two steps is not efficient therefore it is not recommended.

4) USING WHERE TO SUBSET IN THE DATA STATEMENT – ONE DATA STEP

data female (where=(sex='F')) male (where=(sex='M')) ;
  set everyone ;
run;

Again, we’re reading all the data, subsetting in the data statement but using one step. Data is being read once so it is efficient. Only a small amount of code is required and it’s clear. If manipulation of the values for Sex occurs in the data step, the WHERE clause in the DATA statement, selecting them at the end, is appropriate.
data female (where=(sex='F')) male (where=(sex='M')) checkothers (where = (sex not in ("M","F")))
   set everyone ;
run;

proc print data=checkothers;
run;

Here is the same step with some quality control checking.

5) USING WHERE TO SUBSET IN THE SET STATEMENT – TWO DATA STEPS

data female;
   set everyone (where=(sex='F')) ;
run;

data male;
   set everyone (where=(sex='M')) ;
run;

Although you're reading data twice, the amount is reduced for each step. This would be recommended if you only need to output one subset and can do so at the beginning of the data step, ie, don't need to manipulate the values.

6) PROC SQL USING WHERE STATEMENT

proc sql;
create table males as
   select *
   from everyone
   where sex = 'M';
quit;

7) PROC SQL USING WHERE DATASET OPTION

proc sql;
create table males as
   select *
   from everyone (where=(sex='M'));
quit;

Another SQL example that has the same result. You can also use other dataset options such as KEEP, DROP and RENAME.

8) CREATING BOTH IN THE SAME SQL PROCEDURE

proc sql;
create table males as
   select *
   from everyone (where=(sex='M'));
create table females as
   select *
   from everyone (where=(sex='F'));
quit;
9) SUBSETTING SOME VARIABLES INTO ONE DATASET AND OTHER VARIABLES INTO ANOTHER WITH THE DATA STEP

```plaintext
data age (keep = who age) sex (keep = who sex);
set everyone;
run;
```

10) SUBSETTING SOME VARIABLES INTO ONE DATASET AND OTHER VARIABLES INTO ANOTHER WITH SQL

```plaintext.proc sql;
create table age as
select who, age
from everyone;
create table sex as
select who, sex
from everyone;
quit;
```

COMBINING, COMBINING, COMBINING

Using the MERGE statement is usually “match-merging” since we merge on the basis of matching observations from one dataset to the observations in another. In our example we want to match the observations in Everyone to the observations in Activity by the Who variable. We always have to ensure that the datasets to be combined are sorted by the BY variables prior to merging. As for whether the results are correct, it depends on how you hope to combine the data. If you omit the BY variables the results will probably be wrong. You will get a message in the log but not an error. The BY variables should have the same name, length and justification. If the BY variable is left-justified in one dataset and right-justified in the other, the BY variables will not match and the resulting dataset will be wrong but there will be no message, no error. Doing quality control checking of the results should be standard practice. Be aware that if you have variables with the same name that are not BY variables, the values from the first dataset will be overwritten by values in the second.

Sample datasets to combine:
EVERYONE (same as in earlier, 5 observations)

<table>
<thead>
<tr>
<th>Age</th>
<th>Who</th>
<th>Sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Annie</td>
<td>F</td>
</tr>
<tr>
<td>15</td>
<td>Bill</td>
<td>M</td>
</tr>
<tr>
<td>20</td>
<td>Chandra</td>
<td>F</td>
</tr>
<tr>
<td>25</td>
<td>David</td>
<td>M</td>
</tr>
<tr>
<td>30</td>
<td>Eleanor</td>
<td>G</td>
</tr>
</tbody>
</table>

ACTIVITY (6 observations, not sorted)

<table>
<thead>
<tr>
<th>Who</th>
<th>Activity</th>
<th>Sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Igor</td>
<td>Drinking</td>
<td>M</td>
</tr>
<tr>
<td>Karen</td>
<td>Swimming</td>
<td>F</td>
</tr>
<tr>
<td>Jose</td>
<td>Rugby</td>
<td>M</td>
</tr>
<tr>
<td>Chandra</td>
<td>Mariachi</td>
<td>F</td>
</tr>
<tr>
<td>Bill</td>
<td>Karate</td>
<td>F</td>
</tr>
<tr>
<td>Annie</td>
<td>Juggling</td>
<td>F</td>
</tr>
</tbody>
</table>

11) MERGING - KEEPING ALL DATA INCLUDING UNMATCHED OBSERVATIONS

```plaintext.proc sort data=activity;
by who;
run;
```
proc sort data=everyone;
   by who;
run;

data combined_11;
   merge everyone activity;
   by who;
run;

Results:
<table>
<thead>
<tr>
<th>Age</th>
<th>Who</th>
<th>Sex</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Annie</td>
<td>F</td>
<td>Juggling</td>
</tr>
<tr>
<td>15</td>
<td>Bill</td>
<td>F</td>
<td>Karate</td>
</tr>
<tr>
<td>20</td>
<td>Chandra</td>
<td>F</td>
<td>Mariachi</td>
</tr>
<tr>
<td>25</td>
<td>David</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>Eleanor</td>
<td>G</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>Igor</td>
<td>M</td>
<td>Drinking</td>
</tr>
<tr>
<td>.</td>
<td>Jose</td>
<td>M</td>
<td>Rugby</td>
</tr>
<tr>
<td>.</td>
<td>Karen</td>
<td>F</td>
<td>Swimming</td>
</tr>
</tbody>
</table>

This is a full merge, called a full outer join in SQL, containing all data from both datasets including those that don't have a match in the other dataset. With merge, the datasets have to be sorted in advance and the resulting dataset is sorted. Note that the value of Sex for Bill has become 'F' since Sex had a value of 'F' in the second dataset which overwrote the value of 'M' from the first dataset. This will be the case for all the following merges.

12) INNER – DATA FROM BOTH DATASETS MUST MATCH

Results:
<table>
<thead>
<tr>
<th>Age</th>
<th>Who</th>
<th>Sex</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Annie</td>
<td>F</td>
<td>Juggling</td>
</tr>
<tr>
<td>15</td>
<td>Bill</td>
<td>F</td>
<td>Karate</td>
</tr>
<tr>
<td>20</td>
<td>Chandra</td>
<td>F</td>
<td>Mariachi</td>
</tr>
</tbody>
</table>

Because of the IF statement, only matching data is kept - three observations. The "if in_a and in_b" statement is equal to "if in_a = 1 and in_b = 1". This would be called an inner join in SQL.

13) ONE-SIDED MERGE – MUST EXIST IN ONE OF THE DATASETS
Results:

<table>
<thead>
<tr>
<th>Age</th>
<th>Who</th>
<th>Sex</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Annie</td>
<td>F</td>
<td>Juggling</td>
</tr>
<tr>
<td>15</td>
<td>Bill</td>
<td>F</td>
<td>Karate</td>
</tr>
<tr>
<td>20</td>
<td>Chandra</td>
<td>F</td>
<td>Mariachi</td>
</tr>
<tr>
<td>25</td>
<td>David</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>Eleanor</td>
<td>G</td>
<td></td>
</tr>
</tbody>
</table>

This merge keeps all of the data in the first (Everyone) dataset and whatever data that matches from the second (Activity) dataset.

14) MERGE WITH SOME CHECKING

<sorting>

data combined_14;
  merge everyone (in = in_a) activity (in = in_b);
  by who;
  if in_a and in_b then output;
  else if in_a then put "In A only: " Who=; *or output to another dataset;
  else if in_b then put "In B only: " Who=;
run;

Messages in Log:
In A only: Who=David
In A only: Who=Eleanor
In B only: Who=Igor
In B only: Who=Jose
In B only: Who=Karen

It’s a good idea to check the results of your merge. You can do so by outputting to different datasets for observations that don’t match or by sending messages to the log.

15) PROC SQL INNER JOIN

proc sql;
  create table combined_15a as
  select a.*, b.activity, b.sex
  from everyone as a, activity as b
  where a.who = b.who
  order by activity;
quit;

Different syntax for inner join:

proc sql;
  create table combine_15b as
  select a.*, b.activity, b.sex
  from everyone as a inner join activity as b
  on a.who = b.who;
quit;
Results for both:

<table>
<thead>
<tr>
<th>Age</th>
<th>Who</th>
<th>Sex</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Annie</td>
<td>F</td>
<td>Juggling</td>
</tr>
<tr>
<td>15</td>
<td>Bill</td>
<td>M</td>
<td>Karate</td>
</tr>
<tr>
<td>20</td>
<td>Chandra</td>
<td>F</td>
<td>Mariachi</td>
</tr>
</tbody>
</table>

WARNING: Variable Sex already exists on file WORK.COMBINED_15A.

The above two syntaxes generate almost the same results as example 12 with one exception. The value of Bill is now 'M' rather than 'F'. With SQL, if there are non-join variables with the same name (in this case Sex), the values of the left table are kept rather than the right. A warning message is generated in the log. With Proc SQL no sorting is required. The table (dataset) being created can be sorted in proc SQL as well with an ORDER BY clause. The join variables (equivalent to BY variables) may or may not have the same name.

16) PROC SQL LEFT JOIN

```sql
proc sql;
create table combined_16 as
select a.*, b.activity, b.sex
from everyone as a left join activity as b
on a.who = b.who
;
quit;
```

Results:

<table>
<thead>
<tr>
<th>Age</th>
<th>Who</th>
<th>Sex</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Annie</td>
<td>F</td>
<td>Juggling</td>
</tr>
<tr>
<td>15</td>
<td>Bill</td>
<td>M</td>
<td>Karate</td>
</tr>
<tr>
<td>20</td>
<td>Chandra</td>
<td>F</td>
<td>Mariachi</td>
</tr>
<tr>
<td>25</td>
<td>David</td>
<td>M</td>
<td>Drinking</td>
</tr>
<tr>
<td>30</td>
<td>Eleanor</td>
<td>G</td>
<td>Rugby</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Swimming</td>
</tr>
</tbody>
</table>

A left join takes all the records in the left table (matching or not) with only the matching records from the right table. This example is similar to example 13 except for the value of Sex for Bill as in the previous example.

17) PROC SQL RIGHT JOIN

```sql
proc sql;
create table combined_17 as
select a.*, b.activity, b.sex
from everyone as a right join activity as b
on a.who = b.who
;
quit;
```

Results:

<table>
<thead>
<tr>
<th>Age</th>
<th>Who</th>
<th>Sex</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Annie</td>
<td>F</td>
<td>Juggling</td>
</tr>
<tr>
<td>15</td>
<td>Bill</td>
<td>M</td>
<td>Karate</td>
</tr>
<tr>
<td>20</td>
<td>Chandra</td>
<td>F</td>
<td>Mariachi</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Drinking</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Rugby</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Swimming</td>
</tr>
</tbody>
</table>

The results of this example are perhaps not what we would expect. We have missing values for Who and Sex. When the join occurs, for variables that have the same name, the first values are taken so although the second table had values for Who and Sex, they are set to missing because the first had no matching observations therefore no values.
We get the same warning message about variables already being in the dataset. If possible, you should use a left join by reversing the order of the tables in the statement.

18) PROC SQL RIGHT JOIN WITH COALESCE

    proc sql;
    create table combined_18 as
    select coalesce (a.who, b.who) as Who2, a.age, b.activity, coalesce (a.sex, b.sex) as Sex2
    from everyone as a right join activity as b
    on a.who = b.who
    ;
    quit;

Results:

<table>
<thead>
<tr>
<th>Who2</th>
<th>Age</th>
<th>Activity</th>
<th>Sex2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Annie</td>
<td>10</td>
<td>Juggling</td>
<td>F</td>
</tr>
<tr>
<td>Bill</td>
<td>15</td>
<td>Karate</td>
<td>M</td>
</tr>
<tr>
<td>Chandra</td>
<td>20</td>
<td>Mariachi</td>
<td>F</td>
</tr>
<tr>
<td>Igor</td>
<td>.</td>
<td>Drinking</td>
<td>M</td>
</tr>
<tr>
<td>Jose</td>
<td>.</td>
<td>Rugby</td>
<td>M</td>
</tr>
<tr>
<td>Karen</td>
<td>.</td>
<td>Swimming</td>
<td>F</td>
</tr>
</tbody>
</table>

Using the COALESCE function fixes the problem by keeping values if they exist in either of the source tables. COALESCE returns the first nonmissing value that it encounters so Bill has the value of M for Sex2. FULL joins also work the same way as RIGHT joins so either COALESCE should be used or a match-merge.

19) "SETTING" MULTIPLE DATASETS

Previous examples were to match observations from one dataset to observations in another dataset. It may happen that you need to “stack” data rather than to match data.

Starting datasets:

<table>
<thead>
<tr>
<th>Age</th>
<th>Who</th>
<th>Sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Annie</td>
<td>F</td>
</tr>
<tr>
<td>15</td>
<td>Bill</td>
<td>M</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Age</th>
<th>Who</th>
<th>Sex</th>
<th>Pet</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Chandra</td>
<td>F</td>
<td>dog</td>
</tr>
<tr>
<td>25</td>
<td>David</td>
<td>M</td>
<td>monkey</td>
</tr>
<tr>
<td>30</td>
<td>Eleanor</td>
<td>F</td>
<td>yak</td>
</tr>
</tbody>
</table>

data combine_19;
    set first second;
run;

Results:

<table>
<thead>
<tr>
<th>Age</th>
<th>Who</th>
<th>Sex</th>
<th>Pet</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Annie</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Bill</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Chandra</td>
<td>F</td>
<td>dog</td>
</tr>
<tr>
<td>25</td>
<td>David</td>
<td>M</td>
<td>monkey</td>
</tr>
<tr>
<td>30</td>
<td>Eleanor</td>
<td>F</td>
<td>yak</td>
</tr>
</tbody>
</table>

This example shows how to combine datasets with similar attributes as shown in the results.
20) MANY-TO-MANY MERGE/JOIN
Merges where you have repeats of BY variables in both datasets are rarely desired. It should be noted that, once again, merge and SQL operate differently. See the reference papers.

CONCLUSION
A number of examples have been provided to combine and split datasets. There are many factors involved as to which ones you should use, the most important being that they achieve the desired results. Speed in run time may be a concern for large datasets. Also you need to think about how long it takes to code and how readable the code is for other people. Adding explanatory comments in the programming code is recommended. Always make sure that your results are as desired by adding checks either in the same step/proc or afterwards.

REFERENCES


ACKNOWLEDGMENTS
Thanks to Marie-Therese Lai-Tung and Igor Rubets.

CONTACT INFORMATION
Your comments and questions are valued and encouraged. Contact the author at:
Emmy Pahmer
MDS Pharma Services
St. Laurent, Québec
Work Phone: (514) 333-0042 ext. 4222
E-mail: emmy.pahmer@mdsinc.com

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 trademarks of their respective companies.