Paper 83-27

Splitting a Large SAS® Data Set

John R. Gerlach, NDC Health; Yardley, PA
Simant Misra, NDC Health; Phoenix, AZ

Abstract

Imagine that you have a very large SAS® data set that you must split into smaller, more manageable, data sets. Typically, you would write a Data step that employs IF / THEN / ELSE logic, along with the respective OUTPUT statements, each designating the appropriate smaller data sets. Of course, in order to partition the input data set into smaller data sets having similar cardinality (# observations), you must know how many observations there are in the input data set, then write the logic, accordingly. But, such a Data step might be tedious to write. This paper explains the %split macro that writes the Data step automatically to produce the desired smaller data sets.

Introduction

Data analysis often involves large amounts of data such that there may be processing and storage issues. That is, it may be convenient to break-up a large SAS data set into smaller more manageable data sets, thereby partitioning the job or storing the data more conveniently. Of course, writing a Data step would be a reasonable method for accomplishing this task., as illustrated below.

```
data < enumerated data sets >;
   set < input data set >;
   if _n_ le < ith split point >
      then output < ith data set >;
      else if _n_ le < jth split point >
         then output < jth split point >;

          < ad nauseam >

run;
```

The Data step utilizes the automatic variable _N_ along with the appropriate IF / THEN / ELSE logic that determines how to distribute the observations to the several target data sets. That is, the IF logic executes the appropriate OUTPUT statement based on the *i*th iteration of the Data step (denoted by the value of _N_), then writes the observation to the respective data set.

It is desirable that the target data sets should have comparable cardinality, that is, the same number of observations in each. This becomes an important aspect of the automated solution and poses an interesting caveat, which is discussed later in this paper.

The %split Macro

The %split macro, illustrated below, formulates a suitable Data step that will process a large SAS data set and creates a pre-defined number of smaller data sets having comparable cardinality. The macro has a key parameter to ensure that at least two data sets will be created; otherwise, the user specifies the desired number of smaller data sets, for example, %split(ndsn=7).

```
%macro split(ndsn=2);
   data %do i = 1 %to &ndsn.; dsn&i. %end; ;
      retain x;
      set orig nobs=nobs;
      if _n_ eq 1
         then do;
            if mod(nobs,&ndsn.) eq 0
               then x=int(nobs/&ndsn.);
               else x=int(nobs/&ndsn.)+1;
            end;
      if _n_ le x then output dsn1;
         %do i = 2 %to &ndsn.;
            else if _n_ le (&i.*x)
               then output dsn&i.;
            %end;
   run;
%mend split;
```

The macro formulates the Data step by first completing the DATA statement, that is, it enumerates the several target data sets using the %DO loop; a simple task, since the number of target data sets is known, *a priori*. That's the easy part of this SAS solution. The more challenging aspect concerns the actual splitting such that each target data set contains the same number of observations. There are two necessary criteria:

- How many observations are in the input data set?
- How many target data sets will be created?

The NOBS option of the SET statement provides the answer to the first question; whereas, the user answers the second question, accordingly.

Determining the cardinality of the target data sets requires a bit more effort. The naïve solution would be to divide the number of observations in the input data set by the number of desired target data sets. But, this solution fails when the two aforementioned criteria do not divide evenly, that is, not all observations would be distributed. Keep in mind that the objective is to generate *n* data sets having comparable cardinality. Conversely, it

is undesirable to have disproportionate sized data sets or, even worse, to have extraneous empty data sets.

Notice how the %split macro generates the IF / THEN / ELSE logic using the automatic variable _N_ and the computed cut-off (based on a factor of the variable *x*), which determines how the observations are distributed to the several data sets.

Examples

To better understand how the %split macro works, consider the following examples that processes a contrived test data set. The MPRINT option shows in the SAS log how the macro resolves.

```
options mprint;

data orig;
   do i = 1 to 82; output; end;
run;
```

The first simple example splits the original data set into two data sets called DSN1 and DSN2, each having 41 observations. Why 41 observations? Simply because 2 and 41 are integer factors of 82, the number of observations in the original data set. Observe how the macro resolves.

```
%split(ndsn=2)  ;

MPRINT(SPLIT):   data dsn1 dsn2 ;
MPRINT(SPLIT):   retain x;
MPRINT(SPLIT):   set orig nobs=nobs;
MPRINT(SPLIT):   if _n_ eq 1 then x=int(nobs/2);
MPRINT(SPLIT):   else x=int(nobs/2)+1;
MPRINT(SPLIT):   if _n_ le x then output dsn1;
MPRINT(SPLIT):   else if _n_ le (2*x)
                 then output dsn2;
MPRINT(SPLIT):   run;
```

The next example splits the same original data set into 4 data sets, called DSN1 – DSN4. However, the data sets are not distributed evenly, since 4 is not an even factor of 82. Thus, data sets DSN1-DSN3 have 21 observations; whereas, DSN4 contains 19 observations.

```
%split(ndsn=4)   ;

MPRINT(SPLIT):   data dsn1 dsn2 dsn3 dsn4 ;
MPRINT(SPLIT):   retain x;
MPRINT(SPLIT):   set orig nobs=nobs;
MPRINT(SPLIT):   if _n_ eq 1 then x=int(nobs/4);
MPRINT(SPLIT):   else x=int(nobs/4)+1;
MPRINT(SPLIT):   if _n_ le x then output dsn1;
MPRINT(SPLIT):   else if _n_ le (2*x)
                 then output dsn2;
MPRINT(SPLIT):   else if _n_ le (3*x)
                 then output dsn3;
MPRINT(SPLIT):   else if _n_ le (4*x)
                 then output dsn4;
MPRINT(SPLIT):   run;
```

Again, the objective is to generate smaller data sets having comparable cardinality. But, since 4 is not an even factor of 82, the macro uses the MOD function to adjust the multiplier *x* in order to distribute all the observations, as well as to attain comparable cardinality.

```
if _n_ eq 1
   then do;
      if mod(nobs,&ndsn.) eq 0
         then x=int(nobs/&ndsn.);
         else x=int(nobs/&ndsn.)+1;
      end;
```

Error or Caveat

The next example poses a caveat in this solution. What happens when you split a data set containing 82 observations into 43 separate data sets? The %split macro generates a Data step that will produce 43 data sets. But, how many observations will be in each of the 43 data sets? Will all data sets contain observations? Is this a reasonable partitioning of the input data set?

```
%split(ndsn=43)  ;
```

Because 43 is not an even factor of 82, the multiplier is adjusted to the value of 2. Consequently, the data sets DSN1 through DSN41 will have two observations; but, the data sets DSN42 and DSN43 will be empty.

```
MPRINT(SPLIT):   if _n_ le x then output dsn1;
MPRINT(SPLIT):   else if _n_ le (2*x) then
                 output dsn2;
       :      :      :      :      :
MPRINT(SPLIT):   else if _n_ le (42*x) then
                 output dsn42;
MPRINT(SPLIT):   else if _n_ le (43*x) then
                 output dsn43;
MPRINT(SPLIT):   run;
```

Conclusion

The %split macro is a useful tool for partitioning large data sets into more manageable data sets; and, it offers a good lesson in the Macro Language and, even, number theory.

Author Information

John R. Gerlach
NDC Health
Yardley, PA

Simant Misra
NDC Health
Phoenix, AZ