

RETAINing Your Sanity: 5 Ideas to Manipulate Data Using the RETAIN Statement

Matthew Taylor, Bank of America

ABSTRACT

SAS® PROCs are generally structured to work with long, vertically oriented datasets. The challenge with datasets that are created this way is that it is very easy to work with data across observations but it is cumbersome to try to work with data vertically. Thus enters the RETAIN statement, a way to hold values across observations and assist in those comparisons. This paper will review five ways to use the RETAIN statement to solve common dataset related problems.

INTRODUCTION

The RETAIN statement in its simplest form allows a value from a variable to be held in memory during a DATA step for use in the next observation in the series. This functionality will enable the user to work across vertical rows that would otherwise be very difficult to negotiate. Let's walk through a simple example to see what the RETAIN does, here is a simple DATA step and an example of the RETAIN statement:

```
data example2;
    input name $ amount;
    datalines;
Karen 323
Angela 321
Debbie 543
Josh 444
;
run;

data example2;
set example2;
retain total 0;
total=total+amount;
run;
```

The syntax of the RETAIN statement is straightforward, the word retain plus the name of the variable you want to create. Note that a RETAIN statement variable will have a default value set to missing. This is why we are initializing the variable with a zero to complete the summation as we intended.

Figure 1. Results From The Running Total Program

Obs	name	amount	total
1	Karen	323	323
2	Angela	321	644
3	Debbie	543	1187
4	Josh	444	1631

Figure 1. Results From The Running Total Program

So what is happening in this example? The value of total is created and initialized at zero. Upon SAS® returning the first record, the value in amount (323) is added to the value zero and returned in the total column. This continues for every record with SAS® holding the value for each record until the total is completed. Now that we have the basics completed, let's start looking at some interesting ways to use the RETAIN statement.

CREATING A COUNTER VARIABLE

The `_N_` works as a silent counter in your SAS® dataset, but there are occasions when you need to count the number of records in an actual variable. The RETAIN statement allows you to do this and much more. You can also count duplicate records using BY variable processing and later use that to select specific records out of your dataset for your purposes. First, we will create a simple counter for each row, here is our base data:

```
data example1;
    input name $ dept $ account balance;
    datalines;
Karen ACC 12345 54021
Angela ACC 12346 43121
Debbie FIN 12347 31211
Josh MGT 12348 12412
Karen ACC 12345 21243
Debbie FIN 12347 3231
Angela ACC 12346 1211
;
run;

data example1;
set example1;
retain counter 0;
counter=counter+1;
run;
```

Figure 2. Results From Adding a Permanent Counter Variable

Obs	name	dept	account	balance	counter
1	Karen	ACC	12345	54021	1
2	Angela	ACC	12346	43121	2
3	Debbie	FIN	12347	31211	3
4	Josh	MGT	12348	12412	4
5	Karen	ACC	12345	21243	5
6	Debbie	FIN	12347	3231	6
7	Angela	ACC	12346	1211	7

Figure 2. Results From Adding a Permanent Counter Variable

The resulting data shows us a numeric counter with one record for each row in the dataset. This simplified counter field gives you a permanent equivalent to the field `_N_`. Adding BY variable processing increases the uses of the RETAIN counter. Let's take a look at this approach to creating running totals.

UTILIZING TO CREATE RUNNING TOTALS

The next level of RETAIN statement use would be using the FIRST. and LAST. syntax in conjunction with BY variable processing. This can allow you to create running totals and subtotals in your datasets which can be very helpful depending on what type of output the user wishes to create. We will sort our data by name which as seen above has duplicate records. Adding in BY variable processing to our DATA step allows us to create a counter per BY value which can then identify duplicates in our existing data and allow us the control to pick which duplicate we are interested in analyzing.

```
proc sort data=example1;
by name;
run;

data example1;
set example1;
by name;
retain counter2 0;
if first.name then counter2=1;
else counter2=counter2+1;
run;
```

In order to properly use the BY statement in SAS® you must sort that BY variable ahead of the DATA step. In this example we utilize the FIRST. syntax to set the value of the first observation as one, using the RETAIN statement to increment the counter for each by variable.

Figure 3. Counter Variable using BY processing

Obs	name	dept	account	balance	counter2
1	Angela	ACC	12346	43121	1
2	Angela	ACC	12346	1211	2
3	Debbie	FIN	12347	31211	1
4	Debbie	FIN	12347	3231	2
5	Josh	MGT	12348	12412	1
6	Karen	ACC	12345	54021	1
7	Karen	ACC	12345	21243	2

Figure 3. Counter Variable using BY processing

This result provides us with a number of possibilities. We could sort by additional values to choose a preferred row if there were known duplicates that we valued. We can also quickly ascertain how many duplicates are included in the data source and how many multiples exist for each. This allows the user to separate the data into pieces and quickly isolate potential duplicate values in their own separate dataset. This type of coding will also allow for running totals and subtotals if we had used it on the balance field instead.

COMPARING VALUES ACROSS ROWS

When processing data in a vertical dataset, one issue that can arise is the desire to compare data between vertical values. This can also be achieved with the help of the RETAIN statement and FIRST. and LAST. processing. By focusing on one occurrence and holding the value using the RETAIN statement, one can compare across records and select those that match. We will be using the same data as before, only this time sorted by balance in order to calculate the median value:

```
data example1;
input name $ dept $ account balance;
datalines;
```

```

Karen ACC 12345 54021
Angela ACC 12346 43121
Debbie FIN 12347 31211
Josh MGT 12348 12412
Karen ACC 12345 21243
Debbie FIN 12347 3231
Angela ACC 12346 1211
;
run;

proc sort data=example1;
by balance;
run;

```

The following code calculates the median for the dataset and then resorts the data to put the median value at the top. One of the limitations of the RETAIN statement is that DATA step processing always starts with the top of the dataset and moves downward. Therefore we can't use a value for comparison that doesn't appear in row 1 of our dataset. Here is the code for the 2nd part of this process:

```

data example1;
set example1 nobs=numobs;
if _N_=round(numobs/2) then med=balance;
run;

proc sort data=example1;
by descending med;
run;

```

The final part of the code is where we use the RETAIN statement. We keep the value of median and compare it across the rows to result in a flag that categorizes each row as the median, high or low:

```

data example1(drop=median med);
set example1;
length flag $6.;
retain median;
if _N_=1 then median=med;
if balance=median then flag='Median';
else if balance<median then flag='Low';
else if balance>median then flag='High';
run;

```

Figure 4. Flag Derived Off Comparison of Median Balance

Obs	name	dept	account	balance	flag
1	Karen	ACC	12345	21243	Median
2	Angela	ACC	12346	1211	Low
3	Debbie	FIN	12347	3231	Low
4	Josh	MGT	12348	12412	Low
5	Debbie	FIN	12347	31211	High
6	Angela	ACC	12346	43121	High
7	Karen	ACC	12345	54021	High

Figure 4. Flag Derived Off Comparison of Median Balance

The comparison is complete and using the RETAIN statement allowed us to carry forward the value of the median and apply it to all rows within the SAS® dataset. The next item we are going to investigate is regarding data that appears incomplete and needs the RETAIN statement to help fill in the blanks.

FILLING IN MISSING DATA

The RETAIN statement can also be used in special circumstances to help fill in data that may not be populated for all your records. However, you must be careful due to the nature of how the statement works, it can produce unforeseen results without proper coding. Let's look at an example:

```
data test;
  input name $ balance;
  /*store not empty ID in different retained variable*/
datalines;
Karen 54021
. 43121
. 21423
Angela 4213
. 54656
. 5345
;
run;

data test(drop=newname);
set test;
retain newname;
if not missing(name) then newname=name;
else name=newname;
run;
```

This data has values for balance but there is only one name over the group of records. In this case we can fill in the records using the RETAIN statement. The result of this code is the name field being filled in through the use of the interim variable "newname." The code looks for the values of name that are populated and using the RETAIN the value can be held in memory. As the dataset iterates to the next record, the value is filled in with the value of newname.

Figure 5. Filling in Data Using the RETAIN Statement

Obs	name	balance
1	Karen	54021
2	Karen	43121
3	Karen	21423
4	Angela	4213
5	Angela	54656
6	Angela	5345

Figure 5. Filling in Data Using the RETAIN Statement

This allows you to fill in the missing data with values that were populated in other records. It can be used in a variety of different data structures to help fill in the blanks.

ORDERING VARIABLES IN YOUR SAS® DATASET

The final tip for using a RETAIN statement is in the ordering of variables. In some scenarios, the order of the variables in your dataset is important and helps for display during reporting. There are a few ways to get variables arranged in proper order and one of the key ways is simply using a RETAIN statement. We will use the data that has been shown in previous examples:

```
data example1;
    input name $ dept $ account balance;
    datalines;
Karen ACC 12345 54021
Angela ACC 12346 43121
Debbie FIN 12347 31211
Josh MGT 12348 12412
Karen ACC 12345 21243
Debbie FIN 12347 3231
Angela ACC 12346 1211
;
```

This time we wish to order the variables in the dataset a certain way. That can be done using the RETAIN statement. The key is putting the RETAIN statement before the SET statement:

```
data example5;
    retain name account balance dept;
    set example1;
run;
```

Figure 6. Ordering Variables Using the RETAIN Statement

Obs	name	account	balance	dept
1	Angela	12346	1211	ACC
2	Angela	12346	43121	ACC
3	Debbie	12347	3231	FIN
4	Debbie	12347	31211	FIN
5	Josh	12348	12412	MGT
6	Karen	12345	21243	ACC
7	Karen	12345	54021	ACC

Figure 6. Ordering Variables Using the RETAIN Statement

Ordering data in a specific way can be important in certain types of reports. There are other ways that this can be accomplished including a PROC SQL but the RETAIN statement is just as useful for the task.

CONCLUSION

The RETAIN statement can be used in clever ways to manipulate information within the DATA step. It can be an extremely helpful tool to manipulate data that is vertically oriented but still utilize DATA step processing. Using the techniques reviewed in this paper will help to uncover solutions the user will need to work with data.

REFERENCES

Zhao, Yi. 2009. "Effective use of the RETAIN Statement in SAS® Programming." *Proceedings of the Stats Section*, D14. Cleveland, OH: Midwest SAS® Users Group.

Fine, Lisa. 2011. "Ready, Set, Retain and Then Maybe Reset." *Proceedings of the Coder's Corner Section*, 091-2011. Las Vegas, NV: SAS® Global Forum

Chakravarthy, Venky. 2005. "RETAIN or NOT? Is LAG Far Behind?" *Proceedings of the Coder's Corner Section*, CC19. Phoenix,AZ: PharmaSUG.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Matt Taylor
One Bank of America Center
150 N College St.
Charlotte NC 28255
Matthew.k.taylor@bankofamerica.com