Paper 2417- 2018

If You Need These OBS and These VARS, Then Drop IF, and Keep WHERE

Jay Iyengar, Data Systems Consultants LLC

ABSTRACT

Reading data effectively in the DATA step requires knowing the implications of various methods, and DATA step mechanics; The Observation Loop, and the PDV. The impact is especially pronounced when working with large data sets. Individual techniques for subsetting data have varying levels of efficiency and implications for input/output time. Use of the WHERE statement/option to subset observations consumes less resources than the subsetting IF statement. Also, use of DROP and KEEP to select variables to include/exclude can be efficient depending on how they're used.

INTRODUCTION

The DATA step is the primary construct and tool for data manipulation in the BASE SAS® package. As such, it provides a litany of capabilities for processing data. Subsetting, conditional processing, merging, by-group processing, appending, summarizing, and deriving variables are all processes that the DATA step can execute. It's also a versatile tool which provides multiple methods to perform the same tasks and accomplish the same result. The basic structure of the DATA step reads an input data set, and writes an output data set. This paper primarily focuses on methods to select observations and variables, and the efficiency of different techniques using the basic form of the DATA step.

THE DATA STEP LOOP

The DATA step has a built-in automatic loop known as "The Observation Loop". The DATA step reads observations 'Sequentially', meaning in order, one at a time. When the loop executes, the first observation is read from the input data set, each statement within the DATA step executes, and changes are made to the observation. At the end of the loop, when SAS encounters the RUN statement, the observation is written to an output data set at the top of the step. Consequently, the pointer moves down to read the next observation, and a new loop begins. The second observation is read from the input data set. SAS statements within the DATA step are executed, and the process goes on and on. This process continues until all observations have been read from the input data set.

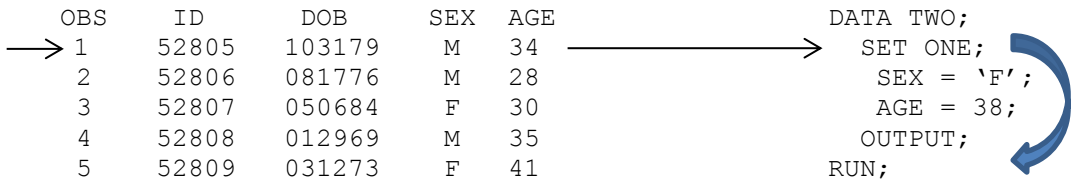
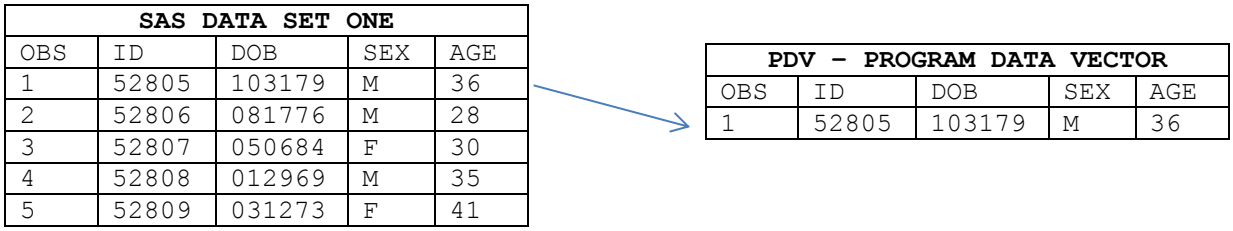


Figure 1. SAS data set ONE

As shown in Figure 1, the input data set (ONE) is specified in the SET statement, while the output data set (TWO) is specified in the DATA statement.

THE PDV

The PDV, which stands for Program Data Vector, is a key DATA step concept. The PDV is an internal record in memory where data is held while it's being processed through the DATA step. When the DATA step is compiled, the PDV is created containing all variables on the input SAS data set. Any new variables created during the step are initialized to missing. As Figure 2 shows, when the DATA step executes, SAS reads the first observation from the input data set, and sends it to the PDV. The record is held in the PDV during the DATA step loop.

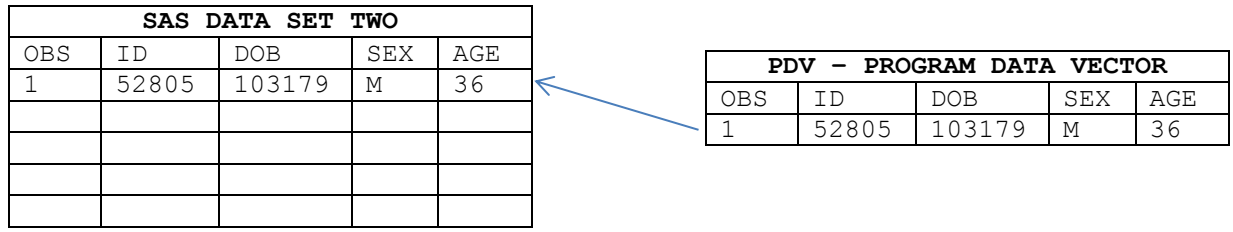


SAS DATA SET ONE				
OBS	ID	DOB	SEX	AGE
1	52805	103179	M	36
2	52806	081776	M	28
3	52807	050684	F	30
4	52808	012969	M	35
5	52809	031273	F	41

PDV – PROGRAM DATA VECTOR				
OBS	ID	DOB	SEX	AGE
1	52805	103179	M	36

Figure 2. SAS data set ONE and the PDV

During DATA step execution, the statements within the DATA step are executed, and changes are applied to the observation held in the PDV. If new variables are assigned, or values of current variables are modified, the changes to variables appear in the PDV. The DATA step has an implicit output at the bottom of the observation loop. As Figure 3 shows, when SAS reaches the end of the DATA step, the observation held in the PDV is released and written to the output data set.



SAS DATA SET TWO				
OBS	ID	DOB	SEX	AGE
1	52805	103179	M	36

PDV – PROGRAM DATA VECTOR				
OBS	ID	DOB	SEX	AGE
1	52805	103179	M	36

Figure 3. SAS data set TWO and the PDV

It's worthwhile to note that SAS data sets have an end-of-file indicator flag. After all observations have been read, SAS encounters the end-of-file marker, and the DATA step stops executing.

SUBSETTING DATA

In the DATA step there are multiple methods available to subset observations. There's the IF statement, and the WHERE statement/option. Both statements specify a condition for a variable. If the condition evaluates to true, then processing continues, and the observation is either read from or written to a SAS data set. If the condition is false, the record is either deleted or excluded, and the pointer moves to read the next observation.

IF STATEMENT

With the IF statement, you control which observations are written to the output data set. The IF statement is also called the subsetting IF. It should be noted that the IF statement is separate from the IF-THEN-ELSE series of statements, which are used for executing statements conditionally.

Both single and multiple conditions can be specified in an IF statement. As Figure 4 below shows, multiple conditions are connected by the AND or OR operators.

```
DATA NDF1;
   SET NDF;
   IF GENDER='F' AND STATE='AZ';
RUN;
```

Figure 4. DATA step with IF statement.

WHERE STATEMENT

For most of my SAS career, I only used the subsetting IF statement to subset in a DATA step. I didn't realize the WHERE statement could be used to subset in a DATA step. I also didn't know it was an effective tool for subsetting. With WHERE you control which observations are read from a SAS data set. The WHERE statement has essentially the same syntax as the IF statement except for the keyword. Just as with IF, both single and multiple conditions can be specified using WHERE. WHERE can be used in SAS Procs, in addition to DATA steps. Figure 5 shows a DATA STEP with a WHERE statement.

```
DATA NDF2;
   SET NDF;
   WHERE GENDER='F' AND STATE='AZ';
RUN;
```

Figure 5. DATA step with WHERE statement.

WHERE can be coded as a DATA step statement, and also as a data set option. As Figure 6 shows, the data set option can be either an input data set option (coded on the SET statement), or an output data set option (on the DATA statement). The WHERE input data set option performs the same action as the WHERE statement.

```
DATA NDF1;
 SET NDF(Where=(State='AZ'));
RUN;
```

```
DATA NDF2(Where=(State='AZ'));
 SET NDF;
RUN;
```

Figure 6. DATA steps with WHERE input and output data set options.

STEP AND FILE DIFFERENCES

There are different places and situations within your SAS code where WHERE and IF can be used. To review, WHERE can be used both in a DATA step, and SAS Procs, while IF can only be used in a DATA step. Both IF and WHERE can be used to subset data based on a SAS data set. Some programming tasks involve reading data in a text, ASCII, or flat file format, and converting it to a SAS data set. Suppose you're reading an external file using INFILE and INPUT, and then want to subset your data. In this case, you need to use IF or the WHERE output data set option. Figure 7 shows DATA step code for reading an external file, and sub setting with IF in the same step.

```
DATA NEW;
    INFILE FILEREF MISSOVER;
    INPUT ID GENDER $ AGE DOB STATE;
    IF GENDER='F' AND STATE='AZ';
RUN;
```

Figure 7. DATA step with INFILE, INPUT and IF

The variable specified in the WHERE subsetting condition is required to be a SAS data set variable. The DATA step also has temporary variables available to it, and you might need to reference these in your subsetting condition. If you're performing a DATA step merge, you'll probably need to use the IN= temporary variable to output match results. This requires use of subsetting IF. Figure 8 shows a DATA step merge with the IN= variable.

```
DATA THREE;
   MERGE ONE (IN=A)  TWO (IN=B);
    BY ID;
   IF A AND B;
RUN;
```

Figure 8. DATA step merge with IF.

Figure 9 lists the specific places in a SAS program IF and WHERE can be used and files they can be used with.

	IF	WHERE
DATA Steps	X	X
PROC Steps		X
SAS data sets	X	X
External Files (Text, Raw data)	X	
Automatic/Temporary Variables	X	
Computed/Derived Variables	X	

Figure 9. IF vs. WHERE Comparison

IF VS. WHERE PROCESSING

The key difference between IF and WHERE is in DATA step mechanics, and the implications of each technique for efficient processing of data. This involves understanding the PDV. The WHERE statement or input data set option applies its condition before a record is read and loaded to the PDV. Only those records which meet the WHERE condition are read from the input data set. The IF statement applies its condition after a record is read and loaded to the PDV, but before its written to the output data set. All records in the input data set are read using IF instead. Only those records meeting the IF condition are written to the output data set. Figure 10 shows this distinction between WHERE and IF during the flow of a DATA step.

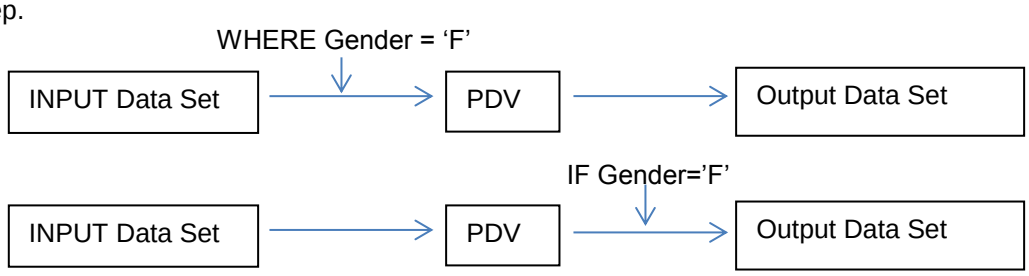


Figure 10. WHERE and IF along the Observation Loop.

Below are excerpts from a SAS log which show the processing difference between IF and WHERE. The National Downloadable File is a database of physicians containing 1048575 observations with one record per physician. It contains 22518 physicians practicing in Maryland.

```
56          DATA NDF_MD1;
57            SET NEWFILE.NDF;
58            IF STATE = 'MD';
59          RUN;

NOTE: There were 1048575 observations read from the data set NEWFILE.NDF.
NOTE: The data set WORK.NDF_MD1 has 22518 observations and 36 variables.

61          DATA NDF_MD2;
62            SET NEWFILE.NDF;
63            WHERE STATE = 'MD';
64          RUN;

NOTE: There were 22518 observations read from the data set NEWFILE.NDF
WHERE STATE='MD';
NOTE: The data set WORK.NDF_MD2 has 22518 observations and 36 variables.
```

Figure 11. Examples with subsetting IF and WHERE

As Figure 11 shows, using IF to produce a subset of Maryland physicians, the entire 1048575 records were read from the input data set (NDF). The new data set (NDF_MD1) contains the subset of 22518 records. Using WHERE to produce the same subset, 22518 records for Maryland physicians only were read from the input data set. The new data set (NDF_MD2) contains the same record set as NDF_MD1.

All else being equal, WHERE is more efficient because smaller amounts of data are processed. Both the WHERE statement and WHERE input data set option have the same effect, and the same efficiency implications. The subsetting IF has the same effect as the WHERE output data set option, and the same implication for efficiency.

In today's computing world of big data, programmers are processing data sets containing hundreds of millions to billions of records. In my experience in the Healthcare field, this is especially the case, because of the need to process large healthcare claims data sets. Reading an entire data set which contains a hundred million records or more requires SAS to perform millions of read operations. Processing only necessary records saves valuable input/output or I/O time.

Suppose you need to use IF to subset, because you're reading a text file, or running a DATA step merge. It's best to position it as close to the top of the step as possible. Then IF will be executed before any remaining code in the step. The remaining code will only be executed if the IF condition is met. Positioning IF at the top of the step saves CPU or processing time. Figure 12 shows a DATA step with optimal placement of the IF statement.

```
DATA MedU16;
    SET MedUtiliz16;
    IF CITY='Bethesda';

    IF PROVIDER_CITY='Baltimore' THEN PROVIDER_STATE='MD';
    ELSE IF PROVIDER_CITY='Arlington' THEN PROVIDER_STATE='VA';

    IF BILLTYPE = 13 THEN BILLTYPEDSC = 'Hospital Outpatient';
    ELSE IF BILLTYPE = 11 THEN BILLTYPEDSC = 'Hospital Inpatient';
    ELSE IF BILLTYPE = 33 THEN BILLTYPEDSC = 'Home Health Agency';
RUN;
```

Figure 12. DATA step with subsetting IF.

RECOMMENDATION

I suggest using WHERE as a statement or input data set option to subset when reading large data sets of a million or more records. Using these forms of WHERE will result in reduced I/O time. I also suggest using IF to subset in a DATA step merge, and when reading external files. IF is better than the WHERE output data set option. You can't use temporary variables with WHERE, and you can position IF to avoid extra processing time. However, the results depend on the system at your site, the nature of your data, and available resources such as memory. It's a good idea to run some comparative tests between IF and WHERE, and do some benchmarking to find the most efficient method.

KEEP AND DROP

Experienced SAS users know that SAS features more than one technique to accomplish a given task. The DATA step also features more than one method to select variables. To subset variables in a DATA step, you use the KEEP or DROP option/statement. With KEEP and DROP you specify the variables to be included or excluded for processing. In a DATA step, KEEP and DROP can be specified either as a statement or a data set option in parentheses; the same as WHERE.

As Figure 13 shows, as an option, KEEP/DROP can be coded either as an input or output data set option. The KEEP/DROP input data set option controls which variables are read or not read from an input SAS data set. The input data set option is coded on the SET statement. The KEEP/DROP output data set option controls which variables are written or not written to an output data set. The output data set option is coded on the DATA statement.

```
DATA TWO;                        DATA TWO (DROP = SEX DOB);
 SET ONE (KEEP=ID GENDER);        SET ONE;
RUN;                             RUN;
```

Figure 13. Keep/Drop data set option(s)

The KEEP/DROP statement controls which variables are written or not written to an output data set. The KEEP/DROP statement performs the same action as the KEEP/DROP output data set option. Figure 14 shows DATA steps with the KEEP/DROP statements.

```
DATA TWO;                  DATA TWO;
  SET ONE;                   SET ONE;
  KEEP ID GENDER;            DROP SEX DOB;
RUN;                       RUN;
```

Figure 14. KEEP/DROP statements

STEP AND FILE-SPECIFIC USE

The specific parts of your SAS program where KEEP/DROP can be used depend on the form of use. As we've seen, KEEP/DROP can be used in a DATA step, either as an input data set option, output data set option, or SAS statement. But KEEP/DROP can be used in SAS Procs only as a data set option. The KEEP/DROP statement can only be used within a DATA step. Figure 15 shows using KEEP/DROP in PROC SORT as a data set option.

```
PROC SORT DATA=ONE OUT=TWO(KEEP=ID GENDER) NODUPKEY;
    BY ID;
RUN;
```

Figure 15. PROC SORT with KEEP/DROP option.

Both the KEEP/DROP statement, and the KEEP/DROP data set option can list variables to select or exclude from a SAS data set. However, some programming tasks require coding a DATA step to read an external file using INFILE and INPUT. Only the KEEP/DROP statement or output data set option can be used in a DATA step that reads an external file, such as a text or flat file. The concept here is the appropriately named data set option can only refer to a SAS data set. Figure 16 shows a DATA step with INFILE, INPUT, and the KEEP statement.

```
DATA TEMP1;
    INFILE FILEREF MISSOVER;
    INPUT ID GENDER $ AGE DOB CITY $ STATE $;

    KEEP ID CITY STATE;

    IF CITY='Bethesda' AND STATE='MD';
RUN;
```

Figure 16. DATA step reading external file.

Granted, if you only want a specific set of variables read from the file, you can always omit them from the INPUT statement. Figure 17 is a table that lists the places or situations in a program where the KEEP/DROP statement/option can be used.

	DROP Option/Statement	**KEEP Option/Statement**
DATA step	X	X
PROC Step	X (option only)	X (option only)
SAS data sets	X	X
External files	X (statement only)	X (statement only)

Figure 17. KEEP/DROP step and file differences

The KEEP/DROP statement and output data set option have the same effect of controlling which variables are written to the output data set. Conceptually, there's no difference in processing efficiency between these two methods. Deciding between these two methods might seem an arbitrary choice. The advantage of using the output data set option comes when you want to output multiple data sets.

Using the KEEP/DROP statement, the same list of variables is applied to each output data set. Using the data set option instead, variable lists custom to each data set can be created. Figure 18 demonstrates this advantage of the KEEP output data set option.

```
DATA HOSPOUT(KEEP=PATID PROVID AGE) HOSPIN(KEEP=PATID PROVID LOS)
     HHA(KEEP=PATID PROVID SEX);

  SET MEDUTIL16;

  IF BILLTYPE = 13 THEN OUTPUT HOSPOUT;
  ELSE IF BILLTYPE = 11 THEN OUTPUT HOSPIN;
  ELSE IF BILLTYPE = 33 THEN OUTPUT HHA;
RUN;
```

Figure 18. DATA step with multiple output data sets.

DATA step merges often involve combining two files which have common variables. If you're combining two files with same-named variables, the variable in the second file will overwrite the variable in the first. This is by default. The variable in the output data set will have the values of the variable in the second file. It will still retain the attributes (length, format) of the variable in the first. Using KEEP or DROP as an input data set option can be handy in preventing variables from overwriting each other.

```
DATA THREE;
   MERGE ONE(IN=A KEEP=ID AGE) TWO(IN=B DROP=AGE);
      BY ID;
   IF A AND B;
RUN;
```

Figure 19. DATA step merge with KEEP/DROP

In the example in Figure 19, AGE exists on both data sets. To prevent AGE in TWO from overwriting AGE in ONE, AGE in ONE was kept, and AGE in TWO was dropped. Data set THREE contains AGE with the values and attributes from data set ONE.

KEEP/DROP AND THE PDV

In deciding which form of KEEP/DROP to use, its key to understand where they're executed during DATA step processing. The KEEP/DROP input data set option is applied before variables are assigned to the PDV. Only the variables that are listed (KEEP) or not listed (DROP) are read from the input data set, using this option. The KEEP/DROP statement or output data set option is applied after variables have been loaded into the PDV, but before they're written to the new data set. All variables are read from the input data set, using either the statement or output data set option. Figure 20 shows the different points of a DATA step execution where the KEEP/DROP options or statements are applied.

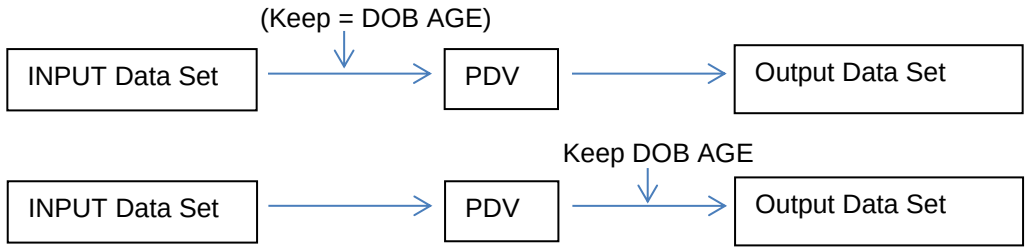


Figure 20. KEEP/DROP and the observation loop

Large data sets may contain hundreds of variables, in addition to millions of records. This is even more the case in today's environment of big data. It makes sense to limit the variables you read to necessary variables which you're going to use. It's preferable to process a tall and narrow data set to processing a tall and wide data set. Processing only necessary variables saves valuable resources, such as input/output time.

RECOMMENDATION

I suggest using the KEEP/DROP input data set option to select necessary variables in a DATA step. It's especially efficient when reading a large data set with a hundred or more variables. I suggest using KEEP/DROP as a DATA step statement or output data set option when reading external files. For DATA step merges, I recommend using KEEP/DROP as an input data set option. This can prevent common variables from overwriting each other. Using these methods will result in decreased input/output (I/O) or processing time, all else being equal. It's also worthwhile to run comparative tests to find the best method, because results depend on your system and data.

CONCLUSION

In a DATA step, I recommend the WHERE input data set option to subset records from a SAS data set. Also, use of the subsetting IF rather than the WHERE output data set option when reading external files or executing DATA step merges. This will result in decreased processing time and improved efficiency, especially when dealing with large data files. For variables, I recommend using KEEP/DROP as an input data set option to select only necessary variables from a SAS data set. Also in a DATA step merge to prevent overwriting. I further recommend doing comparative testing with these methods because results are environment and data dependent.

REFERENCES

Philip, Steven, (2006) “Programming with the KEEP, RENAME, AND DROP data set options”, in Sugi 31 Conference Proceedings, Cary, NC: SAS Institute.

“SAS® Fundamentals: A Programming Approach Course Notes”. SAS Institute Inc. Cary, NC, 1996

ACKNOWLEGEMENTS

The author would like to thank Goutam Chakraborty, SAS Global Forum 2018 Conference Chair, Nancy Moser, Presenter Liason, Tricia Aanderud, Presenter Room Coordinator, and the SAS Global Forum 2018 Executive Committee and Conference Team for accepting my abstract and paper, and organizing a great conference.

Jay Iyengar is Principal of Data Systems Consultants LLC. He is a SAS Consultant, Trainer, and SAS Certified Advanced Programmer. He is co-leader of the Chicago SAS Users Group, WCSUG. He’s presented papers at SAS Global Forum (SGF), Midwest SAS Users Group (MWSUG), Wisconsin Illinois SAS Users Group (WIILSU), Northeast SAS Users Group (NESUG), and Southeast SAS Users Group (SESUG) conferences. He has been using SAS since 1997. His industry experience includes Healthcare, Pharmaceutical, Public Health, Direct Marketing and Educational Testing.

CONTACT INFORMATION

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

Jay Iyengar
Data Systems Consultants LLC
Oak Brook, IL 60523
Email: datasyscon@gmail.com
Linkedin: https://www.linkedin.com/in/jisasprogconsult

TRADEMARK CITATION