(4) SELECTING AND RESTRICTING OBSERVATIONS

Up to now, all the examples of creating SAS data sets from raw data have converted all of the available records in the raw data into observations a SAS data set. In many of the few examples that showed the use of procedures, each procedure processed all the observations in a data set. There may be occasions where you want to select specific records from raw data to be placed into a data set. You may also want to restrict the observations that are used by a procedure. A few of the examples showing procedures thus far use a WHERE statement to select observations from a data set rather than using all the observations. The WHERE statement is just one of many ways to select and/or restrict observations.

...INFILE OPTIONS (FIRSTOBS & OBS)...

The FIRSTOBS and/or OBS options can be used to restrict the records in a raw data file that are processed by an INPUT statement in a data step. Within a data step, they are used as options on the INFILE statement.

...Example 4.1...

data cancer99;
  infile 'k:\epi514\data\cancer99.txt' firstobs=101 obs=200;
  input @01 county $2.
      @03 gender $1.
      @04 age     3.
      @07 cause  $4.
      @11 place  $1.
  ;
run;

The FIRSTOBS and OBS options in the INFILE statement tell SAS to only read records 101 through 200 from the raw data file. While FIRSTOBS is clear, the first record to be processed is number 101, OBS really means the last record to be processed by the data step, not the number of records. When you use the OBS option, you should think of it as the (last)OBS option, not the (number of)OBS option. Also, when reading raw data, the FIRSTOBS and OBS options in the INFILE statement are not really referring to observations. They are referring to records in the raw data that have yet to be converted to observations in a data set. By default, FIRSTOBS has a value of 1, while OBS has a value of MAX (the last record in the raw data file). The INPUT statement uses formatted input to read the values of five variables, but only from 100 records in the raw data.

You need not use both options. If you just wanted to skip the first one hundred records, you could use only FIRSTOBS ...

infile mydata firstobs=101;

If you just wanted to read the first one hundred record observations, you could use only OBS ...

infile mydata obs=100;

Remember that OBS refers to an actual record number, not to the number of observations. If you write the following INFILE statement with the intent of starting at record number 101 and reading 100 records, you will not read any records ...

infile mydata firstobs=101 obs=100;
DATA SET OPTIONS (FIRSTOBS & OBS)...

If your data are already in SAS data set, you can use FIRSTOBS and/or OBS as data set options rather than as INFILE options.

Example 4.2...

data cancer99;
  infile "k:\epi514\data\cancer99.txt";
  input @01 county $2.
@03 gender $1.
@04 age 3.
@07 cause $4.
@11 place $1.
; run;

data cancer99_100;
  set cancer99 (firstobs=101 obs=200);
run;

There are no restrictions placed on the number of records that are read from the raw data file. Data set CANCER99 will contain one observation for each record in the file CANCER99.TXT. The second data step uses FIRSTOBS and OBS data set options to select 100 observations from data set CANCER99. Notice that the options are now placed in parentheses and they are placed directly after the name of the data set in the SET statement. If all you wanted to do is look at a portion of a large data set, you can also use the data set options in a procedure.

Example 4.3...

data cancer99;
  infile "k:\epi514\data\cancer99.txt";
  input @01 county $2.
@03 gender $1.
@04 age 3.
@07 cause $4.
@11 place $1.
; run;

title '100 OBSERVATIONS FROM DATA SET CANCER99';
proc print data=cancer99 (firstobs=101 obs=200);
run;

You can use any combination of FIRSTOBS and LASTOBS as data set options, just as you could when they are used as INFILE options. The only difference is that they must be placed inside parentheses when used as data set options. No parentheses are used when they are used as INFILE options.
Rather than select or restrict observations based on either the record or observation number, you can also select or restrict records and observations based on the value of one or more variables. For example, you have a raw data file with information on cancer deaths. You want to create a data set containing only observations in which the age at death was less than 5.

Example 4.4...

* SUBSETTING IF;
   data cancer1_4;
   infile "k:\epi514\data\cancer99.txt";
   input
     @01 county $2.
     @03 gender $1.
     @04 age    3.
     @07 cause $4.
     @11 place $1.
   ;
   if age lt 5;
   run;
   title 'CANCER DEATHS ... AGE < 5';
   proc print data=cancer1_4;
   run;

This statement is known as a SUBSETTING IF statement. If the statement is true (the value of the numeric variable age is less than 5), SAS continues to process subsequent statements. If the statement is false, SAS immediately returns to the top of the data step. The 'lt' in the subsetting IF statement is a SAS operator meaning 'less than' and is equivalent to using '<' (explained in appendix A).

What happens when SAS immediately returns to the top of the data step, when the SUBSETTING IF statement is false? One consequence is that the bottom of the data step is never reached. If the bottom of the data step is not reached, no observation added to the data set. This is the first time that you have seen an explicit mention of when an observation is added to a SAS data set. That occurs when the end of the data step is reached. Any SAS statement that prevents the end of the data step being reached prevents an observation from being added to a data set.

A portion of data set CANCER1_4 is displayed on the right. Look at the values of the variable AGE. In addition to all records with an age less than 5, you also selected records with a missing value for age. SAS considers missing numeric data to have a value less than the lowest negative number (sort of a minus infinity). If you ask for observations with an age less than 5, a missing value for age is less than 5. If you want only observations with an age in the range 0 to 4, you would use a different SUBSETTING IF statement...

... if age ge 0 and age lt 5;

Notice that you cannot use ...

... if age ge 0 and lt 5;

You must repeat the name of the variable AGE.
A SUBSETTING IF statement is a very easy way to select observations based on the value of a variable and you can use more than one variable.

...Example 4.5...
* SUBSETTING IF;
data cancer1_4m;
infile "k:\epi514\data\cancer99.txt";
input @01 county $2. @03 gender $1. @04 age 3. @07 cause $4. @11 place $1.;
if age ge 0 and age lt 5 and gender eq '1';
drop gender;
run;
title 'CANCER DEATHS ... MALES, AGE < 5';
proc print data=cancer1_4m;
run;

Two variables are used to select observations, AGE and GENDER. Notice that a range of ages is used to eliminate observations with a missing value for AGE. In the raw data, males are coded as 1, not M. Since GENDER is a character variable, the value of that variable is placed in quotes in the SUBSTETTING IF statement. There is no limit on the number of variables you can use in such a statement. The only thing that matters is if the statement is true or false. There is a new statement in the data step, DROP. Since the newly created data set contains only males, there is no reason for it to contain a variable with information on gender. There is more information about DROP statements in the next chapter. For now, all you need know is that a DROP statement tells SAS what variables you want to leave out of a data set. The values of variables for all observations in that data are shown on the right in output from PROC PRINT.

When the selection of observations relies on checking multiple values of a variable, you can reduce the amount of writing of SAS by using the IN operator. The county numbers for counties in the capital district around Albany are as follows: 01, Albany; 41, Rensselaer; 45, Saratoga; 46, Schenectady. If you wanted to create a data set with only observations from those counties, a SUBSETTING IF statement could be ...

if county eq '01' or county eq '41' or county eq '45' or county eq '46';

However, you can also write ...

if county in ('01', '41', '45', '46'); or if county in ('01', '41', '45', '46');

eliminating the need to repeat the name of the variable four times. In example 4.5, COUNTY is a character variable; its value is read with the informat $2. If COUNTY is a numeric variable, the SUBSETTING IF statement is written as ...

if county in (1 41 45 46);
...Example 4.6...
* IN operator;
data cancer_local;
infile "k:\epi514\data\cancer99.txt";
input
@01 county $2.
@03 gender $1.
@04 age $3.
@07 cause $4.
@11 place $1.;
if county in ('01' '41' '45' '46') and age ge 0 and age lt 25;
run;

run;
title 'CANCER DEATHS ... CAPITAL DISTRICT, AGE < 25';
proc print data=cancer_local;
run;

The above example creates a data set with observations from four counties with age less than 25. Only seven observations met the criteria in the SUBSETTING IF statement and they are shown on the right.

Thus far, observations have been selected using the variables AGE, GENDER, and COUNTY. What if you wanted to create a data set that contained only observations for deaths due to lung cancer? To do that, you would have to know all the 4-digit ICD-10 codes that are used for lung cancer. They are ...

C34  Malignant neoplasm of bronchus and lung
C340  Main bronchus
C341  Upper lobe, bronchus or lung
C342  Middle lobe, bronchus or lung
C343  Lower lobe, bronchus or lung
C348  Overlapping lesion of bronchus and lung
C349  Bronchus or lung, unspecified

No death record can contain only the 3-digit code C34. There must be a 4th digit if a death is due to lung cancer. Given what you have recently learned about the IN operator, you could write ...

if cause in ("C340' 'C341' 'C342' 'C343' 'C348' 'C349");

However, if you look at the list of cause, they all have one thing in common in that they all start with the C34. There is a shortcut that allows you to select observations based on the starting rather than complete value of a character variable.

...Example 4.7...
* COLON modifier;
data cancer_lung;
infile "k:\epi514\data\cancer99.txt";
run;

run;
title 'LUNG CANCER DEATHS BY CAUSE';
proc freq data=cancer_lung;
   table cause;
run;

The SUBSETTING IF statement select all lung cancer deaths. The variable CAUSE is a character variable with a length of 4 (read with an informat $4$). You want to select observations when the value of CAUSE starts with the characters C34. Placing a colon (:) between the operator 'eq' and the value 'C34' tells SAS to limit the comparison the length of the character string on the right ("C34"), or just look at the first three characters. PROC FREQ (no, it has not been in the notes prior to this) is used to count the number of deaths from each type of lung cancer shown in the table on the previous page. The counts are shown on the right. Notice that 99% of the lung cancer deaths are C349, or 'bronchus or lung, unspecified'. Unfortunately, that is true for most causes of death. Do you have any thoughts as to why that might happen?

What would occur if the SUBSETTING IF statement is written without the colon ...

if cause eq 'C34';

The LOG is shown on the right. No observations are selected. Without the colon in the SUBSETTING IF statement, you are really asking ...

if cause eq 'C34 ';

All causes that start with C34 are followed by another digit, not a blank. So, no observations are selected.

In a SUBSETTING IF there is an implied result based on the statement being either true or false. If the statement is false, the implied result is that SAS returns to the top of the data step. If the statement is true, the implied action is that SAS proceeds to the next statement in the data step. In an IF-THEN statement, the result is explicitly stated rather than being implied. Once again, the action is based on whether the IF portion of the statement is true or false. If it is true, the result is based upon what follows the THEN portion of the statement.

...Example 4.8...
* IF-THEN (DELETE);*
   data cancer_24;
   infile "k:\epi514\data\cancer99.txt";
   input
     @01 county $2.
     @03 gender $1.
     @04 age     3.
     @07 cause  $4.
     @11 place  $1.
     ;
   if age gt 24 then delete;
run;
Instead of a SUBSETTING IF statement, an IF-THEN statement is used to select only those observations that have an age less than 25. If an observation is found in which the age is greater than 24, the instruction to SAS is to DELETE that observation. What DELETE really means is go back to the top of the data step right now. So, this statement is the opposite of the SUBSETTING IF used earlier but has the same result. Only observations with an age less than 25 are put in the data set.

There is another way to write an IF-THEN statement when selecting observations. Instead of DELETE, you can substitute the command OUTPUT.

...Example 4.9...
* IF-THEN (OUTPUT);
data cancer_24;
  infile "k:\epi514\data\cancer99.txt";
  input @01 county $2.
  @03 gender $1.
  @04 age 3.
  @07 cause $4.
  @11 place $1.
  ;
  if age lt 25 then output; ①
run;

The command OUTPUT is used in an IF-THEN statement. If an observation is found in which the age is less than 25, the instruction to SAS is to write an observation to a data set. This statement is different from both the SUBSETTING IF and the IF-THEN statement that included DELETE. If the condition is true, an observation is written to a data set and the data step proceeds to the next statement. If the condition is false, SAS just proceeds to the next statement.

In addition to writing an observation to a data set, the OUTPUT command has another effect on the data step. Earlier in this chapter, you learned that observations are added to a SAS data set at the bottom of the data step, just before SAS returns to the top of the data step to read another record of raw data or another observation from a data set. Think of the bottom of the data step as having an implied OUTPUT statement. If you add an explicit OUTPUT command any place in a data step, you eliminate the implied OUTPUT statement at the bottom of the data step. If there is an OUTPUT command any place within a data step, observations are written to a data set only when that command occurs.

The following table summarizes what happens with the various IF statement seen thus far.

<table>
<thead>
<tr>
<th>STATEMENT</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBSETTING IF</td>
<td>TRUE: CONTINUE TO NEXT STATEMENT</td>
</tr>
<tr>
<td></td>
<td>FALSE: GO BACK TO THE BEGINNING OF THE DATA STEP</td>
</tr>
<tr>
<td>IF-THEN (DELETE)</td>
<td>TRUE: GO BACK TO THE BEGINNING OF THE DATA STEP</td>
</tr>
<tr>
<td></td>
<td>FALSE: CONTINUE TO THE NEXT STATEMENT</td>
</tr>
<tr>
<td>IF-THEN (OUTPUT)</td>
<td>TRUE: WRITE AN OBSERVATION TO A DATA SET, THEN CONTINUE TO THE NEXT STATEMENT</td>
</tr>
<tr>
<td></td>
<td>FALSE: CONTINUE TO THE NEXT STATEMENT</td>
</tr>
</tbody>
</table>

With the first two statements in the above table, going back to the top of the data step means that an observation is not added to the data set being constructed in the data step. In four of the boxes, the action is to continue to the next statement. In one of those four, IF-THEN (OUTPUT), an observation is written to a data set prior to continuing. There is no general rule for what type of statement to use. Using any of the above depends on what you want to do with the SAS code you place in a data step.
There is yet another type of IF statement, IF-THEN-ELSE, in which you can specify what to do if a specific condition is either true or false. What is you wanted to create two data sets, one with only observations for males and another with observations only for females.

...Example 4.10...
* IF-THEN-ELSE (OUTPUT);  
data males females;  
infile "k:\epi514\data\cancer99.txt";  
input @01 county $2.  
@03 gender $1.  
@04 age 3.  
@07 cause $4.  
@11 place $1.  
;  
if gender eq '1' then output males;  
else                  output females;  
drop gender;  
run;

The keyword DATA is followed by the names of two data sets. There is no limit to the number of data sets you can create with one data step (though you might get tired of typing data set names at some point). An IF-THEN-ELSE statement with OUTPUT commands is used to direct observations to the appropriate data set. When an observation is found in which the character variable GENDER had a value of ‘1’, it is written to the data set MALES. Any other value of GENDER puts an observation in the data set FEMALES. Even though there are two semi-colons within the IF-THEN-ELSE statements, you should think of it as one continuous statement that has two (or more) parts. The first part in this example is IF-THEN and the second part is ELSE. If the condition in the first part is true, an observation is written to the data set MALES and the second part of the statement is skipped. That is important to remember. An IF-THEN-ELSE statements continues until a part is found that is true or if an ELSE part is found. As soon as that part is found, the remainder of the statement is skipped and the data step proceeds to next statement. If no part is found that is true, no action is performed. In this example, if the first part of the IF-THEN-ELSE statement is false, the ELSE part is encountered and an observation is written to the data set FEMALES. The DROP statement is used since there is no need for the variable GENDER when the data sets are gender-specific.

There is a difference in how the OUTPUT command is used in the last two examples. In example 4.10, the each command OUTPUT is followed by the name of a data set. No data set name is used after the OUTPUT command in example 4.9. When no data set name is specified, an observation is written to all data sets named in the DATA statement at the start of the data step. In example 4.9, only one data set name is used so there is no need to specify a data set after OUTPUT. In example 4.10, two data set names are used and you must be explicit, telling SAS where to place an observation. To test your understanding of how the OUTPUT statement works, what happens if the data set name FEMALES is left out of the IF-THEN-ELSE statement in example 4.10? You could also try to sketch out a table like the one on the previous page or draw a diagram that explains how an IF-THEN-ELSE works.

There is one last command to describe that can be used in place of DELETE and OUTPUT. That command is RETURN. One feature of the RETURN command that makes it a bit more difficult to understand is that it can result in either of two different actions taking place within a data step. The next two examples describe the different actions based on the following ... imagine you receive data from several different clinics. You know that the blood pressure values from two of them are consistently low since those clinics use old equipment. All values from those clinics must be multiplied by 1.1 (they are always 10% too low).
...Example 4.11...
* IF-THEN (RETURN);
data clinics;
  input
    @01 clinic $1.
    @02 sbp 3.
    @05 dbp 3.
  ;
  if clinic eq '2' then return;
  sbp = 1.1 * sbp;
  dbp = 1.1 * dbp;
  label
    sbp = 'SYSTOLIC BLOOD PRESSURE'
    dbp = 'DIASTOLIC BLOOD PRESSURE'
  ;
datalines;
1080120
2075110
3080150
110200
;
run;

The data from clinic 2 is correct. The IF-THEN statement first checks to see if an observation is from clinic 2. If the observation is from clinic 2, the RETURN command executes. When used in this manner, RETURN results in two actions. First, an observation is written to the data set CLINICS. Second, SAS then goes back to the top of the data step to read another record. Since the observation is written to the data set prior to the statements that multiply both SBP and DBP by 1.1, those values are not changed. If the observation is not from clinic 2, the RETURN command does not execute and SAS moves to the next statement in the data step. The blood pressures are corrected and an observation is added to data set CLINICS when the bottom of the data step is reached. Remember the implied PRINT output at the bottom of the data step. PROC PRINT output on the right shows that only the blood pressures not from clinic 2 have been changed.

<table>
<thead>
<tr>
<th>DATA SET CLINICS (CLINIC #2 NOT CORRECTED)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obs</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

The next example uses the same data and it demonstrates the other action of the RETURN command. What if the task was changed so what you wanted in the final data set was to create two new data sets, one with data from clinic 1, the other with data from clinic 3. You do not want any records from clinic 2.

...Example 4.12...
* IF-THEN (RETURN+OUTPUT);
data clinic1 clinic3;
  input
    @01 clinic $1.
    @02 sbp 3.
    @05 dbp 3.
  ;
  if clinic eq '2' then return;
  sbp = 1.1 * sbp;
  dbp = 1.1 * dbp;
  if clinic eq '1' then output clinic1;
  else                  output clinic3'
drop clinic;

Introduction to SAS®
Mike Zdeb (send comments, corrections to: msz03@albany.edu)

label
sbp = 'SYSTOLIC BLOOD PRESSURE'
dbp = 'DIASTOLIC BLOOD PRESSURE'
;
datalines;
1080120
2075110
3080150
1110200
;
run;

Two data sets are created. The IF-THEN statement with the RETURN command is still in the data step. However, there is now an OUTPUT statement at the bottom of the data step. The effect of the OUTPUT statement is that observations are written to the data set CLINICS only at the point in the data step where the OUTPUT command occurs. The RETURN command now only means go back to the top of the data step. The IF-THEN portion of the statement checks to see if an observation is from clinic 2. If it is, the RETURN command does not write an observation to the data set. It just returns you to the top of the data step. All other observations are corrected and then written to the appropriate data set. The variable CLINIC is dropped since the new data sets are clinic-specific. The LOG on the upper right shows that data step created only two data sets. PROC PRINT output on the lower right shows the corrected blood pressures in the two data sets.

The following table summarizes the action produced by DELETE, OUTPUT, and RETURN. The SAS online help refers to all three of these as statements, not commands. However, as you have seen, they often occur within statements, not as statements in and of themselves. That is why the word command is used in these notes to indicate an instruction that occurs within a statement (confused?).

<table>
<thead>
<tr>
<th>COMMAND (STATEMENT)</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE</td>
<td>GO TO THE BEGINNING OF THE DATA STEP</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>WRITE AN OBSERVATION TO A DATA SET, THEN CONTINUE TO THE NEXT STATEMENT IN THE DATA STEP</td>
</tr>
<tr>
<td>RETURN</td>
<td>IF NO OUTPUT COMMAND (STATEMENT) IN THE DATA STEP... WRITE AN OBSERVATION TO A DATA SET, THEN GO TO THE BEGINNING OF THE DATA STEP</td>
</tr>
<tr>
<td></td>
<td>IF OUTPUT COMMAND (STATEMENT) IN THE DATA STEP... GO TO THE BEGINNING OF THE DATA STEP</td>
</tr>
</tbody>
</table>

One very important thing to remember before moving on to WHERE statements and data set options is that all the various varieties of IF statements (SUBSETTING IF, IF-THEN, IF-THEN-ELSE) can only be used in data steps, not in procedures. The only way to limit observations that are used in a procedure is by using either a WHERE statement or data set option.
...WHERE STATEMENTS AND DATA SET OPTIONS...

If your data are already in a SAS data set, you can tell SAS that you only want to use selected observations using either a WHERE statement or data set option. Both the statement and data set option can be used in either a data step or a procedure. In a data step, you may be using either raw data (reading it with an INPUT statement) or using an already existing data set (reading it with a SET statement). In either case, you can use the various IF statements you have already seen to select or restrict the use of specific observations. However, if you are using an already existing data set, you may also use a WHERE statement or data set option. In a procedure, you cannot use an IF statement and the only way to select or restrict the use of observations based on the value of one or more variables in a data set is with either a WHERE statement or data set option.

Assume that you have a data set name CANCER99. If you use PROC CONTENTS, you see that it contains the variables shown on the right. You want to count the number of males and females who died from cancer in the range 10 to 14. Since the data are already in a data set, you can use a WHERE statement.

...Example 4.13...

```sas
proc freq data=x.cancer99;
table gender;
where age ge 10 and age le 14; 
run;
```

The default behavior of SAS procedures is to use all the observations in data set. The WHERE statement limits the observations used in PROC FREQ to only those with an age in the range 10 to 14. The results are shown on the right.

Example 4.13 uses a WHERE statement. The WHERE data set option will produce the same results.

...Example 4.14...

```sas
proc freq data=x.cancer99 (where=(age ge 10 and age le 14)) ;
table gender;
run;
```

The WHERE data set option looks similar to the WHERE statement. However notice that there are a few differences. First, the data set option is placed directly after the name of the data set used in the procedure. Next, you need two sets of parentheses, one around the entire option and one around the selection criteria. Last, there is an '=' between the WHERE and the selection criteria. In some cases, there are reasons to use a data set option instead of a statement. In this example, the WHERE statement is a more straightforward method (less typing!!!).

If you look in the LOG file after using either of the above, you will see that SAS repeats your observation selection instructions ...

```
NOTE: There were 34 observations read from the data set X.CANCER99.
WHERE (age)=10 and age(<=14);
```

and can also see that SAS rewrote the WHERE statement (or data set option) so it looks different from the one provided in the above examples.
When limiting observations using a range of values, you can make use of another feature of WHERE clauses, the word BETWEEN.

...Example 4.15...
proc freq data=x.cancer99;
  table gender;
  where age between 10 and 14; ①
run;

A WHERE statement using BETWEEN produces the same results as the previous two examples ①. The statement includes observations in the range 10 through 14 (not 11 to 13, another way of interpreting BETWEEN, but not how SAS interprets that word in a WHERE statement or data set option).

In addition to BETWEEN, you can also write another WHERE statement that also looks like a real sentence and select observations with missing values.

...Example 4.16...
proc print data=x.cancer99;
  where age is missing or place is missing; ①
run;

Both a numeric variable (AGE) and a characted variable (PLACE) are used in a WHERE statement ①. If you did not use 'IS MISSING', you might have written the statement as ...

where age eq . or place eq ' ';

and you would have to remember to use a period to represent missing numeric values (for AGE) and a space to represent missing character values (for PLACE). You can also use the word NOT within a WHERE statement.

...Example 4.17...
proc print data=x.cancer99;
  where age is not missing and place is missing; ①
run;

proc print data=x.cancer99;
  where not (age between 5 and 99) and age is not missing; ②
run;

proc print data=x.cancer99;
  where not (place in ('A' 'B' 'C' 'D')); ③
run;

WHERE statements select observations from data sets by evaluating an expression as being either true or false. Normally, if the expression is true, observations are selected. Including the word NOT prior to the expression in a WHERE statement select observations when the expression is false. The three WHERE statements in example 4.17 all contain NOT. The first WHERE uses not missing to select observations with non-missing values of AGE ①. The next WHERE statement selects observations with non-missing values AGE that are outside the range 5 to 99 ②. The last WHERE selects observations with a value of PLACE that is not in the list of values specified ③. Using NOT in a WHERE statement should be dictated by whether it reduces the amount of SAS code you must write while still having the logic if the statement remain clear to you (and perhaps other users of the code).
Thus far, all the examples of using WHERE have been in procedures. You can also use WHERE statements and data set options within a data step.

...Example 4.18...
data ca_albany;
set x.cancer99;
where county eq '01';  
run;

data ca_albany;
set x.cancer99 (where=(county eq '01'));
run;

The WHERE statement and the WHERE data set option both have the same result (a recommendation ... since there is less SAS code to be written, use the WHERE statement for similar tasks). A new data set is created that contains only those observations in which the value of the variable COUNTY is '01'. The data set X.CANCER99 has over 37,000 observations. However, when you look at the LOG after using either of the above data steps, you only see a count of the number of observations that meet the criteria in the WHERE statement or data set option.

| 76  | data ca_albany;             |
| 77  | set x.cancer99;             |
| 78  | where county eq '01';      |
| 79  | run;                       |

NOTE: There were 724 observations read from the data set X.CANCER99. WHERE county='01';
NOTE: The data set WORK.CA_ALBANY has 724 observations and 5 variables.

If a SUBSETTING IF statement had been used in place of the WHERE statement, the LOG looks a bit different.

| 89  | data ca_albany;             |
| 90  | set x.cancer99;             |
| 91  | if county eq '01';          |
| 92  | run;                        |

NOTE: There were 37583 observations read from the data set X.CANCER99. WHERE county='01';
NOTE: The data set WORK.CA_ALBANY has 724 observations and 5 variables.

In either case, a new data set with only 724 observations is created. The WHERE and IF statements operate differently. The WHERE statement looks at an observation prior to moving it from the data set to memory. If the observation meets the criteria in the WHERE statement, the data step reads the entire observation and then writes it to the new data set. All you see in the LOG is the number of observations that met the criteria in the WHERE statement, COUNTY having a value of '01'. When the SUBSETTING IF statement is used, all observations are moved into memory. Then a decision is made whether an observation is written to the new data set. All 37,583 observations are read and the 724 that meet the IF criteria are placed in the new data set. The distinction is a bit subtle. Many SAS users will tell you that in this example, the WHERE statement is more efficient since the data step only reads the observations it needs to find to create the new data set. In reality, if efficiency is defined by how much time it takes to complete the data step, there is no difference between WHERE and IF (the IF may even be faster).
WHERE VERSUS IF...

Some questions you might have at this point are: when should I use WHERE and when should I use IF; is one preferable over the other; do they work exactly the same. Here is a brief (and not exhaustive) list of comparisons between WHERE and IF.

#1 The only way to limit observations used in a procedure is with a WHERE statement or data set option. You cannot use IF in a procedure. There have been numerous examples thus far of using WHERE in a procedure, but you have not seen IF used in a procedure.

conclusion: ADVANTAGE WHERE (only thing you can use in a PROC)

#2 You can use IF or WHERE in a data step. However, if you are using a WHERE statement, there must be a SET statement somewhere in the data step since a WHERE statement can only be used to select observations from an already existing data set. Any variable used in WHERE statement must already exist in a data set and the only way you use an already existing data set in a data step is in a SET statement. So if you have a WHERE statement in a data step, there must be a SET statement in that data step. Variables created in a data step cannot be used in a WHERE statement.

If you are using an IF statement, you might be using an already existing data set or reading raw data. The variable(s) used in an IF statement can be from a data set read with a SET statement, from the list of variables in an INPUT statement, or a variable newly created with the data step.

conclusion: ADVANTAGE IF (more flexible in a data step, can be used in more situations)

#3 As discussed in the previous section, when used in a data step, the WHERE statement can be thought of as peeking at your data prior to actually using an observation. If the result of this initial look meets the criteria expressed in the WHERE statement, an observation is read. When an IF statement is used, the observation is read first, then a decision is made based on the expression in the IF statement. The WHERE statement is thought of as more efficient, but if efficiency is measured by time to complete a data step, there is no difference (and IF might even be faster).

conclusion: DRAW (no difference in time to complete a data step)

#4 The IF statement is 'forgiving' when you forget how to use numeric and character variables in expressions. The WHERE statement is not 'forgiving' and your data step will stop because of an error when you forget how to use numeric and character variables in expressions. Here is an example.

...Example 4.19...
data ca_albany;
set x.cancer99;
   where county eq 1;  ☐
run;

data ca_albany;
set x.cancer99;
   if county eq 1;  ☒
run;

In data set CANCER99, county is a character variable. In the first data step, you are asking SAS to select observations where COUNTY has a numeric value of 1 ☐. In the second data step, you make the same mistake, but this time the mistake is in an IF statement ☒. The statements should look as follows ...

where county eq '01';
if county eq '01';
The LOG that results from the use of each of the data steps in example 4.19 are shown above. Notice the error message that results from not remembering to treat COUNTY as a character variable in the WHERE statement. You are told that the expression in the WHERE statement requires compatible variables. That is not much information since you have no idea as to what compatible means in this context. What it does mean is that if you ask if a character variable has a certain value or values, the value of values must be character constants, values within quotes. So, SAS is telling you that you should know more about your data and how to write WHERE statements, and until you learn, the data step will not work.

There is no error in the LOG when you used the wrong type of constant in an IF statement. The data step actually worked. There is a note in the LOG that tells you "...Character values have been converted to numeric values..." in line 228. That is the line with the IF statement and the note is telling you "...you asked me to compare a character variable to a numeric constant and I do not know how to do that, but for the duration of this IF statement, I will pretend that COUNTY is a numeric variable, convert its character value to a numeric values and check if that numeric value is equal to the numeric constant, which in this statement has a value of 1..." So, SAS is telling you that you should know more about your data and how to write IF statements, but until you learn, you are forgiven.

CONCLUSION: ADVANTAGE WHERE (incorrect SAS code produces error, not just a note)

There is no final score (OK, WHERE/2, IF/1, ONE TIE), and there are many more things to learn about WHERE and IF statements. At this point, if you want to simplify your knowledge base ... use WHERE in procedures, use IF in data steps. That is all you have to remember for now.

What you should remember...
Everything, well almost everything except for the section that covers RETURN. Some of the topics covered in this chapter are more important than others. But there is a very good chance that if you continue to use SAS, you will encounter situations where you will use many if not all of the topics in this chapter. You might even have occasion to use RETURN.