Session 10: Batch Edit and Export

At the end of this session participants will be able to:

- Implement consistency checks in a batch edit application
- Use batch edit to add calculated variables and recoded variables
- Use batch edit to convert checkbox values to yes/no variables for analysis

Creating a Batch Edit Application

A batch edit application is like a data entry application but without the forms. It is meant to be run after data entry to detect and fix problems in the data file. A batch edit application takes an input data file and runs logic on it. It generates both a report, called a listing file, and optionally an output data file which is a modified version of the input file. A batch edit application never changes the input file.

To create a batch, edit application you choose File→New from CSPro and choose Batch Edit Application. You then choose a dictionary. This is usually the same dictionary that you used for data entry.

The user interface for working with batch applications is similar to the one for working with data entry applications except that there are no forms. Instead there is a tab for edits. Just like in data entry, you add logic to PROCS. Instead of running interactively, all the error messages are written out to a log file for review after the whole program has run.

Checking for Errors

To add consistency checks we proceed just as we did in our data entry application by adding logic to the appropriate PROC. Let's start with a simple check that the age of first marriage is not greater than the age.

```
PROC DEMEDC
// Check if currently attending and never attended school
if DEMEDC = 1 and DEMEDL = 1 then
    errmsg("DEMEDC=currently attending, DEMEDL=never attended");
endif;
```

Next, we run the application but first we need a test data file. You can use the file PopstanLFS.csdb. Run the application against this test data. After the application runs, we see the log file. The log file reports that we have a case where this error exists.

```
Process Messages

*** Case [110101101] has 1 messages (0 E / 0 W / 1U)
    U    -9 DEMEDC=currently attending, DEMEDL=never attended

User unnumbered messages:

   Line    Freq  Pct.  Message text                                          Denom
   ----    ----  ----  ------------                                         -----
      9       1    -   DEMEDC=currently attending, DEMEDL=never attended       -

CSPRO Executor Normal End
```

To figure out what the problem is we can open up the problem case in data entry. The printout in the listing file contains the case identifiers which we can use to find the case. You can copy the case id from the listing file and use it with Find Case on the Edit menu in CSEntry.

Getting Summary Information

If you add *summary* to your error message the individual error cases will not be written to the listing file. With *summary,* only the total number of times each error is encountered is written to the file.

```
PROC DEMEDC
// Check if currently attending and never attended school
if DEMEDC = 1 and DEMEDL = 1 then
    errmsg("DEMEDC=currently attending, DEMEDL=never attended") summary;
endif;
```

```
User unnumbered messages:

   Line    Freq  Pct.  Message text                                          Denom
   ----    ----  ----  ------------                                         -----
      9       1    -   DEMEDC=currently attending, DEMEDL=never attended       -

CSPRO Executor Normal End
```

If you add a denominator using the *denom* keyword, CSPro will calculate the percentage of cases where the error occurred. The denominator to use depends on what is being counted. In our case it is the total number of household members between 5 and 29.

```
PROC GLOBAL
numeric numberOfHHMembers5to29 = 0;

PROC DEMEDC

if DEMAG1 in 5:29 then
    numberOfHHMembers5to29 = numberOfHHMembers5to29 + 1;

    // Check if currently attending and never attended school
```

```
    if DEMEDC = 1 and DEMEDL = 1 then
        errmsg("DEMEDC=currently attending, DEMEDL=never attended") summary
denom = numberOfHHMembers5to29;
    endif;
endif;
```

```
User unnumbered messages:

   Line    Freq  Pct.  Message text                                          Denom
   ----    ----  ----  ------------                                         -----
     14       1  25.0  DEMEDC=currently attending, DEMEDL=never attended       4

CSPRO Executor Normal End
```

Correcting Errors

In addition to using batch edit to find errors you can also use it to correct problems by modifying variables in your logic. Let's simply set the level of education to less than primary.

```
// Check if currently attending and never attended school
if DEMEDC = 1 and DEMEDL = 1 then
    errmsg("DEMEDC=currently attending, DEMEDL=never attended");
    DEMEDL = 2;
endif;
```

When we run this time, we will specify an output file: PopstanLFSEdited.csdat. The changes we make will only be made to the output file. We can then rerun the batch application on the output file and make sure that you don't have any error messages.

Instead of just assigning the value of `DEMEDL` we can use the *impute* command which does the assignment just like "=" but also generates a nice report showing the values that were imputed.

```
// Check if currently attending and never attended school
if DEMEDC = 1 and DEMEDL = 1 then
    errmsg("DEMEDC=currently attending, DEMEDL=never attended");
    impute(DEMEDL, 2);
endif;
```

The imputation report will be opened in TextViewer after you run the batch application but to see it you will need to go to the Window menu and choose the file that ends in ".frq.lst".

```
                          IMPUTE FREQUENCIES                              Page

_____________________________________________________________________________
   Imputed Item DEMEDL: Highest level of education - all occurrences
                                  _____________________________ _____________
 Categories                             Frequency       CumFreq       %  Cum %
_____________________________________ _____________________________ _____________
```

```
  2 Less than primary                    1              1   100.0   100.0
_____________________________ _____________________________ _____________
 TOTAL                                  1              1   100.0   100.0
```

Adding Calculated Variables

It is often useful to add additional variables to your data file after data collection that are computed from the collected variables. For example, let's add a yes/no variable EMPLOYED to the labor record that determines if the individual is employed. First, we add the new variable EMPLOYED to the dictionary (at the end so that we do not mess up our existing data). Then we add logic to the PROC of our new variable to impute the value.

```
PROC EMPLOYED
impute(EMPLOYED, EMPPAY=01 OR pos("3",AGFCHK)>0 OR AGFMKT=01 OR AGFMAI=01 OR
AGFHIS=01 OR AGFHIR=01);
```

Run the program and look at the imputation report to see how many employed people are in our data set.

Converting Checkboxes to Yes/No

The checkbox makes a nice interface but the resulting data is a string that is hard to interpret and deal with in other software. To make it easier to use for export we can convert from the checkbox to a repeating item with yes/no options. Create a new item in the labor record named AGFCHECK_YESNO with length 1 and 3 occurrences. Add a value set with Yes – 1 and No – 0.

For each item in the checkbox field that is checked we want to set the corresponding item in the repeating field to one and for each entry in the checkbox field that is not checked we set the corresponding item to two. To find out if an item is checked we use the function ***pos()*** which finds the position of one string in another and returns zero if the string is not found:

```
PROC AGFCHK

// Fill in crops repeating item based on crops checkboxes
if pos("1", AGFCHK) > 0 then
    AGFCHK_YESNO(1) = 1;
else
    AGFCHK_YESNO(1) = 0;
endif;
if pos("2", AGFCHK) > 0 then
    AGFCHK_YESNO (2) = 1;
else
    AGFCHK_YESNO (2) = 0;
endif;
if pos("3", AGFCHK) > 0 then
    AGFCHK_YESNO(3) = 1;
else
    AGFCHK_YESNO(3) = 0;
endif;
```

This works but it is a lot of code when the number of options is long. We can simplify it using a loop and a clever trick.

```
PROC AGFCHK

// Fill in crops repeating item based on crops checkboxes
numeric i;

// Loop from 1 to 3 and for each checkbox option
// find out it if it was checked by
// seeing if the i is in the checkbox field.
do i = 1 while i <= 3
    if pos(edit(i, "9")), AGFCHK) > 0 then
        AGFCHK_YESNO(i) = 1;
    else
        AGFCHK_YESNO(i) = 0;
    endif;
enddo;
```

Exercises

1. Add a check to see if the year of job start (DMJSYR) is before the year of birth. Print an error message when this occurs.
2. Add the derived variables for Unemployed, Potential Labor Force and Labor Force Status per the specifications.
3. Recode the checkbox field MJBREM to a repeating yes/no field.