Paper 227-2011
Macros for Managing Messy Data: Handling Duplicate Study Participants, and Making Fuzzy Matches Across Multiple Data Sets.

Jefferson L. Wood, Dennis G. Fisher, Grace L. Reynolds, & Lucy E. Napper
CSULB Center for Behavioral Research & Services, Long Beach, CA

ABSTRACT

The CSULB, Center for Behavioral Research & Services (CBRS) conducts behavioral research among hard-to-reach populations in Southern California. During data collection, CBRS staff members attempt to identify each research participant and link the participant to a unified participant record which details all prior contact with the agency. However, incomplete or inconsistent personal information can cause staff members to establish a new ID rather than amend an existing record. These errors are often identified after substantial research data has been collected. CBRS uses a SAS® macro and hash object to manage participants with multiple IDs without revising raw data sets or requiring analysts to know anything about the IDs established for an individual. Changes can be rolled-back in the rare case of a misidentified duplicate (e.g., twins or a Sr./Jr. relationship misidentified as the same individual). CBRS researchers also use a macro to accomplish fuzzy matching of one data set to any number of additional data sets by ID and date.

INTRODUCTION

The California State University Long Beach, Center for Behavioral Research & Services has conducted HIV, STD and other health related research in Long Beach California and surrounding communities since 1988. Research data collected by CBRS includes demographic information, contact records, laboratory test results, biological samples, and a wide variety of research instruments and questionnaires. Many participants are current or former drug users, sex workers, or have engaged in other activities associated with an elevated risk of exposure to disease. Monetary incentives and non-monetary rewards (such as grocery or fast food gift cards) are often employed by CBRS and are popular among study participants. Incentives, in conjunction with other outreach efforts, help to encourage individuals to maintain periodic contact with CBRS, comply with follow-up appointments and respond positively to invitations to participate in new research. When an individual initially participates in any research through CBRS, he or she is assigned a unique six-digit ID code establishing an individual research participation record. Staff members at CBRS make every effort to ensure that each individual is assigned only one ID code throughout their association with CBRS, without regard to amount of time between research contacts or the number of studies in which the individual participates.

In spite of their efforts, staff members sometimes establish new records and ID codes for individuals with a history of previous research participation through CBRS. This happens for a number of reasons. Home addresses and contact numbers among research participants change frequently; individuals may be known on the streets and to CBRS staff by more than one name, and dates of birth are sometimes misreported. Many individuals do not carry, or even possess any reliable form of identification, and participants often have very little reason to ensure they are correctly identified. In fact, misidentification may be rewarded if, by being misidentified, the client gains additional incentives. Over time, most of these errors are discovered by the research staff, however, this usually happens only after research data has been collected and compiled using the erroneous ID. Rather than attempt to correct raw data, paper instruments and file records for every instance of an incorrectly assigned ID, and in doing so risk the introduction of new errors and the creation of "orphan" records, CBRS employs a SAS Macro to recode records during analysis, assigning the lowest common ID to all of the records belonging to a person mistakenly assigned more than one ID code. The DUPTASTIC macro reads a list of known duplicates and assigns revised ID codes based both on errors which are explicitly identified in the list (e.g. if a=b, a=c, & a=d then Respondents b, c, & d are re-identified as Respondent a) and those which can be implied from the existing data (e.g. if a=b, b=d, & c=d then Respondents b, c, & d are re-identified as Respondent a). Because no permanent changes are made to the data or the underlying hard copies, the consequences of a misidentified duplicate are greatly reduced. Furthermore, individual data analysts are not required to maintain any list of known errors and in some cases have been able to recover missing data when a research instrument previously thought to have not been administered was discovered assigned to an alternate ID belonging to the same subject.

In conjunction with %DUPTASTIC, CBRS researchers also employ a second macro which facilitates matching a primary dataset to any number of additional datasets by a common ID variable and a date variable. The date variable must fall within a specified range relative to the primary dataset, but need not match precisely. Where more than one observation meets the matching criteria, the closest match is selected. The FUZZTASTIC macro is limited

to producing a dataset containing only a single ID variable and a date variable for each dataset named in the parameters. Additional variables can be easily added in subsequent steps, but even without them %FUZZTASTIC has proven immediately useful at CBRS as a data cleaning tool with it's ability to construct snapshots of study participation episodes which allow for "real world" events in which, for example an individual may complete a study instrument on one day, but return for the related blood draw a day or two later, then miss an intervention session which they had completed once before a year earlier. Using only the respondent ID as a matching criterion tended to mask missing measurements by allowing matches between events that occurred at very different times, while including date in the BY statement prevents matches which are "good enough" from being made. Other strategies using IF and WHERE statements or PROC SQL either failed to ensure that the best match would be selected, or allowed "one-to-many" type matches. This paper provides an overview of each of these two "Messy Data" taming macros.

PREPARING A LIST OF DUPLICATE ID'S

The %DUPTASTIC macro does not help to identify duplicate study participants. Its purpose is to manage data which contains observations from study subjects who are known to have established more than one "identity" in the research data. Before using %DUPTASTIC, a two-column list of the known duplicate identities should be constructed containing pairs of numeric identification codes which are known to belong to the same individual. CBRS assigns a six digit code to each research participant. The example below shows a partial list of code pairs. Each line represents two ID codes which were both assigned to one individual.

```
 ID_A   ID_B
252193 233116
235908 237069
350065 238109
239607 350346
237237 350095
230115 233116
237102 277553
234107 237499
277553 237102
...
```

The order of the list is arbitrary, as is the assignment of codes to the first or second column. Redundant entries are harmless, but should be avoided. In the example, respondent 233116 has established three separate research ID's: 252193, 233116, & 230115. When used in the replace mode %DUPTASTIC will reassign any observations with one of these three ID codes to the numerically lowest common value (in this case 230115.) Once the initial list is prepared, new pairs may be added as they are identified.

%DUPTASTIC: A MACRO TO HANDLE DUPLICATE STUDY PARTICIPANTS

The DUPTASTIC maco should be executed in open code, outside of any DATA or PROC step. It produces one temporary dataset, "DupIDs," in the work library and can be used in two distinct modes. If provided with a single numeric argument, the macro prints the argument along with any matching id's to the log. This allows the user to quickly determine if a particular pair of ID codes has been previously identified as a duplicate.

```
%Duptastic(252193);

ID Being tested: 252193
   Matching IDs: 252193
                 233116
                 230115
```

Providing the name of a valid numeric variable and dataset will cause each observation in the specified dataset to be recoded based on the current duplicate list. The original variable value is stored in a new variable called OLD_*varname* and a note is written to the log for each recoded observation.

```
%Duptastic(IDVar, MyData);

Record 271 : IDVar 252193  replaced with 230115
Record 415 : IDVar 350346  replaced with 239607
Record 419 : IDVar 237142  replaced with 232166
Record 484 : IDVar 277553  replaced with 237102
Record 582 : IDVar 233116  replaced with 230115
```

It is important to keep in mind that a dataset sorted by the ID variable will no longer be correctly sorted after %DUPTASTIC has run. If as sorted dataset is required then the PROC SORT; BY ID; statements should appear after %DUPTASTIC.

MACRO CODE PART 1: READ THE CURRENT LIST OF DUPLICATES

The first section of the macro code uses a DATA step to read the current list of duplicates and construct a temporary dataset which will be loaded into a SAS hash object. Each pair from the duplicate list is read twice to ensure that every duplicated id will be a KEY value in the resulting hash. The first two lines should be edited to suit the computing environment, fileNameVal should be a valid quoted FILENAME reference to the duplicate ID list, and second should be a valid numeric variable format. The next three lines determine the mode in which the %DUPTASTIC will run.

```
 /*--------------------------< %DUPTASTIC() >------------------------------*\
|   The Duptastic macro may be used in either a look-up or replace mode.      |
|                                |                                            |
|   Look-up mode: IDVar = Any single numeric value representing an ID code    |
|                 %Duptastic(237563);                    |                    |
|                 Prints the original ID with all know duplicate ID's to log  |
|                                |                                            |
|   Replace mode: IDVar = The name of a numeric ID variable                   |
|                 DataSetName = The name of a SAS dataset|                    |
|                 %Duptastic(MyidVar, work.mydata);           |               |
|                 Converts IDVar to the lowest know duplicate ID and saves    |
|                 the original variable value in the variable OLD_[IDVar]     |
|                 A list of replacements will be written to the Output log.   |
|                                |                                            |
|   Invoke the macro in open code, outside of any Data or Proc step.          |
 \*-------------------------------------------------------------------------*/
%MACRO Duptastic(IDVar, DataSetName );

%LET fileNameVal = "\\DATASERVER\SHARE\DUPLICATES\DUP_DATA*.DAT";
%LET IDVarLen = 6.;

%LET ShowDups = 0;
%IF %LENGTH(&DataSetName) = 0 %THEN %LET ShowDups = 1;
%IF &ShowDups = 1 %THEN %LET DataSetName = _NULL_;

    FILENAME duplist &fileNameVal;

DATA DupList;
    INFILE duplist  TRUNCOVER;
    INPUT @1 ID_a &IDVarLen ID_b &IDVarLen @@;
    OUTPUT;
    INPUT @1 ID_b &IDVarLen ID_a &IDVarLen;
    OUTPUT;
RUN;
```

MACRO CODE PART 2: DECLARE THE HASH OBJECTS

Three SAS hash objects and a hash iterator object are created in the next section. The first is a multidata hash object pre-loaded with the dataset created during the DATA step, the second is a sorted hash used to store ID

values identified as duplicates of the ID currently under investigation and to return the lowest value within a set of duplicates. The third hash contains a list of ID values which have been thoroughly inspected and, if encountered again, do not need to be checked. Because, duplicates will be identified and added to the sorted hash in an arbitrary order, a hash iterator object is employed to iterate through the sorted list of known duplicates checking for new values until no new values are encountered.

```
DATA &DataSetName (DROP = ID_a ID_b Matching_ID Test_ID RC N R Exhausted_ID);

%IF &ShowDups = 0 %THEN %DO;

          SET &DataSetName;
          OLD_&IDVar = &IDVar;

%END;

    IF _n_ = 1 THEN DO;

          CALL missing(ID_a,ID_b);
          CALL missing(Matching_ID,Test_ID);

          DECLARE hash AllDups( multidata: 'Y', dataset: "DupList");
                rc = AllDups.definekey('ID_a');
                rc = AllDups.definedata('ID_b');
          AllDups.definedone();

          DECLARE hash DupIDs();
                DupIDs = _NEW_ hash(ordered: 'a');
                rc = DupIDs.definekey('Matching_ID');
                rc = DupIDs.definedata('Matching_ID','Test_ID');
          DupIDs.definedone();

          DECLARE hash Exhausted_IDs(multidata: "N");
                Exhausted_IDs = _NEW_ hash();
                rc = Exhausted_IDs.definekey('Exhausted_ID');
          Exhausted_IDs.definedone();
    END;

    DECLARE hiter iter('DupIDs');
    rc=Exhausted_IDs.clear();
    Test_ID=&IDVar;
    Matching_ID=Test_ID;
    ID_A=Test_ID;

    N = DupIDs.add();
```

MACRO CODE PART 3: LOOPING THROUGH

The final section locates any recorded duplicate values for the current ID and loops through each data element in the multidata hash, adding these to the list of known duplicates. It then repeats the process using each of the identified values as the Key and adding any additional values to the list of known duplicates. The cycle is repeated until no new values are encountered. If %DUPTASTIC is being used in "look-up mode" each duplicate value is reported to the log as they are identified; log entries will not be sorted. In "replace mode" only the lowest value is returned after all duplicate values are identified. If no duplicate values are encountered, then the initial ID value is returned.

```
%IF &ShowDups = 1 %THEN %DO;

          IF (N = 0)THEN PUT @1 "ID Being tested: " Test_ID ;
          IF (N = 0)THEN PUT @4 "Matching IDs: " Matching_ID ;

%END;

    IF (N = 0) THEN DO;
```

```
            DO WHILE (N = 0);
                  ID_a = Matching_ID;
                  rc = AllDups.find();
                  IF (rc NE 0) THEN GOTO OUT;
                  Matching_ID = ID_b;
                  rc = DupIDs.add();
                  %IF &ShowDups = 1 %THEN %DO;
                        IF (rc = 0)THEN PUT @18 Matching_ID ;
                  %END;

                  AllDups.has_next(RESULT:R);
                  DO WHILE (R NE 0);
                        rc = AllDups.Find_next();
                        Matching_ID = ID_b;
                        rc = DupIDs.add();
                        %IF &ShowDups = 1 %THEN %DO;
                              IF (rc = 0)THEN PUT @18 Matching_ID ;
                        %END;
                        AllDups.has_next(RESULT:R);
                  END;

                  Exhausted_ID = ID_a;
                  rc = Exhausted_IDs.add();
                  rc = iter.First();
                  rc = Exhausted_IDs.check(KEY:Matching_ID);
                  IF (rc = 0) THEN DO WHILE(rc = 0);
                        rc = iter.Next();
OUT:                    IF (rc NE 0) THEN N = 1;
                        ELSE rc = Exhausted_IDs.check(KEY:Matching_ID);
                  END;
            END;

%IF &ShowDups = 0 %THEN %DO;

            rc = iter.First();
            IF (Matching_ID NE Test_ID) THEN DO;
               PUT "Record " _n_ ": &IDVar " Test_ID " replaced with " Matching_ID;
               &IDVar = Matching_ID;
            END;
            rc=iter.delete();
            rc=DupIDs.clear();

%END;

    END;
RUN;
%MEND;
```

%FUZZTASTIC: MERGING MULTIPLE DATA SETS BY ID AND A FUZZY DATE

The FUZZTASTIC macro constructs a merged dataset which contains all of the observations from an initial "Master" dataset, and data from any number of additional datasets up to the limits of the operating environment. Observations are matched by ID and date variables. The ID variables must match the master dataset, but date values may vary within a preset range, the "fuzz factor". When more than one observations falls within the limits of the fuzz factor, the observation with the best fit is used. Several conditions must be taken into account before using %FUZZTASTIC.

- Each dataset MUST contain ID and date variables of the same name and type.
- Each dataset must be sorted by the ID and date variable, if %FUZZTASTIC is used in conjunction with %DUPTASTIC remember to resort datasets after %DUPTASTIC executes.
- The combination of ID and date should uniquely identify each record within its respective dataset.
- The first dataset named in the argument list will be considered the "Master" dataset. The output dataset, WORK.FUZZY, will contain exactly the same number of observations as the master dataset, a single ID variable, and one date variable for each dataset named. If any additional variables are required they must be added to WORK.FUZZY in a subsequent step.

```
EXAMPLE:

%FUZZTASTIC(DataSets=MyData Merge1 Merge2, IDvar=Pid, DateVar=Date, FuzzFactor=14);

Proc Print;
RUN;
```

Obs	Pid	MyDataDate	Merge1Date	Merge2Date
1	102100	10/18/2002	.	.
2	123990	04/28/2005	.	.
3	202429	04/29/2009	.	.
4	230004	07/07/2009	07/07/2009	07/07/2009
5	230013	04/26/2007	.	.
6	230497	09/20/2006	.	.
7	231027	08/19/2009	.	08/19/2009
8	232111	01/21/2004	01/24/2004	.
9	232111	08/13/2004	08/13/2004	08/12/2004
10	232147	11/19/2004	.	11/17/2004
11	232147	04/08/2010	04/12/2010	04/12/2010
12	232166	04/28/2004	04/28/2004	04/28/2004
13	232193	03/29/2005	03/29/2005	03/29/2005
14	232219	07/29/2008	07/29/2008	07/29/2008
15	232219	11/14/2008	.	11/14/2008
16	232219	08/24/2010	08/22/2010	.
17	232225	01/15/2010	01/15/2010	01/15/2010
18	232225	05/25/2010	05/25/2010	05/25/2010

The Fuzztastic Macro uses a single DATA step to interleave observations from each dataset in order by ID and date. It depends on the RETAIN statement to allow comparison between new and prior observations, retaining whichever value is the best fit with the current observation in the master dataset. Execution of the OUTPUT statement depends on the value of two variables, "OutputFlag" which indicates whether or not an observation from the master dataset has been encountered since the last time the OUTPUT statement executed, and "Trigger" which, when true, indicates that the current observation either contains a new ID value, or a new observation from the master dataset, requiring that any retained values either be output to a new observation or flushed.

MACRO CODE PART 1: OUTPUT AN OBSERVATION (MAYBE)

During execution of the macro, the observations output to WORK.FUZZY by %FUZZTASTIC always reflect the value of retained data from previous iterations of the DATA step (with the possible exception of the last iteration.) For this reason the macro begins in each iteration by determining whether execution of the OUTPUT statement is necessary based on the value of the "OutputFlag" and "Trigger" variables.

The familiar %DO-%WHILE loop reoccurs throughout the macro in the following form:

```
%LET num=2;
%LET dsname=%SCAN(&DataSets,&num);
%DO %WHILE(&dsname ne);

    /*SOME CODE HERE WHICH INCLUDES &dsname;*/

    %LET num=%EVAL(&num+1);
    %LET dsname=%SCAN(&DataSets,&num);
%END;
```

This structure is used to scan the 'DataSets' parameter and assemble sets of statements or lists of variables which reference each of the datasets being merged into the master dataset. Refer to the comments within the following code to provide details about the function of each of the various sections.

```
 /*---------------------------< %FUZZTASTIC() >-----------------------------*\
|   The FUZZTASTIC macro matches the best fitting record from each of several |
|   datasets to a master dataset.  Produces a dataset in the work library     |
|   named FUZZY.                                                              |
|                                                                             |
|   DataSets = A list of datasets to be merged, the first dataset listed will |
|   provide the observations, data from the best fitting observation from     |
|   each of the remaining sets will be added.                                 |
|                                                                             |
|   IDVar = Subject ID variable name must be present in each dataset          |
|                                                                             |
|   DateVar = Date variable be must be present in each dataset, default value |
|             is "Date"                                                       |
|                                                                             |
|   FuzzFactor = Number of days by which date can vary from master observation|
|   and still be considered "matching".  Default value is 5.                  |
|                                                                             |
|   Invoke the macro in open code, outside of any Data or Proc step.          |
 \*-----------------------------------------------------------------------------*/
%MACRO Fuzztastic(DataSets = ,IDVar = , DateVar = Date, FuzzFactor = 5 );

*Begin data step, a KEEP statement retains the ID and 'best fitting' date     ;
*variable for each data step, a RENAME statement restores original variable   ;
*names (appended to dataset names for data variables)                         ;

DATA FUZZY (KEEP= Last&IDVar Out&DateVar

   %LET num=2;
   %LET dsname=%SCAN(&DataSets,&num);
   %DO %WHILE(&dsname ne);
 Out&dsname&DateVar
      %LET num=%EVAL(&num+1);
      %LET dsname=%SCAN(&DataSets,&num);
   %END;
   %LET dsname=%SCAN(&DataSets,1);
RENAME=(Last&IDVar = &IDVar Out&DateVar = &dsname&DateVar
   %LET num=2;
   %LET dsname=%SCAN(&DataSets,&num);
   %DO %WHILE(&dsname ne);
 Out&dsname&DateVar = &dsname&DateVar
      %LET num=%EVAL(&num+1);
      %LET dsname=%SCAN(&DataSets,&num);
   %END;
));

*A SET statement using BY IDVar and DateVar interleaves each of the sorted    ;
*datasets and retains up to two previous date values from each dataset along  ;
*with the date of the most recent observation from the master dataset which   ;
*has not yet been output, and the value of OutputFlag.                        ;
```

```
%LET dsMaster=%SCAN(&DataSets,1);
SET &dsMaster (IN=NewMaster)
   %LET num=2;
   %LET dsname=%SCAN(&DataSets,&num);
   %DO %WHILE(&dsname ne);
      &dsname(IN=OB_&dsname)
      %LET num=%EVAL(&num+1);
      %LET dsname=%SCAN(&DataSets,&num);
   %END;
END=LastRec;
BY &IDVar &DateVar;
Last&IDVar = LAG(&IDVar);
NewID = DIF(&IDVar);
FuzzFactor = &FuzzFactor;

RETAIN Out&DateVar
   %LET num=2;
   %LET dsname=%SCAN(&DataSets,&num);
   %DO %WHILE(&dsname ne);
 Last&dsname&DateVar Out&dsname&DateVar
      %LET num=%EVAL(&num+1);
      %LET dsname=%SCAN(&DataSets,&num);
   %END;
 OutputFlag 0;

*The next loop prevents output of date values which fall outside the          ;
*acceptable range set by FuzzFactor.                                          ;

   %LET num=2;
   %LET dsname=%SCAN(&DataSets,&num);
   %DO %WHILE(&dsname ne);
IF ABS(Out&dsname&DateVar - Out&DateVar)> FuzzFactor THEN Out&dsname&DateVar = .;
  %LET num=%EVAL(&num+1);
      %LET dsname=%SCAN(&DataSets,&num);
   %END;

*Encountering a new subject ID requires that retained values be output if     ;
*necessary.                                                                   ;

IF NewID THEN DO;
   Trigger = 1;

*Compare recently retained observation dates from each dataset to current     ;
*tentative output values and FuzzFactor.                                      ;

   %LET num=2;
   %LET dsname=%SCAN(&DataSets,&num);
   %DO %WHILE(&dsname ne);
         IF ABS(Last&dsname&DateVar-Out&DateVar)<
         ABS(IFN(Out&dsname&DateVar,Out&dsname&DateVar,0,0) - Out&DateVar)
         AND ABS(Last&dsname&DateVar-Out&DateVar)<=FuzzFactor THEN
         Out&dsname&DateVar=Last&dsname&DateVar;
      Last&dsname&DateVar = .;
      %LET num=%EVAL(&num+1);
      %LET dsname=%SCAN(&DataSets,&num);
   %END;
END;

*Encountering a new observation in the master dataset also requires that      ;
*retained values be output if necessary.                                      ;

ELSE IF NewMaster THEN DO;
Trigger = 1;

*In this case the previous observation from the master dataset and the current;
*observation have the same subject ID.  Retained date values must be compared ;
*to each to determine the best fit.                                           ;
```

```
   %LET num=2;
   %LET dsname=%SCAN(&DataSets,&num);
   %DO %WHILE(&dsname ne);
            IF ABS(Last&dsname&DateVar- Out&DateVar)<
            ABS(IFN(Out&dsname&DateVar,Out&dsname&DateVar,0,0)-Out&DateVar)
            AND ABS(Last&dsname&DateVar-Out&DateVar)<=FuzzFactor
            AND ABS(Last&dsname&DateVar-Out&DateVar)<=
            ABS(Last&dsname&DateVar-&DateVar)THEN DO;
               Out&dsname&DateVar=Last&dsname&DateVar;
               Last&dsname&DateVar = .;
           END;
      %LET num=%EVAL(&num+1);
      %LET dsname=%SCAN(&DataSets,&num);
   %END;
END;

*If retained output data exists and a new ID or new master record has been    ;
*encountered, then an observation is written to WORK.FUZZY and retained output;
*values are cleared.                                                          ;

IF Trigger AND OutputFlag THEN DO;
          OUTPUT;
          OutputFlag=0;
          Out&DateVar=.;

   %LET num=2;
   %LET dsname=%SCAN(&DataSets,&num);
   %DO %WHILE(&dsname ne);
        Out&dsname&DateVar=.;
      %LET num=%EVAL(&num+1);
      %LET dsname=%SCAN(&DataSets,&num);
   %END;
                                                                        END;
```

MACRO CODE PART 2: EXAMINE AND RETAIN THE CURRENT OBSERVATION (MAYBE)

Once the retained values from previous iterations of the DATA step have been either output as a new observation, purged, or held over for possible future use, the current observation can be examined and possibly retained. The first IF-THEN block in this section of the macro assigns a new observation date and sets the OutputFlag variable if the current record represents a new observation in the master dataset. As in the previous section, refer to comments within the code for additional details.

```
IF NewMaster THEN DO;
    Out&DateVar= &DateVar;
    OutputFlag=1;
END;

*This loop first determines the source of the current record if it is NOT from;
*the master dataset.  It then compares the current observation to retained    ;
*values from the same dataset and retains the observation which fits best.    ;

   %LET num=2;
   %LET dsname=%SCAN(&DataSets,&num);
   %DO %WHILE(&dsname ne);
IF OB_&dsname THEN DO;
    IF OutputFlag THEN DO;
          IF ABS(Last&dsname&DateVar-Out&DateVar)<
          ABS(IFN(Out&dsname&DateVar,Out&dsname&DateVar,0,0)-Out&DateVar)AND
          ABS(Last&dsname&DateVar-Out&DateVar)<=FuzzFactor THEN
          Out&dsname&DateVar=Last&dsname&DateVar;
          Last&dsname&DateVar = &DateVar;
    END;
    ELSE DO;
```

```
            Last&dsname&DateVar = &DateVar;
            Out&dsname&DateVar=Last&dsname&DateVar;
    END;
END;

       %LET num=%EVAL(&num+1);
       %LET dsname=%SCAN(&DataSets,&num);
    %END;

*Finally, if the last record is encountered, it then assigns a format to the ;
*date variables in WORK.FUZZY and outputs the final observation if necessary.;

IF LastRec THEN DO;
FORMAT Out&DateVar
   %LET num=2;
   %LET dsname=%SCAN(&DataSets,&num);
   %DO %WHILE(&dsname ne);
      Out&dsname&DateVar
      %LET num=%EVAL(&num+1);
      %LET dsname=%SCAN(&DataSets,&num);
   %END;
   mmddyy10.;
    IF OutputFlag THEN OUTPUT;
END;
RUN;
%MEND Fuzztastic;
```

CONCLUSION

The FUZZTASTIC and DUPTASTIC macros have been employed at CBRS since 2009 to help manage and clean data collected over the past decade. While they were initially conceived and written to address specific needs, they are flexible enough to address a variety of situations without modification. Minor revision would allow them to be applied to other data formats and types, or to expand their functionality with additional options. The un-annotated code, including any recent revisions or corrections is available through the contact information below.

ACKNOWLEDGMENTS

The project described was supported by Award Number P20MD003942 from the National Center on Minority Health and Health Disparities. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Center on Minority Health and Health Disparities or the National Institutes of Health.

Support for Dr. Napper was provided by fellowship #F32DA022902 from the National Institute on Drug Abuse.

CONTACT INFORMATION

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

Name:	Jefferson L. Wood
Enterprise:	CSULB Center for Behavioral Research & Services
Address:	1090 Atlantic Ave.
City, State ZIP:	Long Beach, CA 90813
Work Phone:	(562) 495-2330 ext.141
Fax:	(562)983-1421
E-mail:	jlwood@csulb.edu