Paper AD-130

Rapidly Assessing Data Completeness

David H. Abbott, Veterans Affairs Health Services Research

ABSTRACT

Data analysts are often asked to work with collections of data sets prepared by others and with varying degrees of history/documentation. An important early question is, "How complete are these data? What data completeness issues might be present?" This paper presents an efficient technique for addressing this question both in terms of characterizing the number and patterns of missing values and, similarly, the omitted rows of data (i.e., primary identifier values not occurring in a given data set and occurring in some other dataset).

Several short macros and two key algorithms enable the technique presented. The first algorithm produces a table of missing value patterns in the style of PROC MI on a per dataset basis. The second performs the manipulations needed to exhibit patterns of missing identifiers across a collection of datasets.

Following this technique, analysts will be able to rapidly assess data completeness of inherited data set collections, provided a primary identifier (e.g., a subject ID) is used consistently in the collection.

THE CHALLENGE

Data analysts often have experience with this type of scenario: the boss says, "Here are the 26 SAS® datasets that belong to this project we are inheriting from another group. The documentation is quite limited and the former owners are difficult to reach. Please investigate the completeness of the data and let me know about any problems pronto."

The first step to meet this challenge is to formulate the goals for the investigation:

- Examine counts of missing values for each variable in each dataset.
- Examine patterns of missing values among variables in each dataset, e.g. when BMI is missing height and weight are also missing.
- Examine counts of omitted identifiers in each dataset, e.g., the count is 2 when Alabama and Texas are omitted from a file of death statistics by state.
- Examine patterns of omitted identifiers among the datasets, e.g., Alabama and Texas not being present in datasets dealing with health-related statistics.

This paper shows how to achieve these goals rapidly using a systematic approach and a small set of macros designed for this task.

FOUR USEFUL DISPLAYS

The preceding four goals are naturally enough addressed by the generation of four displays, one per goal, and cast as output datasets. An example of each report is given in this section.

The examples were created by applying the macros in this paper to three data sets representing medical data. Each dataset has some missing values injected into it randomly using one or more values of *p,* where *p* is the probability that any given value of the target variable is set to missing. Specifically, the datasets are:

1. PATIENTS – 100 rows of patient data designed to have 0 missing PatientIDs with about 20% of the values of PATIENTFIRST, AGE, and WEIGHT set to missing.
2. VISITS – 200 rows of visit data fabricated to have two visits recorded for 90 of the 100 patients with about 35% of the rows of having SURGERY and ENDDAY set to missing.

3. CAREEVENTS – 300 rows of care event data created to have 3 care events recorded for 85 of 100 patients with about 15% of rows having PROVID and NOTES missing and about 30% having STARTTIME and ENDTIME missing.

The percentages above are noted as “about” because the assignment of missing values is probabilistic and the count of actual missing values differs some from expected count. This was done to enhance verisimilitude.

Display 1. Missing values reported by %missingValuesRpt

	Dataset	Variable	MissCnt	PresCnt	pctMiss
1	CAREEVENTS	provID	42	213	16
2	CAREEVENTS	patientID	0	255	0
3	CAREEVENTS	eventDay	0	255	0
4	CAREEVENTS	startTime	76	179	30
5	CAREEVENTS	endTime	76	179	30
6	CAREEVENTS	notes	42	213	16
7	PATIENTS	patientFirst	20	80	20
8	PATIENTS	patientLast	0	100	0
9	PATIENTS	patientID	0	100	0
10	PATIENTS	age	20	80	20
11	PATIENTS	weight	20	80	20
12	VISITS	patientID	0	180	0
13	VISITS	startDay	0	180	0
14	VISITS	endDay	59	121	33
15	VISITS	surgery	59	121	33

This report shows the number of values missing and present for each of the datasets of interest for variables in these datasets. As expected, these specific realizations of PATIENTS, VISITS, AND CAREEVENTS exhibit the approximately 20%, 35%, 15%, and 30% of missing values intended in their construction.

Readers may recognize the formatting of the display as that of the **SAS Viewtable**. That is because %missingVauesRpt does not produce a formatted report, rather, it creates a SAS dataset that users can format up as they so choose with PROC PRINT, PROC REPORT, PROC TABULATE, or whatever. The other macros in this set of macros follow the same principle - create an output dataset rather than a formatted report.

Display 2. Missing value patterns reported by calls of %missingDataPatterns

CareEvents

	Pattern ID	Occurs	provID	patientID	eventDay	startTime	endTime	notes
1	0	179	X	X	X	X	X	X
2	114	42	.	X	X	.	.	.
3	48	34	X	X	X	.	.	X

Patients

	Pattern ID	Occurs	patientFirst	patientLast	patientID	age	weight
1	0	80	X	X	X	X	X
2	50	20	.	X	X	.	.

Visits

	Pattern ID	Occurs	patientID	startDay	endDay	surgery
1	0	121	X	X	X	X
2	24	59	X	X	.	.

The missing patterns display is formatted in the manner used by MissPattern table of PROC MI – an "X" stands for any non-missing value and a "." for a missing value. This display complements the missing values display, e.g., it reveals that the missing values of startTime and endTime (of CareEvent) occur together and not separately. It shows that 179 of 255 CareEvent observations are complete, i.e., have no missing values. It shows that the missing values of endDay and surgery (of Visits) occur together.

Display 3. Counts of omitted identifiers reported by %idOccursSumry

	Dataset	Present	Absent	AbsPercent
1	CAREEVENTS	85	15	15
2	PATIENTS	100	0	0
3	VISITS	90	10	10

The two preceding displays both showed that the patientID has no missing values in any of the three data sets. However, that does not imply that each dataset includes occurrences of all 100 patientID values. This display shows that 10 patients are absent from the VISITS dataset and 15 patients have no recorded CareEvent. This may or may not be a data completeness issue, though having 5% of visits without any associated CareEvent entry might raise suspicions.

Display 4. Patterns of omitted identifiers reported by %missingDataPatterns

	Pattern ID	Occurs	patientID	CAREEVENTS	PATIENTS	VISITS
1	0	85	X	X	X	X
2	20	10	X	.	X	.
3	4	5	X	.	X	X

This display makes it clear that 10 patients have neither Visits nor CareEvents, i.e. their PatientID does not occur in either of these two datasets. The presence of 5 patients who have one or more visits but no CareEvents is made even more clear in this display.

FRAMEWORK FOR REPETITION

The four displays above constitute a good first pass at assessing data completeness in a collection of datasets. The key question that remains is how to most effectively implement the set of macros that generate these displays.

The work required to accomplish the displays is very much concerned with repetition – repeating certain operations over datasets and variables. So, it is beneficial to have a framework for accomplishing the repetition. The framework used here is composed of three parts:

1. A macro fpr enumerating the datasets to be operated on, **%datasetsOf**
2. a macro for enumerating the variables occurring in each of these datsasets, **%variablesOf**
3. a macro for performing a chunk of work over a list values, e.g., datasets or variables, %**applyForValues.**

DatasetsOf Signature

```
%macro datasetsOf( /* set &macvar to a string listing dataset names */
  lib=,   /* library for which a list of datasets is desired */
  macvar=, /* macro variable name, set to null string before call */
);
```

The only tricky thing about this macro is that it returns the list of datasets as the value of a macro variable, whose name is given by macvar= and that macro variable must be in the callers macro symbol table. The easiest way to ensure that it is in said symbol table is to set this macro variable to null before calling datasetsOf, for example:

```
%let dsList=; %datasetsOf(lib=Save,  macvar=dsList);
```

After the above line is executed, &dsList evaluates to a blank separated list of the data sets in the indicated library.

VariablesOf Signature

```
%macro variablesOf( /* resolves to a string listing variable names */
  dsn=   /* dataset for which a list of variables is desired */
);
```

This macro is very convenient to use because it acts like a true function and can be invoked wherever in a SAS statement and doesn't need to occur in a statement by itself like %datasetsOf, for example:

```
%put Variables of VISITS are: %variablesOf(dsn=Save.VISITS);
```

This statement simply puts a line listing the variables of Save.VISITS in the SAS log.

ApplyForValues Signature

```
%macro applyForValues(/*Invoke a macro once for each value in a list*/
  valueList=, /* list of values with blanks separating */
  invokedMacro=,/* name of macro to invoke for each value */
);
```

This macro is very much the key to getting list-driven work done conveniently and succinctly in SAS. The part that may be unfamiliar to users is defining a macro to be used as an argument to another macro. Happily, SAS supports this form of procedural abstraction and it works well in %applyForValues. For example, here is the code for listing the variables of all the data sets in library Save to the SAS log:

```
%macro putVarsToLog(lib=);
  %local vars;
  %put; %* make easier to read;
  %let vars= %variablesOf(dsn=&lib..&value);
  %put &value variables are: &vars;
%mend;
%let dsList=; %datasetsOf(lib=Save, macvar=dsList);
%applyForValues(valueList=&dsList, invokedMacro=putVarsToLog(lib=Save));
```

The resulting log file entries are:

```
CAREEVENTS variables are: provID patientID eventDay startTime endTime notes

PATIENTS variables are: patientFirst patientLast patientID age weight

VISITS variables are: patientID startDay endDay surgery
```

PRODUCING THE DISPLAYS

For a user intending to produce the four displays shown for a given set of datasets, the first step is to arrange for the datasets of interest, and only the datasets of interest, to reside in a SAS library created by the user. Typically, this will require creating a new data set library, e.g.,

```
LIBNAME  ProjDs "<a folder or directory name>";
```

and then moving or copying the datasets of interest to this library. The second step is to make the macros defined in this paper usable in the SAS session, e.g.,

```
%include "H:\SESUG2015\datasetsOf.sas";
%include "H:\SESUG2015\variablesOf.sas";
%include "H:\SESUG2015\applyForValues.sas";
%include "H:\SESUG2015\countMissings.sas";
%include "H:\SESUG2015\missingValuesRpt.sas";
%include "H:\SESUG2015\missingDataPatterns.sas";
%include "H:\SESUG2015\idOccursSumry.sas";
```

After this bit of preparatory work, the displays can be produced as detailed in subsections below.

Counts of missing values

The desired display is produced directly with the use of:

```
%macro missingValuesRpt(/* Generate missing values report */
  lib= /* library containing all datasets of interest */,
  dsList= /* specific set of datasets to analyze */,
  dsOut= /* output dataset containing counts of missing values */
);
```

Only a single invocation of %missingValuesRpt is required, e.g.,

```
%let dsList=; %datasetsOf(lib=ProjDs, macvar=dsList);
%missingValuesRpt(lib=ProjDs, dsList=&allDs, dsOut=work.mvrOut);
```

Patterns of missing values

The applicable macro for this display is:

```
%macro missingDataPatterns(/* analyze missing data patterns, like PROC MI
                              but without all its baggage */
  dataIn=,    /* dataset to be analyzed */
  vars=,      /* variables thereof to be analyzed */
  dataOut=,   /* resultant dataset depicting missing value patterns */
```

```
);
```

Each data set has its own missing data patterns, so the macro needs to be invoked once for each and this can easily be accomplished with the help of %applyForValues, e.g.,

```
%macro tmpMac(lib=);
  %missingDataPatterns(dataIn=&lib..&value,
       vars=%variablesOf(dsn=&lib..&value), dataOut=MP_&value.);
%mend tmpMac;
%applyForValues(valueList=&dsList,invokedMacro=tmpMac(lib=ProjDs));
```

Counts of omitted rows

The applicable macro for this display is:

```
%macro idOccursSumry( /*generates ID occurrence summary dataset*/
  lib=,       /* library containing datasets of interest */
  dsList=,    /* datasets to be analyzed */
  idVar=,      /* variable composing the primary ID, only 1 allowed now */
  idOccursOut=,   /* resultant dataset providing ID occurrence matrix */
  sumryOut=,   /* resultant dataset depicting occurrence summaries */
);
```

Only a single invocation of %idOccursSumry is required, e.g.,

```
%idOccursSumry(lib=ProjDs, dsList=&dsList, idVar=patientID,
               idOccursOut=work.occOut, sumryOut=work.sumOut);
```

Note that %idOccursSumry uses two output data sets. The data set providing the count of omitted rows in the specified data sets is &sumryOut. The idOccursOut dataset is captured for subsequent use with the fourth and final display.

Patterns of omitted Identifiers

No additional macro is required to produce the display for the patterns of omitted identifiers. Rather, the %missingDataPatterns macro can be reused to generate the display. The trick that makes this possible is representing an omitted identifier in the &idOccursOut data set as a missing value of a variable named the same as one of the analyzed datasets. So, the code required to generate the display is,

```
%missingDataPatterns(dataIn=OccOut,
  vars=%variablesOf(dsn=OccOut), dataOut=work.MissingIDpatterns);
```

NOTES ON IMPLEMENTATION

Full source for the macros is given in the Appendix; however, some comments about interesting aspects of these implementations may be helpful:

- The signatures of the macros in the Appendix are not, in general, identical to the signatures cited above since some signatures were presented in simplified form above for improved pedagogy.
- The versions of the macros in the Appendix may not be the best available versions at the time readers find they need to employ them. Contact the author for the latest versions.
- One question that comes up regarding %idOccursSumry is how the macro determines the universal set of IDs (so that omitted IDs can be identified and counted). The approach is basically brute force; Values of the ID variable (along with the data set in which it occurs) are extracted from all the datasets of interest (&dsList) and a set of distinct values is thereby compiled. Using the same list of ID occurrences, it is straight-forward to derive the &idOccursOut

dataset that has a row for each distinct ID value and indicators showing which datasets a given identifier value does and does not occur in.

- The key to implementing %missingDataPatterns is getting a good representation for the pattern identifier. A given pattern of missing values is an array of ones and zeros where each element of the array represents one of the variables in the dataset. Such an array can be uniquely named, sorted, etc. by a numeric value whose binary digits correspond to the values of the array. This approach does, however, limit the number of variables analyzed for missing patterns at one time to 53 or less.
- Why not just use PROC MEANS NMISS to do the work that %countMissings does via DATA steps? The main reason is that NMISS can't be used with character variables so handling a dataset with mixed numeric and character variables is problematic.

TAKE AWAYS

- Well-designed set of task-oriented macros can expedite assessment of data completeness.
- There are advantages to a "tool box" approach rather than the grand macro approach.
- It is possible to rapidly assess data completeness without extraordinary effort.

REFERENCES

- Cody, Ron. 2008. *Cody's Data Cleaning Techniques Using SAS, Second Edition*. Cary, NC: SAS Institute Inc.
- Schwarz, Teresa; Chen, Qixuan; Duan, Naihua. 2011. "Studying Missing Data Patterns Using a SAS® Macro." *Proceedings of SAS Global 2011 Conference*
- Lindsey Brown Philpot, Gabriela Cantu. 2012. "Dirty Data? Clean it up with SAS®." *Proceedings of SCSUG 2012 Conference*

ACKNOWLEDGMENTS

The views expressed in this paper are those of the author and do not necessarily reflect the position or policy of the Department of Veterans Affairs or the United States government.

Without the leadership and encouragement of Dr. Dawn Provenzale, director of the Durham Epidemiologic Research and Information Center at the Durham VA Medical Center, this work could not have occurred. She takes a strong interest in fostering many dimensions of excellence in her employees.

CONTACT INFORMATION

Name	David H. Abbott
Enterprise	Center for Health Services Research in Primary Care
Address	Durham Veterans Affairs Medical Center
	HSR&D Service (152)
	508 Fulton St.
City, State ZIP	Durham, NC 27705
Work Phone:	919-286-0411
E-mail:	david.abbott@va.gov

APPENDIX — SOURCE CODE FOR MACROS

```
%macro datasetsOf( /* set &macvar to a string listing dataset names */
  lib=,   /* library for which a list of datasets is desired */
  macvar=, /* macro variable name, set to null string before call */
  sp=tmp /*variable name starting with this prefix is safe to overwrite*/
);
/* sample use: %let res=;%datasetsOf(lib=sashelpm, macvar=res)
   => this macro derived from SAS sample 25083
*/
  ods output Members=&sp.Mems;
  ods listing close;
  proc datasets library=&lib;
  quit;
  ods listing;
  ods output;
  data &sp.xxx;
    retain rtnStr;
    length rtnStr $8146;
    set work.&sp.Mems  end=lastRec;
    if _N_ eq 1 then string = "";
    rtnStr = catx(" ", rtnStr, name);
    *put _N_ rtnStr;
    if lastRec then call symputx("&sp.Holdit", rtnStr, "L");
  run;
  %let &macvar=&&&sp.Holdit;
%mend;


%macro variablesOf( /* resolves to a string listing variable names */
  dsn=   /* dataset for which a list of variables is desired */
);
%* sample use: %let vars=%variablesOf(dsn=myDs)
  => this macro derived from SAS sample 25083
;
  %local dsid cnt rc i rtnStr;
  %let dsid=%sysfunc(open(&dsn));
  %let cnt=%sysfunc(attrn(&dsid,nvars));

  %do i = 1 %to &cnt;
    %let rtnStr=&rtnStr %sysfunc(varname(&dsid,&i));
  %end;
  %let rc=%sysfunc(close(&dsid));
  &rtnStr
%mend variablesOf;


%macro applyForValues(/*Invoke a macro once for each value in a list*/
  valueList=, /* list of values with blanks separating */
  invokedMacro=,/* name of macro to invoke for each value */
  mvName=value /* macro variable name used to carry the value */
); /*
Sample invocation:
  %macro myMac();%put &value; %mend;
  %applyForValues(valueList=f1 f2 f3, invokedMacro=myMac);
```

```
*/
  %local &mvName cnt;
  %let cnt=0;
  %do %while(1 eq 1);
    %let cnt = %eval(&cnt+1);
    %let &mvName = %scan(&valueList,&cnt,%str( ));
    %if &&&mvName= %then %return;
    %&invokedMacro;  %* macro references &value;
  %end;
%mend applyForValues;


%macro countMissings(
  dataIn=,    /* dataset to be analyzed */
  vars=,      /* variables thereof to be analyzed */
  dataOut=,   /* resultant dataset depicting missing value counts */
  cleanUp=1,  /* controls whether scratch datasets are cleaned up */
  sp=tmp      /* scratch prefix, i.e. WORK datasets, macro variables,
                 and variables names beginning with same are fair game*/
);
%* Sample invocation: %countMissings(dataIn=xxx,vars=a b c,dataOut=yyy)

Required macros: %applyForValues innerCountMissings(in this file)
Assumptions:
  => Variable names are 30 chars or less in &dataIn
  => A pair of variables with names of the from  m_<string1> and <string1>
     do not exist in &dataIn (for any string1)
  => The &sp value has been suitably set to prevent any name conflict issues
;

%let varNameListM=;
%let varNameListP=;
data &sp.1;
  set &dataIn end=lastRec;
  %applyForValues(valueList=&vars, invokedMacro=innerCountMissings);
  if lastRec then output;
  keep &varNameListM &varNameListP;
run;

proc transpose data=&sp.1 out=&sp.2a;
  var &varNameListM; run;
data &sp.outa;
  set &sp.2a;
  label _name_="Variable";
  rename col1=MissCnt _name_=Variable;
run;
proc transpose data=&sp.1 out=&sp.2b;
  var &varNameListP; run;
data &sp.outb;
  set &sp.2b;
  label _name_="Variable";
  rename col1=PresCnt _name_=Variable;
run;
options mergenoby=warn;
data &dataOut;
  merge  &sp.outa &sp.outb;
  label _label_="Variable";
  drop variable;
```

```
run;
options mergenoby=error;


  %put =>> results written to &dataOut;

* clean up;
%if &cleanUp eq 1 %then %do;
   proc datasets library=work nolist;
     delete &sp.1 &sp.2   ;
     quit;
   run;
%end;
%mend;

%macro innerCountMissings;
  if missing(&value) then do;
    m_&value + 1;
  end;
  else p_&value +1;
  label m_&value="&value";
  label p_&value="&value";
  %let varNameListM= &varNameListM m_&value;
  %let varNameListP= &varNameListP p_&value;
%mend;


%macro missingValuesRpt(
  lib=,
  dsList=,
  dsOut=,
  sp=tmp
);
%* Sample invocation:
    %missingValuesRpt(lib=Save, dsList=&dsList, dsOut=rpt)
   Required macros: %countMissings %applyForValues;

  proc datasets library=work nolist; delete &sp.base; quit;
  %applyForValues(valueList=&dsList, invokedMacro=innerMVR);
  data &dsOut;
    length ds $32;
    set &sp.base;
    pctMiss=100*missCnt/(missCnt+presCnt);
    format pctMiss 4.;
    label ds="Dataset";
  run;
%mend;

%macro innerMVR;
  %local vars;
  %let vars=%variablesOf(dsn=&lib..&value);
  %countMissings(dataIn=&lib..&value, vars=&vars,dataOut=&sp.11);
  data &sp.12; set &sp.11; length ds $32; ds="&value"; run;
  proc append base=&sp.base data=&sp.12;run;
%mend;
```

```
%macro missingDataPatterns(/* analyze missing data patterns, similar to proc
MI
                           but without the associated contrivances and
problems */
  dataIn=,    /* dataset to be analyzed */
  vars=,      /* variables thereof to be analyzed */
  dataOut=,   /* resultant dataset depicting missing value patterns */
  leaveSp1=0, /* set to 1 to keep &sp.1 DS even when cleanUp=1 */
  cleanUp=1,  /* controls whether scratch datasets are cleaned up */
  sp=tmp      /* scratch prefix, i.e. WORK datasets, macro variables,
                 and variables names beginning with same are fair game*/
);
%*
Sample invocation: %missingDataPatterns(dataIn=xxx,vars=a b c,dataOut=yyy)
Required macros: %applyForValues %innerMissingDataPatterns(appended)

Assumptions:
  => fails when large number of variables (>53) are in the vars= list
     this is due to a precision limitation in SAS
     (my tests show that 9999999999999998 is the largest integer represented
exactly
     and 2**54 is greater that this number)
  => Variable names are 30 chars or less in &dataIn
  => A pair of variables with names of the from  i_<string1> and <string1>
     do not exist in &dataIn (for any string1)
  => The &sp value has been suitably set to prevent any name conflict issues
;

%let nvars=words(&vars);
%let varNameList=;
data &sp.1;
  format &sp.PatNum 18.; /* work-around for proc freq bug */
  set &dataIn;
  &sp.Incre = 1; &sp.PatNum=0;
  %applyForValues(valueList=&vars, invokedMacro=innerMissingDataPatterns);
  drop &sp.Incre;
run;

* get the counts for each Pattern Number;
proc freq data=&sp.1 order=freq noprint;
  tables &sp.PatNum / out= &sp.2;
run;
proc sort data=&sp.2; by &sp.PatNum; run;

* get the indicator values for each pat num;
proc sort data= &sp.1; by &sp.PatNum;
data &sp.3 ;
  set &sp.1; by &sp.PatNum;
  if first.&sp.PatNum then output;
run;

* Merge the two data streams;
data &dataOut &sp.BadOnes;
  merge
    &sp.2(in=inCnt rename=(count=&sp.Count))
    &sp.3(in=inPat);
  by &sp.PatNum;
  label &sp.Count="Occurs";
```

```
  if (not inCnt) or (not inPat) or missing(&sp.Count) then output
&sp.BadOnes;
  else output &dataOut;
  label &sp.PatNum="Pattern ID";
  keep &sp.PatNum  &sp.Count &varNameList;
run;
proc sort data=&dataOut; by descending &sp.Count; run;

%if &leaveSp1 eq 1 %then %let toDelete=&sp.2 &sp.3;
%else %let toDelete=&sp.1 &sp.2 &sp.3;

* clean up;
%if &cleanUp eq 1 %then %do;
   proc datasets library=work nolist;
     delete &toDelete;
     quit;
   run;
%end;
%mend;

%macro innerMissingDataPatterns;
  &sp.Incre = &sp.Incre * 2;
  if missing(&value) then do;
    &sp.PatNum + &sp.Incre;
    i_&value = ".";
  end;
  else i_&value = "X";
  label i_&value="&value";
  %let varNameList= &varNameList i_&value;
  *put "&value " &sp.PatNum;
  * NOTE: i_&value gets set identically for all instances of
    a pattern for all variables under consideration - kind
    of repetitive but easier than deriving from the patnum;
%mend;


%macro idOccursSumry( /*generates ID occurrence summary dataset*/
  lib=,       /* library containing datasets of interest */
  dsList=,    /* datasets to be analyzed */
  idVar=,      /* variable composing the primary ID - currently only 1
allowed */
  idOccursOut=,   /* resultant dataset providing ID occurrence matrix */
  sumryOut=,   /* resultant dataset depicting occurrence summaries */
  cleanUp=1,  /* controls whether scratch datasets are cleaned up */
  sp=tmp      /* scratch prefix, i.e. WORK datasets, macro variables,
                 and variables names beginning with same are fair game*/
);
* invoke ex.: idOccursSumry(lib=Save, dsList=&dsList, idVar=patientID,
               idOccursOut=res, sumryOut=sumryOut1) ;

* Collect up all occurrences of &idVar into OccursAll;
options nofmterr;

proc datasets library=work nolist; delete &sp.Occurs;quit;
%macro collectUp(lib=);
  data occursTmp;
    set &lib..&value;
    length dsName $32;
```

```
    *format &idVar 6.;
    dsName="&value";
    keep dsName &idVar;
  run;
  proc append base=&sp.Occurs data=occursTmp; run;
%mend;
%applyForValues(valueList=&dsList., invokedMacro=collectUp(lib=&lib.));

* Collapse to 1 row per ID with indicators for each dataset;
%macro initIndic;
  retain &value;
  &value=.;
%mend;
%macro setIndic;
  if dsName eq "&value" then &value = 1;
%mend;
proc sort data=&sp.Occurs out=&sp.All; by &idVar; run;
data &idOccursOut;
  set &sp.All(where=(not missing(&idVar))); by &idVar;
  if first.&idVar then do;
    %applyForValues(valueList=&dsList, invokedMacro=initIndic);
  end;
  %applyForValues(valueList=&dsList, invokedMacro=setIndic);
  if last.&idVar then output;
drop dsName;
run;

%* Summarize the absent/present ID counts;
%macro tmpIncre;
  if not missing(&value) then P_&value +1;
  else A_&value+1;
%mend;
%macro tmpOutit;
  length Dataset $24;
  Dataset="&value";
  Present=P_&value;
  Absent=A_&value;
  AbsPercent=100*Absent/(Absent + Present);
  output;
%mend;

data &sumryOut(keep= Dataset Absent Present AbsPercent);
  set &idOccursOut end=lastRec;
  %applyForValues(valueList=&dsList, invokedMacro=tmpIncre);
  if lastRec then do;
    %applyForValues(valueList=&dsList, invokedMacro=tmpOutit);
  end;
run;

%mend idOccursSumry;
```