Paper 1340-2015

Using SAS® Macros to Flag Claims based on Medical Codes

Andy Karnopp, Fred Hutchinson Cancer Research Center

ABSTRACT

Many epidemiological studies use medical claims to identify and describe a population. But finding out who was diagnosed, and who received treatment, isn't always simple. Each claim can have dozens of medical codes, with different types of codes for procedures, drugs, and diagnoses. Even a basic definition of treatment could require a search for any one of 100 different codes.

A SAS® macro may come to mind, but generalizing the macro to work with different codes and types will allow it to be reused in a variety of different scenarios. We'll look at a number of examples, starting with a single code type and variable. Then we'll consider multiple code variables, multiple code types, and multiple flag variables. We'll show how these macros can be combined and tailored to different data with minimal rework.

Macro flexibility and reusability are also discussed, along with ways to keep our list of medical codes separate from our program. Finally, we'll discuss time-dependent medical codes, codes requiring database lookup, and macro performance.

INTRODUCTION

Medical claims hold a wealth of information, but finding the relevant information in all this data can be overwhelming. Patients frequently have hundreds of claims, claims can have dozens of line items, and each line item can have dozens of medical codes. The cryptic alphabet soup of medical codes can make interpretation difficult.

For example, suppose you're asked to find which patients had an office visit, where an office visit is a claim with any Healthcare Common Procedure Coding System (HCPCS) code in the range:

- 99201 to 99205: New Patient Office Visit
- 99211 to 99215: Established Patient Office Visit

This seems straightforward -- use a DATA step with IF, or PROC SQL with WHERE.

But what if you need to identify patients with prostate cancer? You might consider HCPCS procedures

- 52620 Transurethral resection
- 55810 Prostatectomy, radical

and HCPCS (code level II) drug

- J9217 Leuprolide acetate 7.5mg

and International Classification of Disease (ICD-9) procedures

- 602 Transurethral prostatectomy
- 605 Radical prostatectomy

as well as ICD-9 diagnosis

- 185 Malignant neoplasm of prostate

but not ICD-9 *procedure*

- 185 Surgical correction of prominent ear

Suddenly your SAS code is a jumble of IFs, ANDs, ORs, and WHEREs. A more systematic way of managing this logic is needed.

METHODOLOGY

Suppose we have two datasets in SAS, one named CODES and one named CLAIMS. The CODES dataset contains two types of codes, HCPCS codes and ICD-9 procedure codes. These are represented by orange and blue.

We'll start by flagging our claims orange, for HCPCS. This will involve two steps:

1. Pull the orange codes from CODES using PROC SQL
2. Flag the appropriate observations in CLAIMS using a DATA step

Next, we want to flag our claims blue, for ICD-9. We'll build on the CLAIMS dataset we just flagged, so CLAIMS is already flagged orange. The same two steps will be applied:

3. Pull the blue codes from CODES using PROC SQL
4. Flag the appropriate observations in CLAIMS using a DATA step

These steps can be seen in Figure 1:

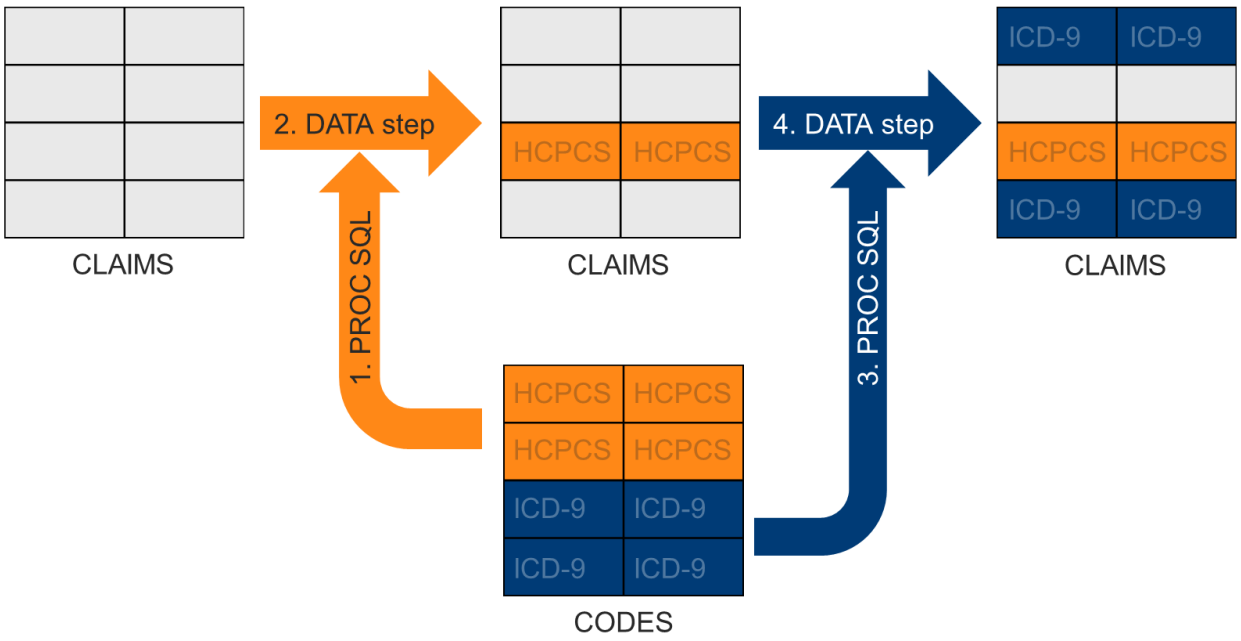

Figure 1. Flagging claims

Because this process is iterative it's an ideal candidate for a macro. Each call to the macro can provide another round of flagging, until finally a fully flagged dataset is complete.

PREPARATION

Before we begin, both the claims and medical codes themselves must be standardized and formatted. The SAS code needed to accomplish these tasks isn't covered here. Instead we'll focus on the form this data needs to take, and the assumptions being made.

CLAIMS

Claims data are assumed to have the following characteristics:

- codes have periods and leading and trailing blanks removed, if present (18.5 becomes 185)

- codes are kept as character variables (leading zeros are retained)
- unpaid, irrelevant, or otherwise invalid claims have been removed
- code variable names match the type of code they represent (hcpcs, icd9)
- multiple variables of the same code type are named using sequential numbers (hcpcs1, hcpcs2)
- boolean flag variables have been created with an underscore, and initialized (_flag=0)
- a patient identifier (id) is present

An example can be seen in the CLAIMS dataset in Table 1. This is a simplification, as you'll often want to keep other variables like claim date, claim id, place of service, and cost of service.

CODE LISTS

Constructing a code list, or map of the code (185) to description (prostate cancer), is a critical step for creating flexible, easy-to-maintain SAS code. It's assumed you already have this list in SAS, like the CODES dataset shown in Table 2.

It's also assumed, like the claims, that the *code* variable is formatted as a character and periods and blanks are removed. The *description* is only for reference and won't be used by the program.

How you establish the CODES dataset is up to you, but PROC IMPORT from a spreadsheet is recommended since:

- the list is usually in a spreadsheet to begin with
- the file can be quickly viewed and edited
- spreadsheets are easily repurposed
- spreadsheets can be shared with non-SAS users

All of these reasons support flexibility and collaboration. Code lists often require validation or revision from multiple subject matter experts, including physicians, specialists, medical coders, or other SAS programmers in the field.

Queries against an existing SAS dataset are also a possibility, and an example using pharmaceuticals in a National Drug Code (NDC) database is covered later. Queries on systems external to SAS (other databases, web services, or APIs) are outside the scope of this paper.

EXAMPLES

EXAMPLE 1: SINGLE CLAIM VARIABLE AND CODE TYPE

The simplest example involves a single claim variable and single code type. Consider the CLAIMS dataset in Table 1, with patient identifier *id*, and HCPCS code *hcpcs1*.

id	hcpcs1	_edvisit
001	99281	0
002	44955	0
003	73610	0
004	99284	0
005	59610	0

Table 1. CLAIMS dataset, before flagging (ex. 1)

Who visited the emergency department? We'll create flag variable *_edvisit* and initialize it to 0. Assume we've constructed the dataset CODES using the HCPCS codes in Table 2.

code	description
99281	ED Visit, minor
99284	ED Visit, urgent, non-life threatening
99285	ED Visit, urgent, life threatening

Table 2. CODES dataset (ex. 1)

Our first step is to comma-separate these codes and combine them into macro variable *&codelist*:

```
proc sql;
select quote(code)
  into :codelist
  separated by "," from CODES;
quit;
```

Next, we'll use a DATA step to flag all observations having a code in *&codelist*: The name of the dataset is kept as CLAIMS to allow for iteration:

```
data CLAIMS;
 set CLAIMS;
  if hcpcs1 in (&codelist)
  then _edvisit=1;
run;
```

The resulting CLAIMS dataset in Table 3 has the value of *_edvisit* flagged as 1 for patients 001 and 004.

id	hcpcs1	_edvisit
001	99281	1
002	44955	0
003	73610	0
004	99284	1
005	59610	0

Table 3. CLAIMS dataset, after flagging (ex. 1)

Whitespace should be scrutinized carefully. If not handled previously as assumed, additional statements will be necessary.

EXAMPLE 2: SINGLE CLAIM VARIABLE, MULTIPLE CODE TYPES

A more complex and more common example involves multiple code types. Suppose we want to identify treatments for prostate cancer. We expand our code list in Table 4 to include ICD-9 procedure codes.

codetype	code	description
HCPCS	52620	Transurethral resection
HCPCS	55810	Prostatectomy, radical
ICD9Px	605	Radical prostatectomy
HCPCS	J9170	Docetaxel 20mg
ICD9Px	9220	Brachytherapy infusion

Table 4. CODES dataset (ex. 2)

An additional variable named *codetype* is required. Note that the code types match the variable names in CLAIMS, as shown in Table 5. This will allow us to pass a single macro argument instead of two.

id	hcpcs1	icd9px1	_prostate
001		9220	0
002		605	0
003		740	0
004	J9170		0
005	99201	991	0

Table 5. CLAIMS dataset, before flagging (ex. 2)

Now we create the macro *%flag* and pass *codetype* as an argument. We enclose both the PROC SQL and DATA steps. The PROC SQL query only returns codes for the code type, and the DATA step only evaluates claim variables of that type:

```
%macro flag(codetype);
 proc sql;
  select quote(code)
   into :codelist
   separated by "," from CODES
   where codetype=&codetype;
 quit;

 data CLAIMS;
  set CLAIMS;
   if &codetype.1 in (&codelist)
    then _prostate=1;
 run;
%mend;
```

To flag the claims we call our macro twice, once for each code type:

```
%flag(hcpcs);
%flag(icd9px);
```

The final dataset appears in Table 6.

id	hcpcs1	icd9px1	_prostate
001		9220	1
002		605	1
003		740	0
004	J9170		1
005	99201	991	0

Table 6. CLAIMS dataset, after flagging (ex. 2)

EXAMPLE 3: SINGLE CLAIM VARIABLE, MULTIPLE CODE TYPES, MULTIPLE FLAGS

There is often a need to search claims for more than one flag. Example 2 identified treatments for prostate cancer, but the specific type of treatment may be needed instead.

The CODES dataset may be expanded to look like the one in Table 7. We've added another variable, *flagtype*, to identify the codes as chemotherapy, radiation, and surgery.

flagtype	codetype	code	description
Surg	HCPCS	52620	Transurethral resection
Surg	HCPCS	55810	Prostatectomy, radical
Surg	ICD9Px	605	Radical prostatectomy
Chemo	HCPCS	J9170	Docetaxel 20mg
Rad	ICD9Px	9220	Brachytherapy infusion
Rad	HCPCS	77401	Radiation delivery

Table 7. CODES dataset (ex. 3)

CLAIMS is also expanded in Table 8. The single flag variable *_prostate* is replaced by three new flag variables: *_chemo*, *_rad*, and *_surg*. These match the values of *flagtype* so that, just like the code types, a single macro variable can reference both the *flagtype* in CODES and the corresponding flag in CLAIMS.

id	hcpcs1	icd9px1	_chemo	_rad	_surg
001		9220	0	0	0
002		605	0	0	0
003		740	0	0	0
004	J9170		0	0	0
005	99201	991	0	0	0

Table 8. CLAIMS dataset, before flagging (ex. 3)

We'll build off the macro from example 2 and add an argument for flagtype:

```
%macro flag(codetype, flagtype);
 proc sql;
  select quote(code)
   into :codelist
   separated by "," from CODES
   where
    codetype=&codetype and
    flagtype=&flagtype;
 quit;

 data CLAIMS;
  set CLAIMS;
   if &codetype.1 in (&codelist)
    then _&flagtype=1;
 run;
%mend;
```

To flag the claims completely we need to call our macro six times, once for each combination of code type and flag type:

```
%flag(hcpcs, surg);   *flag surgery;
%flag(icd9px, surg);

%flag(hcpcs, chemo);  *flag chemotherapy;
%flag(icd9px, chemo);

%flag(hcpcs, rad);    *flag radiation;
%flag(icd9px, rad);
```

Note that SAS macros can call other macros, so another macro could be used for large numbers of calls.

There are no ICD-9 procedure codes for chemotherapy, so this combination will load CLAIMS unnecessarily. A macro %IF statement can be added to check that codes were found in CODES before proceeding to the DATA step. This can be found in the generalized flagging macro in the Appendix.

The final dataset appears in Table 9.

id	hcpcs1	icd9px1	_chemo	_rad	_surg
001		9220	0	1	0
002		605	0	0	1
003		740	0	0	0
004	J9170		1	0	0
005	99201	991	0	0	0

Table 9. CLAIMS dataset, after flagging (ex. 3)

EXAMPLE 4: MULTIPLE CLAIM VARIABLES, MULTIPLE CODE TYPES, MULTIPLE FLAGS

Our examples so far assumed a single code variable, such as *hcpcs1* or *icd9px1*, but this is a simplification. In practice a claim will have many variables for a given code type.

Suppose the CLAIMS dataset in Table 6 contained three ICD-9 procedure variables, as shown in Table 10. Patient 005 received a brachytherapy infusion (9220) along with the outpatient office visit (99201) and injection of a therapeutic substance (991). Since code 9220 is in CODES, patient 005 is now flagged.

id	hcpcs1	icd9px1	icd9px2	icd9px3	_prostate
001		9220			1
002		605			1
003		740			0
004	J9170				1
005	99201	991	9220		1

Table 10. CLAIMS dataset, after flagging (ex. 4)

To accomplish this we'll add a loop to our DATA step that includes all three ICD-9 variables:

```
%macro flag;
 data CLAIMS;
  set CLAIMS;
  %do j= 1 %to 3;
   if icd9px&j in (&codelist)
    then _prostate=1;
  %end;
 run;
%mend;
%flag;
```

We can also build on the previous example by introducing another macro variable, *&codecount*. This will allow us to pass the number of variables to the macro dynamically, without assuming there will always be three. The PROC SQL step remains unchanged:

```
%macro flag(codetype, codecount, flagtype);

 proc sql...

 data CLAIMS;
   set CLAIMS;
     %do j= 1 %to &codecount;
      if &codetype.&j in (&codelist)
      then _&flagtype=1;
     %end;
 run;

%mend;
```

To flag the claims we still call our macro six times, but pass *&codecount* to tell the macro how many loops are required for each variable:

```
%flag(hcpcs, 1, surg);   *flag surgery;
%flag(icd9px, 3, surg);

%flag(hcpcs, 1, chemo);  *flag chemotherapy;
%flag(icd9px, 3, chemo);

%flag(hcpcs, 1, rad);    *flag radiation;
%flag(icd9px, 3, rad);
```

Again, another macro could be used for large numbers of calls. Macro variables could also be used to set the different values of *&codecount* so that they aren't hard-coded.

It's important to note that codes often have different meanings based on their position on a claim. Diagnosis codes, for example, are usually coded with the principal diagnosis in the first variable. Second or higher variable positions only indicate coexisting or relevant conditions, not the primary reason for the claim. Make sure variables are relevant to your analysis before you use them.

EXAMPLE 5: TIME-DEPENDENT CODES

Medical codes frequently change. ICD-10, for example, offers an expanded code list and different notation from ICD-9. Older codes are often retired, but if they're reused it makes flagging your claims time-dependent.

One example is the Diagnosis Related Group (DRG) codes Medicare uses to bundle inpatient services for reimbursement. The coding system was revised significantly in October 2007, so most claims flagged using DRG codes need to take dates into account.

Suppose we wanted to examine cancer treatments using DRG codes. Codes for mastectomy (257-259), rectal resection (146-147) and prostatectomy (306-307) are only valid before October 2007. After this time they indicate pacemaker replacement, toe amputation, ear/nose/throat malignancy, and cardiac disorder.

The converse will also occur. Codes for rectal resection (332-334) are only valid starting October 2007. Before October 2007 they were used for kidney, urinary tract, and pelvic procedures.

There are different ways to incorporate these dates into our macro. A brute-force solution to this problem is to query for these exceptions and set the flag back to 0:

```
proc sql;
  update claims
  set _surg=0
   where
    (drg1 in (
        '146','147',
        '257','258','259','260',
        '306','307') and
    (claimdate >= '01OCT2007'd))
   or
    (drg1 in ('332','333','334') and
    (claimdate < '01OCT2007'd));
quit;
```

Unfortunately this contradicts the reasons for keeping our code list separate from our SAS code. These rules will be messy, and the codes difficult to find, maintain, and audit.

Another option is to add CodeFrom and CodeTo dates to our code list to specify valid date ranges for each code. These dates can then be passed to the macro. A sample CODES dataset for rectal resection is in Table 11.

codetype	code	CodeFrom	CodeTo	description
DRG	146	01OCT05	30SEP07	rectal resection
DRG	147	01OCT05	30SEP07	rectal resection
DRG	332	01OCT07	30SEP12	rectal resection
DRG	333	01OCT07	30SEP12	rectal resection
DRG	334	01OCT07	30SEP12	rectal resection

Table 11. CODES dataset (ex. 5)

This complicates the syntax since each code must now have a dedicated macro variable. We can no longer use a single, comma-separated variable *&codelist:*

```
proc sql;
  select code, codefrom, codeto into
      :code1-:code999,
      :from1-:from999,
      :to1-:to999
   from CODES
   where codetype="DRG";
quit;
```

The value of 999 is chosen arbitrarily and represents an upper bound for the maximum number of codes the macro will support. SAS only creates these macro variables as needed, so choosing a sufficiently large number will not impact performance.

The DATA step also increases in complexity. Another layer of looping is required, once for each code. The number of loops required can be found using *sqlObs*, a system-generated count of observations returned by PROC SQL. This will equal the number of codes found, so it can be used to end the loop:

```
data CLAIMS;
  set CLAIMS;
   %do j =1 %to &sqlObs;
        if (
          drg1 = "&&code&j" and
          ("&&from&j."d <= claimdate <= "&&to&j."d)
          )
          then _surg=1;
   %end;
run;
```

The date of the claim, *claimdate*, is checked against the CodeFrom and CodeTo date in CODES. The claim will only be flagged if the claim lands within the date range and only when the code matches.

This level of precision can also help detect claims that may be miscoded outside the valid range of the code. However, if there are only a handful of codes you wish to examine for time-dependency, a PROC SQL with UPDATE will still be more computationally efficient, even if the method is not ideal.

PERFORMANCE AND OPTIMIZATION

Macro performance will depend on the SAS server(s) and your configuration, but also:

- the number of codes, code types, and flags
- the number of claim variables evaluated
- the number of observations
- additional claim data, and size of the dataset
- dedicated code variables (*:&code1-:&code999*) vs. a single code variable (*&codelist*)

Performance generally ranges between 10,000-1,000,000 observations flagged per second on a single Windows 7 Enterprise server running SAS 9.3 (64-bit). This performance is for one macro call, so six macro calls will mean six passes through the data, and it will take six times as long.

Macro performance can be optimized by setting COMPRESS=YES for all DATA and PROC SQL steps. Compression almost always yields some benefit for claims data since high-numbered code variables (like the 20th procedure code) are mostly empty. Additional claim variables may be dropped from the macro to reduce file size as well, but the benefit may be limited if they need to be joined later.

More code variables will increase the time it takes the macro to flag since there will be more IF statements to evaluate. No significant difference was observed when increasing the number of diagnosis variables from 1 to 3 variables, but 15 variables reduced performance by about 30%.

This performance reduction is compounded when evaluating claim variables code-by-code versus evaluating a comma-separated list of codes. The number of IF statements will always be the number of codes found in CODES (*&sqlObs*) multiplied by the number of code variables in CLAIMS. For example, a list of 1,000 procedure codes evaluated across ten procedure variables means the macro must evaluate 10,000 IF statements for each observation!

Because of this, a comma-separated *&codelist* variable should be used whenever possible. Time-dependent codes should be evaluated separately unless performance is not a concern.

LIMITATIONS

The DATA step in our *%flag* macro always used the dataset name CLAIMS. This allowed the flagging process to be iterative, so multiple calls to the macro will build on flags set in earlier calls.

This is not always optimal. If a claim is already flagged there is no reason to keep flagging it, so the claim should be excluded. However this is not true for multiple flags, so to keep the macro flexible the same claims are iterated each time. If there's only one flag, a WHERE= clause could be used to only bring unflagged claims into the macro.

Determining which codes were used can be difficult. Extracting the flagged claims will be a start, but multiple code variables will make PROC FREQ alone insufficient. One solution is to log the code into a separate variable (or variables) when the flag is set.

SAS macro variables truncate at 4,096 bytes by default. This may result in errors when more than 1,000 codes are loaded into a single *&codelist* variable. There are multiple workarounds to this problem:

1. Expand the limit to 65,534 bytes using the SAS option MVARSIZE
2. Segment the codes into smaller types or flags
3. Use dedicated macro variables (at the cost of performance)

CONCLUSION

Claims data rarely come with variables ready for analysis. Creating these variables can be challenging, both technically and logistically. Analyzing a single disease may involve hundreds of medical codes, and ensuring this code list is accurate for the analysis may be difficult in its own right.

Even after a code list is established, programming can quickly become unmanageable when taking into account the number of code variables, code types, flags, and time dependencies. Hard-coding rules and exceptions make it difficult to add new algorithms and troubleshoot errors.

A flagging macro can help with all these issues. First, it allows the codes to be imported dynamically so that the code list remains separate from the program. This improves flexibility and reusability of the code list itself, and facilitates code list collaboration and transparency.

Second, the macro remains unchanged for any code type or flag, so the method for flagging can be standardized and troubleshooting simplified.

Finally, since both the code list and the macro can be separated from the program, the entire flagging process can be replicated across projects with minimal effort. Code lists can be quickly updated and macro arguments can be easily adjusted to accommodate new claims. Writing a new program to flag claims for each new project is not necessary.

APPENDIX

CODE LOOKUPS

Sometimes codes will require a lookup to another data source, such as a prescription drug National Drug Code (NDC) database. In this example we assume this data source is already in SAS, in a dataset called DRUGLOOKUP as shown in **Error! Reference source not found.**:

drugname	code	dose
ACETYLCYSTEINE 10%	00517750425	1g
ACETYLCYSTEINE 20%	54868567001	1g
⋮	⋮	⋮
Zyvox	00009514001	200mg

Table 12. DRUGLOOKUP dataset

Suppose you want to use DRUGLOOKUP to find which patients were prescribed antibiotics for tuberculosis. You have a list of about 15 different antibiotics, but DRUGLOOKUP could contain hundreds of thousands of entries for every drug, formulation, and dose. Finding these manually could be cumbersome.

A better option is to create a query that performs the lookup for you, based on your list of drugs. First, the list of antibiotics names is loaded into the SAS dataset DRUGLIST in Table 13:

drug
Ethambutol
Amikacin
⋮
Tobramycin

Table 13. DRUGLIST dataset

Now we construct a query on DRUGLOOKUP using a query on the drug names in DRUGLIST. Since the names may not be an exact match, we use the keyword CONTAINS and separate each drug with the text "or drugname contains". This becomes the text in *&drugquery* and gets substituted in the second query:

```
proc sql;
 select drug from DRUGLIST into drugquery
          separated by "or drugname contains";
 create table CODES as select * from DRUGLOOKUP
          where drugname contains &drugquery;
quit;
```

The second query will read "…where drugname contains Ethambutol or Amikacin or..."

The CODES dataset is in Table 14, where the NDC code for any drug in the list is now saved.

drugname	code	dose
Ethambutol (Myambutol)	00555092302	400mg
ETHAMBUTOL	49349006502	100mg
amikacin sulf.	00703903203	500mg
...	...	...

Table 14. Partial CODES dataset from DRUGLOOKUP

GENERALIZED FLAGGING MACRO

This macro brings together the code from all the examples. It assumes you have a code list named CODES with variables code, codetype, codefrom, codeto, and flagtype. It uses dedicated macro variables, one for each code, up to 999 codes. An argument named *flagset* is used for the dataset name.

Usage*: %flag(icd9px, 3, Chemo, CLAIMS)*

```
%macro flag(codetype, codecount, flagtype, flagset);

*load codes and valid dates for this flag type and code type;

proc sql noprint;
 select code, codefrom, codeto into
  :code1-:code999,
  :from1-:from999,
  :to1-:to999
 from CODES
 where flagtype="&flagtype" and codetype="&codetype";
quit;

*only load the claims if codes were returned;
%if &sqlObs > 0 %then %do;
%put ***** Checking claims using a list of &sqlObs &codetype &flagtype
codes;

*load the claims;
data &flagset (compress=yes drop=codecounter claimcounter _claimcounter);
  set &flagset end=end;

  retain codecounter;    *keep track of the number of codes that match;
  retain claimcounter;   *keep track of the number of claims that match;
  _claimcounter = 0;     *flag used by the claimcounter;

  *loop through each code and each variable;
  *check if the code matches, and if it has time-dependency;
  *if the code matches flag the claim and update the counters;
   %do j= 1 %to &codecount;
     %do k= 1 %to &sqlObs;
        if (
               (&codetype&j = "&&code&k") and
               (
               ("&&from&k."d eq .) or
               (("&&from&k."d ne .) and
                      ("&&from&k."d <= claimdate <= "&&to&k."d))
               )
         )
         then do;
              _&flagtype=1;
              if _claimcounter = 0 then _claimcounter = 1;
              if codecounter eq . then codecounter = 1;
                 else codecounter=codecounter+1;
         end;
   %end;
  %end;

*increment the claim counter before loading the next observation;
if _claimcounter = 1 then do;
```

```
   if claimcounter eq . then claimcounter=1;
   else claimcounter=claimcounter+1;
end;

*write the number of claims and codes to the log if this observation is the
last;
if end=1 then do;
   if codecounter ne . then
    put "***** " codecounter "&codetype &flagtype codes found";
    else put "WARNING: No &codetype &flagtype codes were found in the
claims";
   if claimcounter ne . then
    put "***** " claimcounter "&codetype &flagtype claims flagged";
end;

run;

*this ends the loop if codes were returned from the code list;
*if no codes were in the code list, log a warning;

%end;

%else %put WARNING: No &codetype &flagtype codes are in the code list;

*log a final count of the claims flagged;

proc sql noprint;
  select count(*) into :claimsum from &flagset where _&flagtype=1;
  %put ***** %cmpres(&claimsum) claims are currently flagged for &flagtype;
quit;

%mend flag;
```

ACKNOWLEDGEMENTS

Thanks to Elissa Brown, Catherine Fedorenko, Barbara Okerson, and Jesse Plascak for their help with this paper.

CONTACT INFORMATION

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

Andy Karnopp
Fred Hutchinson Cancer Research Center
akarnopp@fredhutch.org
github.com/andykarnopp/SAS