MWSUG 2016 — Paper TT03

Calculating Cardinality Ratio in Two Steps

Ronald J. Fehd, Stakana Analytics

Abstract

Description: The cardinality of a set is the number of elements in the set. The cardinality of a SAS® software data set is the number of observations of the data set, n-obs. The cardinality of a variable in a data set is the number of distinct values (levels) of the variable, n-levels. The cardinality ratio of a variable is n-levels / n-obs; the range of this value is from zero to one.

Previous algorithms combined output data sets from the frequency and contents procedures in a data step. This algorithm reduces multiple frequency procedure steps to a single call, and uses scl functions to fetch contents information in the second data step.

The output data set, a dimension table of the list of data set variable names, has variable cr-type whose values are in (few, many, unique); this variable identifies the three main types of variables in a data set, few is discrete, many is continuous, and unique is a row-identifier.

Purpose: The purpose of this paper is to provide a general-purpose program, ml-namex.sas, which provides enhanced information about the variables in a data set. The author uses this list-processing program in Fast Data Review, Exploratory Data Analysis (EDA) and in Test-Driven Development (TDD), a discripline of Agile and Extreme Programming.

Audience: programmers, data managers, database administrators

Keywords: SAS component language (scl) functions: attrn (nobs, nvar), close, open, varfmt, varinfmt, varlabel, varlength, varnum, vartype; frequency procedure, nlevels option

In this paper

Introduction

Overview

The purpose of this paper is to show an optimized algorithm for the calculations of cardinality ratio and cardinality type which reduces the number of steps from *O(n-vars)* to *O(1)* This subroutine is designed to be used by later list processing programs which provide a frequency of all the discrete variables and a summary of each of the analysis variables.

See SmryEachVar programs on page 17.

This section contains these topics.

- cardinality
- database terminology
- goal overview
- previous algorithm
- issues
- new algorithm
- output

cardinality

This table shows the relationship of cardinality to cardinality-ratio and cardinality-type.

cardinality:	number of elements of a set of an array, the dimension	*n-rows(table)*
levels:	number of values in a column	*n-levels(column)*
cardinality ratio (cr):	$\frac{n\text{–}levels(column)}{n\text{–}rows(table)}$	range: $(0:1]$

		cardinality-type			
		few		many	unique
values:	0	. . . .	mean(cr)	. . . .	1
		distinct		continuous	
		by ∣ class var		analysis var	

Notes: The mean of cardinality ratio, *mean(cr)*, is used to separate category *few* from *many*; this statistic is theoretical and has been validated as empirically true from the test suite of the `sashelp` library.

database terminology

The goal of this program is to produce a database *dimension table*, the essential columns of which are a *primary key*, the row number, and information in three sets: 1. member name, 2. cardinality information, and 3. variable information.
This table, from Fehd [3], shows which database tables contain which columns.

columns		tables			
				snapshots	
name	type	dimension	fact	periodic	accumulating
keys:					
primary	integer	row-number[1]	row-id		entity-id
foreign[2]	integer		●		
composite[3]				●	
boolean	integer				●
facts	real		●	statistic	sum
text	char	information			

Notes:
[1] a *natural key* is a row-number in (1:n-rows)
[2] foreign key is the primary key of a dimension table
[3] composite key: combination of foreign keys

goal overview

When given a new data set, the first task is to place each variable into one of three categories: is it the row identifier? a classification variable? or an analysis variable?

	common	var type		cr-
task	name	c	n	type
data review	row-id	c		unique
	row-number		n	unique
by|classification vars	discrete	c	n	few
analysis vars	continuous		n	many
problem: empty	n-levels=1			

previous algorithm

This is pseudo-code of the previous algorithm first shown in the SmryEachVar suite, 2008: Fehd [2], and further developed in 2008: Fehd [3], 2013: Fehd [6], and 2014: Fehd [8].

```
1. make list of var names                    contents/sql out=list-names

2. for each variable
      make list of values                  proc freq out=freq-of-variable
      save n-obs (n-levels)                                    data n-obs
                                               proc append to list-of-n-obs

3. join list-names with list-of-n-obs                     data list-names
      calculate cardinality ratio

4. calculate mean of cardinality ratio                       proc summary

5. calculate cardinality type (cr-type)                   data list-names
```

issues

The new algorithm addresses these issues.

type : variable type has three sets of values
contents: (1=n, 2=c); scl: (C,N); sql: (char,num)
choice: `$char1` in (c,n) to conform to style guide: use lowcase

attributes : data set attributes n-obs and n-vars are in local symbol table
copy to macro variables in the global symbol table with scl `attrn(...)`
for use in array allocation in second step

calculation : mean of cardinality ratio previously calculated by summary procedure
is calculated in an array

side effects : enhancements for next step:
macro variables with lists of *few*, and *many* are saved
and written to log

new algorithm

- step 1.1: make data set `list-names` with `proc freq nlevels`
- step 1.2: copy data set attributes *n-obs* and *n-vars*
  to macro variables in global symbol table
  for use in array allocation and calculation
- step 2: data structure: attribute, array
  loop: calculate cardinality ratio
  calculate mean(cardinality ratio)
  loop: read `list-names`,
    fetch var-name information with scl functions
    calculate cardinality type
    make lists from cardinality type

output

This is the data structure of the dimension table, list-names.

```
create table WORK.LIST_NAMES
  (label='memname=class,obs=19,vars=5')
  (memname    char(32),
   varnum     num      label='var num',
   cr_type    char(10) label='card. ratio type',
   card_ratio num      label='card. ratio',
   n_levels   num      label='n-levels nobs=19',
   name       char(32) label='name',
   type       char(1),
   length     num,
   format     char(49),
   informat   char(49),
   label      char(256)
```

This is an example report for the data set `sashelp.class`.

```
         var             card_      n_
memname  num  cr_type    ratio  levels  name    type  length etc.
-------  ---  -------  -------  ------  ------  ----  ------ ---
 class     1  .unique        1      19  Name     c       8
 class     2  few      0.10526       2  Sex      c       1
 class     3  few      0.31579       6  Age      n       8
 class     4  many     0.89474      17  Height   n       8
 class     5  many     0.78947      15  Weight   n       8
```

Research for Data Structure: Contents, SQL, SCL, N-levels

Proc Contents

Overview

This section contains these topics.

- contents program
- describe contents output
- contents listing
- contents output data set

contents program

```
%let libname = sashelp;
%let memname = class;
PROC contents data =  &libname..&memname
              out  =   contents;
PROC sql;     describe table &syslast;
              quit;
PROC print    data =         &syslast noobs;
              var      varnum name type length;
```

describe contents output

```
create table WORK.CONTENTS
  (LIBNAME  char(8)   label='Library Name',
   MEMNAME  char(32)  label='Library Member Name',
   NAME     char(32)  label='Variable Name',
   TYPE     num       label='Variable Type',        <---<<<
   LENGTH   num       label='Variable Length',
   VARNUM   num       label='Variable Number',
   LABEL    char(256) label='Variable Label',
   FORMAT   char(32)  label='Variable Format',      <---<<<
```

Notes: contents.type is numeric; length of format is $char32.

contents listing

```
Alphabetic List of Variables and Attributes
#    Variable    Type    Len
-    --------    ----    ---
3    Age         Num       8
4    Height      Num       8
1    Name        Char      8
2    Sex         Char      1
5    Weight      Num       8
```

Notes: #: variable number is the row number;
the contents listing is alphabetically ordered by `Variable` (name)

contents output data set

```
VARNUM    NAME      TYPE*   LENGTH
------    ----      ----    ------
   3      Age         1        8
   4      Height      1        8
   1      Name        2        8
...
```

! → * contents.type in (1=Num,2=Char).

Proc SQL

Overview

This section contains these topics.

- sql program
- describe dictionary.columns
- describe sashelp.class
- sql report

sql program

```
%let libname = sashelp;
%let memname = class;
PROC sql; describe table dictionary.columns;
          describe table &libname..&memname;
          select   memname, varnum, name, type, length
          from     dictionary.columns
          where        libname eq "%upcase(&libname)"
                   and memname eq "%upcase(&memname)";
          quit;
```

describe dictionary.columns

```
create table DICTIONARY.COLUMNS
  (libname char(8)   label='Library Name',
   memname char(32)  label='Member Name',
   name    char(32)  label='Column Name',     unique: row-id,     primary key
   type    char(4)   label='Column Type',     <---<<<
   length  num       label='Column Length',
   varnum  num       label='Column Number',   unique: row-number, primary key
   label   char(256) label='Column Label',
   format  char(49)  label='Column Format',   <---<<<
```

Notes: type is $char4; length of format is $char49.;
both `name` and `varnum` are unique, and therefore each is a primary key.

describe sashelp.class

```
create table SASHELP.CLASS( label='Student Data' )
  (Name   char(8),
   Sex    char(1),
   Age    num,
   Height num,
   Weight num
```

sql report

```
         Column
Member   Number    Column   Column   Column
Name     in Table  Name     Type     Length
------   --------  ------   ------   ------
CLASS           1  Name     char          8
CLASS           2  Sex      char          1
CLASS           3  Age      num           8
CLASS           4  Height   num           8
CLASS           5  Weight   num           8
```

Notes: sql output is ordered by `Column Number (varnum)`; type is in (char,num);
`varnum` is a *natural key*, its values are in *(1:n-vars)*.

SAS Component Language (scl)

Overview

This section contains these topics.

- scl program
- scl output data set

scl program

```
%let libname = sashelp;
%let memname = class;
DATA list_names_scl;
     attrib varnum length =   8
            name   length = $32
            type   length = $ 1;
     drop   _:; *** _temporary vars;

*see the varnum function for this example;
_dsid   = open ("&libname..&memname");
_n_vars = attrn(_dsid,’nvars’);

do _i = 1 to _n_vars;
   name   = varname(_dsid,_i);
   varnum = varnum (_dsid,name);
   type   = vartype(_dsid,_i);
   output;
   end;
_rc = close (_dsid);
stop;
run;
PROC print data = &syslast;
```

Notes: The scl functions `open` and `close` allow access to the other scl functions `varname`, `varnum`, and `vartype`;
`varname` and `vartype` depend on the loop index variable, `_i`,
while `varnum`, which is equal to the loop index variable, depends on the variable `name`.

scl output data set

```
varnum    name      type
------    ------    ---
   1      Name       C
   2      Sex        C
   3      Age        N
   4      Height     N
   5      Weight     N
```

Notes: Scl.type is in upcase(C,N).

Proc Freq N-levels

Overview

This section contains these topics.

- proc freq program
- describe table sashelp.air
- data structure of freq nlevels

proc freq program

```
%let data = sashelp.air;
PROC freq  data    = &data
           nlevels ;
           ods       output
           nlevels = list_names_nlevels;
PROC sql;  describe table &data;
           describe table &syslast;
           quit;
```

describe table sashelp.air

```
create table SASHELP.AIR
  (label='airline data (monthly: JAN49-DEC60)')
  (DATE num format=MONYY.,
   AIR  num label='international airline travel (thousands)'
```

Notes: `sashelp.air` has variables with labels.

data structure of freq nlevels

```
create table WORK.LIST_NAMES_NLEVELS
  (TableVar      char(4)  label='Table Variable',
   TableVarLabel char(40) label='Table Variable Label',
   NLevels       num      format=BEST8.
                          label='Number of Levels'
```

Notes: Variable names, `TableVar, NLevels,` are different from the contents and sql procedures, their variable name for the *row-identifier* is `name`.

Research Summary

Data Structure Issues

The above research highlights these issues in the new algorithm.

order : `memname, varnum` *(row-number)*, `name, type, length,` etc.

freq n-levels : variable names `TableVar` and `NLevels`
to be renamed to `name, n_levels`

type : scl lowcase(c,n)

length : of format and informat: sql $char49.

Explanation

Overview

The explanation of the program contains this list of topics.
The program listing of `ml-namex.sas` is on page 13.

autoexec

All programs shown in this paper depend on an *autoexec* which has two *filerefs*: one named *project* for the folder containing the programs shown here, and the other named *site_inc* for the folder containing the program `ml-namex.sas`.

```
** name:  autoexec.sas;
filename  project  ’.’; *** contains programs listed here;
*author’s useage: folder containing ml-namex.sas;
*filename site_inc ’<...>\SAS-site\includes’;
*for your testing:;
filename  site_inc ’.’;*folder containing ml-namex.sas;
```

testing program

This is the program used to test `ml-namex.sas`.

```
options mprint source2;
%let libname = sashelp;
%let memname = class;
%include site_inc(ml-namex);
proc print data = &syslast   noobs;
proc sql; describe table &syslast;
          quit;
```

proc freq

This is step 1.1. The frequency output data set from a data set with variable labels contains extra variables; keep the two desired variables and rename them.

```
PROC freq data = &libname..&memname    nlevels;
     ods  output nlevels=
     list_names(keep  = tablevar       nlevels
                rename=(tablevar=name  nlevels= n_levels));
```

copy n-obs, n-vars

This is step 1.2. These macro variable assignment statements copy the data set attributes, *nobs* and *nvar* into the global symbol table for use in the next step. N-obs is used as denominator of the calculation of cardinality ratio; n-vars is used for the dimension of the array of variable names.

```
****  copy     local n-obs and n-vars to global symbol table;
%let _dsid   = %sysfunc(open (&libname..&memname,i));
%let _n_obs  = %sysfunc(attrn(&_dsid,nobs));
%let _n_vars = %sysfunc(attrn(&_dsid,nvar));
%let _rc     = %sysfunc(close(&_dsid));
```

data structure

The data structure contains three sets of information:
1. *memname*, the relation to other members in the *libref*,
2. information about cardinality ratio,
3. information about variable.

```
DATA &syslast;
     attrib memname     length = $32
            varnum      length =   8 label = 'var num'
            cr_type     length = $  %length(n-levels=1)  %*** label of 1st cr-type;
                        label  =    'card. ratio type'
            card_ratio  length =   8     %*range=(0:1];
                        format =    bestd7.5
                        label  =    'card. ratio'
            n_levels    length =   8
                        label  =    "n-levels nobs=&_n_obs"
            name        length = $32 label = 'name'
```

array of cardinality ratios

In the previous algorithm cardinality ratio was calculated in one data step and the summary procedure was used to calculate the mean; this can be accomplished in an array.

! → Note the use of the global macro variables, `n_vars` and `n_obs`.

```
array  _cr(&_n_vars);                               *** <---<<< global n_vars;
*...;
** loop: for each row, calculate cardinality ratio;
do _i = 1 to dim(_cr);
   set &syslast   (keep = n_levels)   point = _i;
   _cr(_i) = n_levels  /  &_n_obs;                  *** <---<<< global n_obs;
   end;

***** calculate mean for select card_ratio to cr_type;
_mean_cr = mean(of _cr(*));
```

read data structure

The scl function `open` make the data structure functions `varnum` and `vartype` available. This data-set-reading loop performs two tasks:
1. read the frequency output data set variables: name and n-levels;
2. read the variable information: `varnum, type`, etc.

```
** read syslast(name n_levels), fetch data structure info;
_dsid = open("&libname..&memname");
do _i = 1 to dim(_cr);
   set &syslast   point = _i;
   varnum     =         varnum (_dsid,name);
   type       = lowcase(vartype(_dsid,varnum));
```

Notes: The lookup of `varnum` is based on the *primary-key* of the frequency-n-levels table which is the variable `name`. In fact the index variable `_i` is the same as `varnum`, but that is an assumption, which is the reason that the lookup uses the variable `name`.

make cr-type

Cardinality-type is assigned within the data-set-reading loop.

```
   card_ratio = _cr(_i);
   select;
     when(n_levels   eq 1       ) cr_type = ’n-levels=1’;
     when(card_ratio eq 1       ) cr_type = ’.unique’;
     when(card_ratio gt _mean_cr) cr_type = ’many’;
     otherwise                    cr_type = ’few’;
     end;
```

Notes: Category *n-levels=1* identifies a useless variable with only one value. Category *unique* has a dot in front of the value which places it at the top of an ordered listing.

robust length of lists

The data structure includes temporary variables for lists of variable names. The maximum length of each of these variables is *(32+1)*n-vars*.
The maximum length of a character variable is $2^{15} - 1 = 32767$.
For the case where a data set contains more than $\frac{32767}{33} = 992$ variables, the allocation of the length of these variables fails. This problem is solved by reducing the length to 32767 in the extreme case.

```
     %*** _temp vars for macro variables;
     %*** 33: length(var-name)=32 +1 for delimiter;
     %let _max_length_list = %sysfunc(min(
                        %eval(33*&_n_vars),32767));
           _list_few    length = $&_max_length_list
```

make lists

This paragraph creates global macro variables of lists of the variables in the *few* and *many* categories for use by later list-processing routines.

```
   ** make list_many_c, list_many_n, for mvars;
   if      cr_type eq ’many’ then do;
      if      type = ’c’ then
           _list_many_c = catx(’ ’,_list_many_c,name);
      else if type = ’n’ then
           _list_many_n = catx(’ ’,_list_many_n,name);
      end;
```

make lengths

This paragraph creates global macro variables of lengths of the variable in the *few* category for use by later list-processing routines.

```
      if type eq ’c’ then do;
           _max_length = max(_max_length,length);
           _length_few = sum(_length_few,length);
           end;
      else _length_few = sum(_length_few,%length(&_n_obs)); *<---<<< global n_obs;
      *** add one for delimiter = space;
      _length_few = sum(1,_length_few);
```

make list-few

This paragraph creates a global macro variable of the variable names in the *few* category in sorted order for use by later list-processing routines. This listing has these paragraphs:

1. allocation of the array, the length of the item is the the number of digits in the macro variable `n-obs` plus one for the delimiter and 32 for the length of the variable name
2. assignment of value: `n-levels` with leading zeroes, colon, and variable name
3. sorting the array into `n-levels` order
4. loop to copy each variable name into the list

```
*1   *****  _lfs: list-few-sorted value=000:name;
     array  _lfs(&_n_vars) $%eval(%length(&_n_obs)+33);
*2 ...;
   else if cr_type eq ’few’  then do;
      **** save for sort: lfs(i)= ’000:name’;
      _lfs(_i) = catx(’:’,put(n_levels
                             ,z%length(&_n_obs).),name);
*3 ...;
**** loop: make list-few ordered by n-levels;
call sortc(of _lfs(*));
*4;
do _i = 1 to dim(_lfs);
   ****        _lfs(_i)=000:name;
   name = scan(_lfs(_i),-1,’:’);
   _list_few = catx(’ ’,_list_few,name);
   end;
```

make macro variables

The `call symput` function is used to create the set of macro variables which are used by succeeding routines.

```
call symputx(’_list_few’    ,_list_few   );
call symputx(’_list_many_c’ ,_list_many_c);
call symputx(’_list_many_n’ ,_list_many_n);
```

echo macro variables

This information is written to the log at the end of the subroutine.

```
%put echo: &=memname &=_n_obs &=_n_vars;
%put info: &=_list_few;
%put info: &=_list_many_c;
%put trace: ml-namex make-list-names-cr ending;
```

result:

```
echo: MEMNAME=class _N_OBS=19 _N_VARS=5
info: _LIST_FEW=Sex Age
info: _LIST_MANY_N=Height Weight
trace: ml-namex make-list-names-cr ending
```

Program Listings

Notes: Programs that use this subroutine are on page 16.

ml-namex.sas

This is the listing of the program `ml-namex.sas`.

```
%put trace: ml-namex make-list-names-cr beginning;
%put echo parameters: &=libname &=memname;
 /*    name: ...\SAS-site\includes\ml-namex.sas
     author: Ronald J. Fehd 2016
------------------------------------------------------
Summary:  description: make list of variable names,
                       cardinality ratio and card-type
              purpose: for list processing routines
          --------------------------------------------
Contexts: program group: procedure returns data set
          program  type: subroutine
          SAS      type: parameterized include
          uses routines: n/a
------------------------------------------------------
Specifications: input  : macro variables
                         libname: libref
                         memname: data set name
                process: proc freq n-levels
                         copy memname.nobs, .nvars
                               from local to global
                         data: read var information
                               with scl functions
                         calculate cardinality ratio
                               and card-ratio-type
                output : list_names, a dimension table
                         macro-variables: _length_few
                         _list_few    _list_many_c
                         _list_unique _list_many_n
                         _max_length_c
                           for data structure attrib
                           valu_c length=$&_max_length_c
note: output is ordered by varnum
--------------------------------------------------------
usage:   autoexec:
filename site_inc ’<...\SAS-site\includes>’;
- - -  ml-names-x-test.sas - - -
%let libname = sashelp;
%let memname = class;
%include site_inc(ml-namex);
proc sql; describe table list_names;
          quit;
proc print data = list_names;
---------------------------------------------------------
this is a Derivative Work of the SmryEachVar suite
www.sascommunity.org/wiki/SmryEachVar_A_Data_Review_Suite
this program is available on:
http://www.sascommunity.org/wiki/Making_Lists
http://www.mwsug.org/2016-proceedings.html paper TT03
it is designed to work with parameterized includes:
CxInclude, and SmryHuge; macros: CallMacro and CallText
-------------------------------------------------------
date-time: 2016-08-24 4:00:50 PM
word count        words:  734
                  lines:  195
characters(no   spaces): 5410
characters(with spaces): 7893
**** ..................... */
**** make dimension table: name + n-levels;
ODS  exclude all; * noprint;
PROC freq data   = &libname..&memname nlevels;
     ods  output
          nlevels= list_names
         (keep   = tablevar           nlevels
          rename =(tablevar= name     nlevels=n_levels));
run;
ODS select all;
```

```
****  copy      n-obs, n-vars from local to global;
%let _dsid   = %sysfunc(open (&libname..&memname,i));
%let _n_obs  = %sysfunc(attrn(&_dsid,nobs));
%let _n_vars = %sysfunc(attrn(&_dsid,nvar));
%let _rc     = %sysfunc(close(&_dsid));
%symdel _dsid _rc;

**** enhance dimension table with variable information,
     cardinality ratio, card-ratio-type: cr-type;
DATA &syslast(label=
    "memname=&memname,obs=&_n_obs,vars=&_n_vars");
     attrib memname     length = $32
         %* primary key;
            varnum      length =   8 label = ’var num’
         %* cardinality information;
            cr_type     length = $  %length(n-levels=1)
                        label  =    ’card. ratio type’
            card_ratio  length =   8     %*range=(0:1];
                        format =    bestd7.5
                        label  =    ’card. ratio’
            n_levels    length =   8
                        label  =    "n-levels nobs=&_n_obs"
         %* variable information;
            name        length = $32 label = ’name’
            type        length = $ 1 %*range:(c,n) from scl;
            length      length =   8
            format      length = $49
          informat      length = $49
            label       length =$256
           _length_few  length =   4
     %*** _temp vars for macro variables of lists;
     %*** 33: length(var-name)=32 +1 for delimiter;
     %let _max_length_list = %sysfunc(min(
                        %eval(33*&_n_vars),32767));
           _list_few    length = $&_max_length_list
           _list_many_c length = $&_max_length_list
           _list_many_n length = $&_max_length_list
           _list_unique length = $&_max_length_list;
          %put echo calculation: &=_max_length_list;
     %symdel _max_length_list;
     array  _cr(&_n_vars);
     *****  _lfs: list-few-sorted value=000:name;
     array  _lfs(&_n_vars) $%eval(%length(&_n_obs)+33);
     drop   _:;            * _temporary variables;
     retain memname        "&memname"
            _length_few 0  _max_length 1
            _testing       %eval(
             %sysfunc(getoption(mprint )) eq MPRINT
         and %sysfunc(getoption(source2)) eq SOURCE2);

** loop: for each row, calculate cardinality ratio;
do _i = 1 to dim(_cr);
   set &syslast   (keep = n_levels)   point = _i;
   _cr(_i) = n_levels / &_n_obs;
   end;

***** calculate mean for select card_ratio to cr_type;
_mean_cr = mean(of _cr(*));

*****   loop: read syslast(name n_levels), fetch var info;
_dsid = open("&libname..&memname");** for scl functions;
do _i = 1 to dim(_cr);
   set &syslast   point = _i; *** primary-key=name;
   if _testing then putlog name= n_levels=;
   varnum     =         varnum   (_dsid,name  );
   type       = lowcase(vartype  (_dsid,varnum));
   length     =         varlength(_dsid,varnum);
   format     =         varfmt   (_dsid,varnum);
   informat   =         varinfmt (_dsid,varnum);
   label      =         varlabel (_dsid,varnum);
   card_ratio = _cr(_i);
   select;
     when(n_levels   eq 1) cr_type = ’n-levels=1’;
     when(card_ratio eq 1) cr_type = ’.unique’;%*row-id;
     when(card_ratio gt
                 _mean_cr) cr_type = ’many’; %*analysis;
     otherwise             cr_type = ’few’;  %*by|class;
     end;
   ** make list_many_c, _many_n, for mvars;
   if      cr_type eq ’many’ then do;
      if      type eq ’c’ then
            _list_many_c = catx(’ ’,_list_many_c,name);
      else if type eq ’n’ then
            _list_many_n = catx(’ ’,_list_many_n,name);
      end;
   else if cr_type eq ’few’  then do;
      **** save for sort: lfs(i)= ’000:name’;
      _lfs(_i) = catx(’:’,put(n_levels
                             ,z%length(&_n_obs).),name);
      *** add one for delimiter = space;
      _length_few = sum(1,_length_few);
      if type eq ’c’ then do;
           _max_length=max(_max_length,length);
           _length_few=sum(_length_few,length);
           end;
      else _length_few=sum(_length_few,%length(&_n_obs));
      end;
   else if cr_type eq ’.unique’  then
       _list_unique = catx(’ ’,_list_unique,name);
   output;
   end;
_rc = close(_dsid);

**** loop: make list-few ordered by n-levels;
call sortc(of _lfs(*));
do _i = 1 to dim(_lfs);
   if _testing and _lfs(_i) ne ’ ’ then putlog _lfs(_i)=;
   ****        _lfs(_i)=0000:name;
   name = scan(_lfs(_i),-1,’:’);
   _list_few = catx(’ ’,_list_few,name);
   end;

call symputx(’_length_few’  ,_length_few );
call symputx(’_list_few’    ,_list_few   );
call symputx(’_list_many_c’ ,_list_many_c);
call symputx(’_list_many_n’ ,_list_many_n);
call symputx(’_list_unique’ ,_list_unique);
call symputx(’_max_length_c’,_max_length );
stop;
run;
%put info: &=memname &=_n_obs &=_n_vars;
%put info: &=_length_few;
%put info: &=_list_few;
%put info: &=_list_many_c;
%put info: &=_list_many_n;
%put info: &=_list_unique;
%put info: &=_max_length_c;
%put trace: ml-namex make-list-names-cr ending;
```

Demonstration Programs, Fast Library Review

Overview

This section contains listings of these programs.

- report memnames information
- routine ml-namex, call, append
- program make-list-memnames with cardinality ratio
- make-list-memnames output

report memnames information

This report lists the information — n-obs, n-vars, data set label — of all data sets in a *libref*.

```
%let libname = sashelp;
%put echo parameters: &=libname;
PROC sql; describe table dictionary.tables;
          create   table list_memnames as
          select   memname, nobs, nvar, memlabel
          from     dictionary.tables
          where    libname eq "%upcase(&libname)"
            and    memtype eq 'DATA';
          describe table &syslast;
          select * from  &syslast;
          title3  "&libname members=&sqlobs";
          quit;
%put info: &=sqlobs;
```

output listing

```
memname  nobs  nvar  memlabel
-------
AACOMP   1544     4
AARFM      61     4
ADSMSG    426     6
AFMSG    1090     6
AIR       144     2  airline data (monthly: JAN49-DEC60)
```

routine ml-namex-call-append

This program calls the subroutine `ml-namex.sas` and appends the output to the data set `list_memnames`.

```
*ml-namex-call-append.sas;
%include site_inc(ml-namex);
PROC append data = &syslast
            base = list_memnames;
run;
```

make-list-memnames with cardinality ratio

```
*make-list-memnames-with-cr;
options mprint source2;
%let libname = sashelp;
PROC sql; select catt('%let memname=',memname
                ,';%include project(ml-namex-call-append);')
          into  :list_memnames   separated by ' '
          from   dictionary.tables
          where  libname eq "%upcase(&libname)"
            and  memtype eq 'DATA'
            and  nobs and nvar;
          quit;
%put info: &=sqlobs;
&list_memnames
run;
PROC print data = &syslast;
           title3 &syslast;
           title4 "&libname contains &sqlobs members";
           by     memname;
           id     memname;
```

make-list-memnames output

This is an example of the output of make-list-memnames-with-cr.sas.

```
         var               card-   n-
memname  num  cr-type      ratio   levels  name    type  length
-------  ---  ----------  -------  ------  ------  ----  ------
AACOMP     1  few         0.00583       9  locale   c         5
           2  few         0.11917     184  key      c        60
           3  n-levels=1  0.00065       1  lineno   n         4
           4  many        0.87111    1345  text     c      1200

AARFM      1  n-levels=1  0.01639       1  locale   c         5
           2  .unique           1      61  key      c        60
           3  n-levels=1  0.01639       1  lineno   n         4
           4  many        0.96721      59  text     c      1200
```

! → This data set can be sorted by `lowcase(name)` to identify variables that have different types.

Programs, Summary-Each-Var, version 2016.08

Overview

This section has listings of programs for the SmryEachVar suite, version 2016.08.

- cx-include-list-names
- proc-freq
- proc-smry
- demo-SmryEachVar-2016
- SmryEachVar output

cx-include-list-names.sas

This program is a custom version of the routine CallXinc, Fehd [5].

```
%put trace: cx-include-list-names beginning;
%put echo parameters &=cx_data &=cx_include;
DATA _null_;
     attrib statement length = $
     %sysfunc(max(%eval(
      %length(*let name=)+32+1+%length(*let type=C)+1)
     ,%length(*include &cx_include;)  ));
do until(endofile);
   set &cx_data %*may contain where(...);
       end = endofile;
   statement =  catt(’%let name=’,name,’;’
                    ,’%let type=’,type,’;’);
   call execute(catt(’%nrstr(’,statement,’)’));
   statement = "%include &cx_include;";
   call execute(catt(’%nrstr(’,statement,’)’));
   end;
stop;
run;
%put trace: cx-include-list-names ending;
```

proc-freq.sas

This program standardizes the output data set from the frequency procedure so that it can contain both character and numeric variables, for `cr-type eq ’few’`.

```
%put trace: proc-freq beginning;
%put echo parameters:&=libname &=memname &=name &=type;
%put echo parameter :&=_max_length_c from ml-namex;
PROC freq data   = &libname..&memname;
          tables   &name  / list missing noprint
          out    =  out_freq
         (rename =(&name = var_&type) );
run;
%put echo parameter: &=_max_length_c;
DATA &syslast;
     attrib memname length = $32
               name length = $32
            var_c   length = $&_max_length_c
            var_n   length = 8;
     if 0 then set &syslast(keep = count percent);
     retain memname "&memname"
               name "&name"
            var_c   ’.’
            var_n    .;
do until(endofile);
   set &syslast end = endofile;
   output;
   end;
stop;
run;
PROC append data = &syslast
               base = list_freq;
run;
%put trace: proc-freq ending;
```

proc-smry.sas

This program provides a simple set of summary statistics, one row for each continuous variable, for `cr-type eq 'many' and type eq 'n'`.

```
%put trace: proc-smry beginning;
%put echo parameters:&=libname &=memname &=name &=type;
PROC summary data   = &libname..&memname;
             var      &name;
             output
                out = out_summary
            (  drop =_type_  _freq_)
    mean    (&name) = mean    %*average;
    std     (&name) = std_dev
    min     (&name) = min     %*p000;
    median  (&name) = median  %*p050;
    max     (&name) = max     %*p100;
    ;
run;
DATA &syslast;
     attrib memname length = $32
               name length = $32;
     retain memname "&memname"
               name "&name";
do until(endofile);
   set &syslast end = endofile;
   output;
   end;
stop;
run;
PROC append data = &syslast
            base = list_smry;
run;
%put trace: proc-smry ending;
```

demo-SmryEachVar-2016.sas

This program shows the basic methods of the SmryEachVar suite:

1. call subroutine to make list of variable names
2. call proc frequency for each of the by or class variables
3. call proc summary for each of the analysis variables

```
*name: SmryEachVar, version 2016.08;
*options mprint source2;
%let libname = sashelp;
%let memname = bweight;
%let memname = class;

** 1 make list of variable names;
%include site_inc(ml-namex);
PROC print  data =   &syslast noobs;
     title3 "%cmpres(&syslast unique: &_list_unique)";

** 2 freq of few, either char or num;
%let cx_data    = list_names(where=(cr_type eq ’few’));
%let cx_include = project(proc-freq);
%include project(cx-include-list-names);
PROC print  data =   &syslast noobs;
     title3 "%cmpres(&syslast few: &_list_few)";
            by memname notsorted name;
            id memname           name;

** 3 summary of many, only numeric;
%let cx_data    = list_names(where=(cr_type eq ’many’
                                   and type eq ’n’));
%let cx_include = project(proc-smry);
%include project(cx-include-list-names);
PROC print  data =   &syslast noobs;
     title3 "%cmpres(&syslast many.n: &_list_many_n)";
run;
```

SmryEachVar output

```
Calculating Cardinality Ratio and -type in 2 steps
demo-SmryEachVar-2016
list_names unique: Name
                                  card_
memname    varnum    cr_type      ratio    n_levels    name      type    length    format    informat    label
-------    ------    -------    -------    --------    ------    ----    ------
 class        1      .unique          1          19    Name       c         8
 class        2      few        0.10526           2    Sex        c         1
 class        3      few        0.31579           6    Age        n         8
 class        4      many       0.89474          17    Height     n         8
 class        5      many       0.78947          15    Weight     n         8

list_freq few: Sex Age
memname    name    var_c    var_n    COUNT    PERCENT
-------    ----    -----    -----    -----    -------
 class     Sex     F           .        9     47.3684
                   M           .       10     52.6316
 class     Age     .          11        2     10.5263
                   .          12        5     26.3158
                   .          13        3     15.7895
                   .          14        4     21.0526
                   .          15        4     21.0526
                   .          16        1      5.2632

list_smry many.n: Height Weight
memname     name       mean       std       min    median     max
-------    ------     ------     ------    ----    ------    ---
 class     Height     62.337     5.1271    51.3     62.8      72
 class     Weight    100.026    22.7739    50.5     99.5     150
```

Summary

Commentary

The log of this subroutine's use of the frequency procedure has been reduced by an order of magnitude, from *O(n-vars)* to *O(1)* While this appears impressive theoretically, in fact the frequency procedure still does the counting of n-levels for each variable and the overall time to run the single call of frequency-nlevels is approximately equal to the sum of the time for frequencies of each variable.

Conclusion

By analyzing the data structure of both the input and output data sets from the procedures used in previous algorithms we have found an optimum data structure for the desired output and moved calculations from several data steps and procedures into a single data step with arrays.

Suggested Reading

predecessors : Fehd [1], Macro FreqAll; Fehd [2], SmryEachVar;
Fehd [3], Database Vocabulary: dimension table;
Fehd [9], Read Column into Row;
Fehd [6], Data Review with N-Levels or Cardinality Ratio

routines : that use this subroutine for setup
Fehd [5], List Processing Routine CallXinc;
Fehd [4], Using SmryEachVar
Fehd [7], Macro CallMacro;
Fehd [8], Using Cardinality Ratio

scl : Fraeman [10], using scl functions to get data structure information;
Yindra [11], review of data step and scl functions for use with `sysfunc`

programs : sas.community.org: Cardinality Ratio definition, Making Lists,
Summarize Each Var, Routine CxInclude (CallXinc),
Routine CallXinc,

Author Information

Ronald J. Fehd `Ron.Fehd.macro.maven@gmail.com`

About the author:

sco.wiki `http://www.sascommunity.org/wiki/Ronald_J._Fehd`
LinkedIn `www.linkedin.com/Ronald.Fehd`
affiliation Stakana Analytics, Senior Maverick
also known as `macro maven` on SAS-L, Theoretical Programmer

Programs: **http://www.sascommunity.org/wiki/**

Trademarks

References

[1] Ronald J. Fehd. Journeymen's tools: Data review macro FreqAll — using Proc SQL list processing with Dictionary.Columns to eliminate macro do loops. In *SAS Global Forum Annual Conference Proceedings*, 2007. URL http://www2.sas.com/proceedings/forum2007/028-2007.pdf. Coder's Corner, 10 pp.; attributes, dictionary.columns, metadata, proc append, proc freq, proc sql, program header; bibliography.

[2] Ronald J. Fehd. SmryEachVar: A data-review routine for all data sets in a libref. In *SAS Global Forum Annual Conference Proceedings*, 2008. URL http://www2.sas.com/proceedings/forum2008/003-2008.pdf. Applications Development, 24 pp.; call execute, data review, data structure, dynamic programming, list processing, parameterized includes, utilities (writattr, writvalu) to repair missing elements in data structure; best contributed paper in ApDev.

[3] Ronald J. Fehd. Database vocabulary: Is your data set a dimension (lookup) table, a fact table or a report? In *Western Users of SAS Software Annual Conference Proceedings*, 2008. URL http://wuss.org/proceedings08/08WUSS%2520Proceedings/papers/dmw/dmw04.pdf. Databases and Warehouses, 8 pp.; cardinality ratio, composite key, database design, foreign key, grain, nlevels, normal forms, primary key, relational database, snapshots: accumulating or periodic.

[4] Ronald J. Fehd. Using the SmryEachVar data review suite. In *Western Users of SAS Software Annual Conference Proceedings*, 2009. URL http://www.lexjansen.com/wuss/2009/app/APP-Fehd1.pdf. Applications Development, 20 pp.; call execute, data review, data structure, dynamic programming, list processing, parameterized includes, examples.

[5] Ronald J. Fehd. List processing routine CallXinc: Calling parameterized include programs using a data set as list of parameters. In *Western Users of SAS Software Annual Conference Proceedings*, 2009. URL www.lexjansen.com/wuss/2009/app/APP-Fehd2.pdf. Applications Development, 20 pp.; call execute, data review, data structure, dynamic programming, list processing, parameterized includes, examples.

[6] Ronald J. Fehd. Data review information: N-levels or cardinality ratio. In *SAS Global Forum Annual Conference Proceedings*, 2013. URL http://support.sas.com/resources/papers/proceedings13/299-2013.pdf. Statistics and Data Analysis, 6 pp.; using proc freq nlevels and nobs to calculate cardinality ratio — range in (0:1) — of a variable to determine its type in (continuous, discrete, unique, worthless).

[7] Ronald J. Fehd. List processing macro call-macro. In *MidWest SAS Users Group Annual Conference Proceedings*, 2014. URL www.mwsug.org/proceedings/2014/BB/MWSUG-2014-BB04.pdf. Coders Corner, 19 pp.; using %sysfunc with SCL functions to read a list, a control data set, and for each observation, call a macro with variable names and values as named parameters.

[8] Ronald J. Fehd. Using cardinality ratio for fast data review. In *Western Users of SAS Software Annual Conference Proceedings*, 2014. URL http://www.lexjansen.com/wuss/2014/98_Final_Paper_PDF.pdf. Coders Corner, 14 pp.; successor of SmryEachVar, macros to calculate cardinality ratio and process discrete and continuous variables with procs freq, mode, and summary.

[9] Ronald J. Fehd. Reading a column into a row to count n-levels, calculate cardinality ratio and create frequency and summary output in one step. In *MidWest SAS Users Group Annual Conference Proceedings*, 2015. URL http://www.lexjansen.com/mwsug/2015/BB/MWSUG-2015-RF-04.pdf. Rapid Fire, 10 pp.

[10] Kathy Hardis Fraeman. Get into the groove with %sysfunc: Generalizing SAS(R) macros with conditionally executed code. In *SAS Users Group International Annual Conference Proceedings*, 2008. URL http://www.lexjansen.com/nesug/nesug08/po/po06.pdf. Posters, 8 pp.; using scl functions to get data set information, 4 examples.

[11] Chris Yindra. %Sysfunc — the brave new macro world. In *SAS Users Group International Annual Conference Proceedings*, 1998. URL http://www2.sas.com/proceedings/sugi23/Advtutor/p44.pdf. Advanced Tutorials, 7 pp.; review of data step and scl functions available to %sysfunc, 10 examples.