

Calculating Cardinality Ratio in Two Steps

Ronald J. Fehd, senior maverick, theoretical programmer,
Fragile-Free Software Institute

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 1/n-obs to one: (0–1].

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, empty); this variable identifies the three main types of variables in a data set, few is discrete, many is continuous, unique is a row-identifier, and empty is a single value.

purpose: The purpose of this paper is to provide a general-purpose program, ml-names-card-ratios.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 discipline 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	2
	Research for data structure: contents, sql, scl, n-levels	5
	Explanation	9
	Program listings	13
	Demonstration programs, fast library review	15
	Programs, summary-each-var, version 2016.08	16
	Summary	20
	References	22

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 $O(n\text{-vars})$ 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 16.

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	$n\text{-rows}(table)$
levels:	number of values in a column	$n\text{-levels}(column)$
cardinality ratio (cr):	$\frac{n\text{-levels}(column)}{n\text{-rows}(table)}$	range: (0–1]
	_____ cardinality-type _____	
values: 0	few distinct by class var	mean(cr) continuous analysis var
		many continuous analysis var
		unique 1

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 `sashe1p` 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, "Database Vocabulary: Is Your Data Set a Dimension (LookUp) Table, a Fact Table or a Report?", shows which database tables contain which columns.

columns		tables			
name	type	dimension	fact	snapshots	
				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?

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

previous algorithm

This is pseudo-code of the previous algorithm first shown in the SmryEachVar suite, 2008: Fehd, "SmryEachVar: A Data-Review Routine for All Data Sets in a Libref",

and further developed in 2008: Fehd, "Database Vocabulary: Is Your Data Set a Dimension (LookUp) Table, a Fact Table or a Report?",

2013: Fehd, "Data Review Information: N-Levels or Cardinality Ratio",

and 2014: Fehd, "Using Cardinality Ratio for Fast Data Review".

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 lowercase
- 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

1. make data set list-names
with proc freq nlevels
2. copy data set attributes *n-obs* and *n-vars*
to macro variables in global symbol table
for use in array allocation and calculation
3. 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.

memname	var	card_	n_	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;
run;
```

describe contents output

```
create table WORK.CONTENTES
(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

Member	Column	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
        in SAS documentation 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;
run;
```

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 lowercase(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-names-card-ratios.sas` is on page 13.

- autoexec
- testing program
- proc freq 10
- copy n-obs, n-vars
- data structure
- array of cardinality ratios
- read data structure 11
- make cr-type
- robust length of lists
- make lists
- make lengths 12
- make list-few
- make macro variables
- echo macro variables

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-names-card-ratios.sas`.

```
** name: autoexec.sas;
filename project '.'; *** contains programs listed here;
*author's useage: folder containing ml-names-card-ratios.sas;
*filename site_inc '<...>\sas-includes';
*for your testing;;
filename site_inc '.*';*folder containing ml-names-card-ratios.sas;
```

testing program

This is the program used to test `ml-names-card-ratios.sas`.

```
options mprint source2;
%let libname = sashelp;
%let memname = class;
%include site_inc(ml-names-card-ratios);
proc print data = &syslast noobs;
run;
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));
run;%put echo &=sysnobs &=syslast;
```

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)); * compare to sysnobs;
%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 = '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        = lowercase(vartype(_dsid,varnum)); *NOTE lowercase;
```

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 $(32+1)*n\text{-vars}$.

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));
*** 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-names-card-ratios 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-names-card-ratios make-list-names-cr ending
```

Program listings

notes: Programs that use this subroutine are on page 15.

ml-names-card-ratios.sas

This is the listing of the program ml-names-card-ratios.sas.

```
%put trace: ml-names-card-ratios-types beginning;
%put echo parameters: &=libname &=memname;
/* name: sas-includes\ml-names-card-ratios.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: dataset 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
note: output is ordered by varnum
-----
- - - ml-names-card-ratios-test.sas - - -
%let libname = sashelp;
%let memname = class;
%include 'ml-names-card-ratios.sas';
proc print data = list_names;
-----
date-time: 2024-05-03 09:45:47
**** ..... */
**** make dimension table: name + n-levels;
%let list_names=list_names;
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;
%put echo &=sysnobs &=syslast;
```

```

proc sql;describe table &syslast;quit;
**** 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,nvars     ));
%let _rc      = %sysfunc(close(&_dsid         ));
%symdel _dsid _rc;
%put echo &=_n_obs &=_n_vars;
**** enhance dimension table with variable information,
      cardinality ratio, card-ratio-type: cr-type;
data list_names /*&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 = d7.5
                    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;
  array _cr(&_n_vars);
  drop _.; * _temporary variables;
  retain memname  "&memname"
    _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 &list_names /*&syslast*/
    (keep = n_levels) point = _i;
  _cr(_i) = n_levels / &_n_obs;
end;
**** calculate median for select card_ratio to cr_type;
_mean_cr = mean(of _cr(*));
_median_cr= median(of _cr(*));
if _testing then putlog 'info: ' _mean_cr= _median_cr=;
**** loop: read syslast(name n_levels), fetch var info;
_dsid = open("&libname..&memname");** for scl functions;
*do _i = 1 to dim(_cr);
do varnum = 1 to dim(_cr);
  set &list_names /*&syslast */
    point = varnum; *** primary-key=name;
    *point = _i; *** primary-key=name;
  if _testing then putlog 'info: ' name= n_levels=;
* varnum = varnum (_dsid,name );

```

```

type      = lowercase(vartype (_dsid,varnum));
length   =          varlen  (_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
         _median_cr) cr_type= 'many';  %*analysis;
  otherwise          cr_type= 'few';  %*by|class;
end;
output;
end;
_rc = close(_dsid);
stop;
run;
%put echo &=sysnobs &=syslast;
proc sql;describe table &syslast;quit;
%put trace: ml-names-card-ratios-types ending;

```

Demonstration programs, fast library review

overview

This section contains listings of these programs.

- report memnames information
- routine ml-names-card-ratios, 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=&sqllobs";
quit;
%put info: &=sqllobs;

```

output listing

memname	nobs	nvar	memlabel

AACOMP	1544	4	
AARFM	61	4	
ADSMMSG	426	6	

```
AFMSG    1090    6
AIR      144    2  airline data (monthly: JAN49-DEC60)
```

**routine
ml-namex-call-append**

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

```
*name: 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: &=sqllobs;
&list_memnames
run;
proc print data = &syslast;
            title3 &syslast;
            title4 "&libname contains &sqllobs members";
            by memname;
            id memname;
run;
```

**make-list-memnames
output**

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

memname	var	card-	n-				
-----	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, "List Processing Routine CallXinc: Calling Parameterized Include Programs Using a Data Set as List of Parameters".

```
%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

memname	varnum	cr_type	card_ 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 $O(n\text{-vars})$ 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, "Journeymen's Tools: Data Review Macro FreqAll — Using PROC SQL List Processing with Dictionary.Columns to Eliminate Macro DO Loops" Fehd, "SmryEachVar: A Data-Review Routine for All Data Sets in a Libref"
Fehd, "Database Vocabulary: Is Your Data Set a Dimension (LookUp) Table, a Fact Table or a Report?",
Fehd, "Reading a Column into a Row to Count N-levels, Calculate Cardinality Ratio and Create Frequency and Summary Output In One Step",
Fehd, "Data Review Information: N-Levels or Cardinality Ratio"
- routines : that use this subroutine for setup
Fehd, "List Processing Routine CallXinc: Calling Parameterized Include Programs Using a Data Set as List of Parameters",
Fehd, "Using the SmryEachVar Data Review Suite",
Fehd, "List Processing Macro Call-Macro",
Fehd, "Using Cardinality Ratio for Fast Data Review"
- scl : Fraeman, "Get into the Groove with %SYSFUNC: Generalizing SAS(R) Macros with Conditionally Executed Code", using scl functions to get data structure information;
Yindra, "%SYSFUNC — The Brave New Macro World", review of data step and scl functions for use with `sysfunc`
-

Author Information

Ronald J. Fehd

Ron.Fehd.macro.maven at gmail dot com

LinkedIn
affiliation

<https://www.linkedin.com/in/ronald-fehd-5125991/>
senior maverick, theoretical programmer,
Fragile-Free Software Institute,

also known as

macro maven on SAS-L

Trademarks

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

References

- Fehd, Ronald J. (2007). "Journeyman'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*. Coders Corner, 10 pp.; attributes, dictionary.columns, metadata, proc append, proc freq, proc sql, program header; designing macros for reporting, creating and using macro arrays, writing text of macro calls into macro variable, executing macro calls in macro variable, bibliography. URL: <http://www2.sas.com/proceedings/forum2007/028-2007.pdf>.
- (2008a). "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*. 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. URL: <http://wuss.org/proceedings08/08WUSS%2520Proceedings/papers/dmw/dmw04.pdf>.
- (2008b). "SmryEachVar: A Data-Review Routine for All Data Sets in a Libref". In: *SAS Global Forum Annual Conference Proceedings*. 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. URL: <http://www2.sas.com/proceedings/forum2008/003-2008.pdf>.
- (2009a). "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*. Applications Development, 20 pp.; call execute, data review, data structure, dynamic programming, list processing, parameterized includes, examples. URL: <http://www.lexjansen.com/wuss/2009/app/APP-Fehd2.pdf>.
- (2009b). "Using the SmryEachVar Data Review Suite". In: *Western Users of SAS Software Annual Conference Proceedings*. Applications Development, 20 pp.; call execute, data review, data structure, dynamic programming, list processing, parameterized includes, examples. URL: <http://www.lexjansen.com/wuss/2009/app/APP-Fehd1.pdf>.
- (2013). "Data Review Information: N-Levels or Cardinality Ratio". In: *SAS Global Forum Annual Conference Proceedings*. Statistics and Data Analysis, 6 pp.; using proc freq nlevels and nob to calculate cardinality ratio — range in (0:1) — of a variable to determine its type in (continuous, discrete, unique, worthless). URL: <http://support.sas.com/resources/papers/proceedings13/299-2013.pdf>.
- (Oct. 2014a). "List Processing Macro Call-Macro". In: *MidWest SAS Users Group Annual Conference Proceedings*. 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. URL: <http://www.mwsug.org/proceedings/2014/BB/MWSUG-2014-BB04.pdf>.
- (2014b). "Using Cardinality Ratio for Fast Data Review". In: *Western Users of SAS Software Annual Conference Proceedings*. Coders Corner, 14 pp.; successor of SmryEachVar, macros to calculate cardinality ratio and process discrete and continuous variables with procs freq, mode, and summary. URL: http://www.lexjansen.com/wuss/2014/98_Final_Paper_PDF.pdf.
- (2015). "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*. Rapid Fire, 10 pp. URL: <http://www.lexjansen.com/mwsug/2015/RF/MWSUG-2015-RF-04.pdf>.
- Fraeman, Kathy Hardis (2008). "Get into the Groove with %SYSFUNC: Generalizing SAS(R) Macros with Conditionally Executed Code". In: *SAS Users Group International Annual Conference Proceedings*. Posters, 8 pp.; using scl functions to get data set information, 4 examples. URL: <http://www.lexjansen.com/nesug/nesug08/po/po06.pdf>.
- Yindra, Chris (1998). "%SYSFUNC — The Brave New Macro World". In: *SAS Users Group International Annual Conference Proceedings*. Advanced Tutorials, 7 pp.; review of data step and scl functions available to %sysfunc, 10 examples. URL: <http://www2.sas.com/proceedings/sugi23/Advttutor/p44.pdf>.