PharmaSUG 2026 - Paper AP-246

Appreciating the PROC SUMMARY/MEANS in Many NWAYS vs Summary Function in R

Brian Varney, Experis

ABSTRACT

The summary/means procedure in SAS is a very powerful and flexible procedure for summarizing continuous analysis variables and returning simple summary statistics. This paper intends to examine what happens when the NWAY option is left off PROC SUMMARY in detail, the problems that can arise when you have many class variables. We will examine a SAS Macro that mimics the PROC SUMMARY without the NWAY option. We will also compare the same approach in R using the Tidyverse summary function.

INTRODUCTION

This paper is intended for beginning SAS and R programmers and those interested in SAS Macro and R functions. We will look at what happens in depth when one leaves off the NWAY option in PROC SUMMARY and examine a SAS Macro that mimics that functionality. Finally, we will examine how to carry out these processes using the R language.

By reading this paper, one will better understand how PROC SUMMARY works, learn some SAS Macro techniques, and learn carrying out similar processing in R.

PROC SUMMARY/MEANS

First off, we will recognize that PROC SUMMARY and PROC MEANS are essentially the same procedure with different initial options. Specifically, PROC MEANS automatically sends results to the output/results window while PROC SUMMARY does not. Therefore, we will just make references to PROC SUMMARY for the remainder of this paper.

Let's examine a couple of PROC SUMMARY programs and output with and without the NWAY option respectively.

```
proc summary data=dm nway;
   class sex arm;
   var age dmdy;
   output out=dmsumm_nway n=n_age n_dmdy mean=mean_age mean_dmdy;
run;
```

Program 1. PROC SUMMARY with two class variables with the NWAY option.

	SEX	ARM	_TYPE_	_FREQ_	n_age	n_dmdy	mean_age	mean_dmdy
1	F	Placebo	3	53	53	53	76.358490566	-11.16981132
2	F	Screen Failure	3	36	36	0	73.25	.
3	F	Xanomeline High Dose	3	40	40	40	74.675	-9.75
4	F	Xanomeline Low Dose	3	50	50	50	75.68	-12.72
5	M	Placebo	3	33	33	33	73.363636364	-10.84848485
6	M	Screen Failure	3	16	16	0	79.25	.
7	M	Xanomeline High Dose	3	44	44	44	74.113636364	-10.90909091
8	M	Xanomeline Low Dose	3	34	34	34	75.647058824	-9.941176471

Output 1. PROC SUMMARY output data set with two class variables with the NWAY option.

```
proc summary data=dm;
   class sex arm;
   var age dmdy;
   output out=dmsumm_nway n=n_age n_dmdy mean=mean_age mean_dmdy;
run;
```

Program 2. PROC SUMMARY with two class variables without the NWAY option.

	SEX	ARM	_TYPE_	_FREQ_	n_age	n_dmdy	mean_age	mean_dmdy
1			0	306	306	254	75.088235294	-11
2		Placebo	1	86	86	86	75.209302326	-11.04651163
3		Screen Failure	1	52	52	0	75.096153846	.
4		Xanomeline High Dose	1	84	84	84	74.380952381	-10.35714286
5		Xanomeline Low Dose	1	84	84	84	75.666666667	-11.5952381
6	F		2	179	179	143	75.167597765	-11.31468531
7	M		2	127	127	111	74.976377953	-10.59459459
8	F	Placebo	3	53	53	53	76.358490566	-11.16981132
9	F	Screen Failure	3	36	36	0	73.25	.
10	F	Xanomeline High Dose	3	40	40	40	74.675	-9.75
11	F	Xanomeline Low Dose	3	50	50	50	75.68	-12.72
12	M	Placebo	3	33	33	33	73.363636364	-10.84848485
13	M	Screen Failure	3	16	16	0	79.25	.
14	M	Xanomeline High Dose	3	44	44	44	74.113636364	-10.90909091
15	M	Xanomeline Low Dose	3	34	34	34	75.647058824	-9.941176471

Output 2. PROC SUMMARY output data set with two class variables without the NWAY option.

If you compare Output 1 and Output 2, you will notice that Output 2 gives all the different combinations of breakouts for the variables SEX and ARM. Some programmers like to use this method and filter out the breakouts using the _type_ variable.

The _type_ variable may not seem intuitive at first because of its default representation in the output data set. But if we also specify one additional option called CHARTYPE, we can make sense of it.

```
proc summary data=dm chartype;
   class sex arm;
   var age dmdy;
   output out=dmsumm_nway n=n_age n_dmdy mean=mean_age mean_dmdy;
run;
```

Program 3. PROC SUMMARY with two class variables without the NWAY option specifying the CHARTYPE option.

	SEX	ARM	_TYPE_	_FREQ_	n_age	n_dmdy	mean_age	mean_dmdy
1			00	306	306	254	75.088235294	-11
2		Placebo	01	86	86	86	75.209302326	-11.04651163
3		Screen Failure	01	52	52	0	75.096153846	.
4		Xanomeline High Dose	01	84	84	84	74.380952381	-10.35714286
5		Xanomeline Low Dose	01	84	84	84	75.666666667	-11.5952381
6	F		10	179	179	143	75.167597765	-11.31468531
7	M		10	127	127	111	74.976377953	-10.59459459
8	F	Placebo	11	53	53	53	76.358490566	-11.16981132
9	F	Screen Failure	11	36	36	0	73.25	.
10	F	Xanomeline High Dose	11	40	40	40	74.675	-9.75
11	F	Xanomeline Low Dose	11	50	50	50	75.68	-12.72
12	M	Placebo	11	33	33	33	73.363636364	-10.84848485
13	M	Screen Failure	11	16	16	0	79.25	.
14	M	Xanomeline High Dose	11	44	44	44	74.113636364	-10.90909091
15	M	Xanomeline Low Dose	11	34	34	34	75.647058824	-9.941176471

Output 3. PROC SUMMARY output data set with two class variables without the NWAY option specifying the CHARTYPE option.

Now we can see that the _type_ variable is actually easier to interpret in binary format. Each position of the binary _type_ value represents the class variable in that position in the CLASS statement. If it is a “1”, it is summarized by that value. If it is a “0”, it is summarized over that value.

If we wanted to filter the data set for breakout only by SEX, we could use a where statement on the output data set as follows:

where put(_type_, binary2.) = “10”; ** summarized by SEX but summarized over all ARM values.

If you want to predict how many different type values the output data set contain, you can just convert the binary number of all ones for the number of class variables you have in your CLASS statement. See table 1 below.

Number of CLASS Variables	Binary Representation of Highest _type_ Value	Base 10 Representation of Highest _type_ Value
1	1	1
2	11	3
3	111	7
4	1111	15
5	11111	31
10	1111111111	1,023
15	111111111111111	32,767
20	11111111111111111111	1,048,575

Calculation by hand for 5 CLASS Variables

$(11111)_2 = (1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) = (31)_{10}$

As you can see, the number of _type_ variable values essentially doubles every time you add an additional CLASS variable. This becomes more memory intensive as the number of CLASS variables grows. If your CLASS variables have a lot of levels, the data set can become very large, and you may get errors because of memory limits.

To fully appreciate what PROC SUMMARY is doing in memory when not using the NWAY option, let's look at some code that mimics the operation.

```
%macro construct_non_nway(
  inds =,
  outds = ,
  classvars = ,
  avar = ,
  missing = );

%let nvars = %sysfunc(countw(&classvars.));
%put &nvars.;

data _null_;
   high = input(repeat('1', &nvars.), binary&nvars..);
   call symput('high', trim(left(put(high, best.))));
   put _all_;
run;

data _null_;
   do i = 0 to &high.;
     bin = put(i, binary&nvars..);
       call symput('combo'||trim(left(put(i, best.))), bin);
   end;
run;

%do i = 0 %to &high.;

%let varselected=;
%let varsel=;

%do va = 1 %to &nvars.;
  %if %substr(&&combo&i., &va., 1) ne 0 %then
  %do;
  %let varsel = &varsel. %scan(&classvars., &va.);
  %end;
%end;

%put combo: &&combo&i. varsel: &varsel.;

proc summary data=&inds. nway &missing.;
%if "&varsel." ne "" %then
%do;
   class &varsel.;
%end;
   var &avar.;
   output out=c&&combo&i. n=n sum=sum mean=mean;
   %if "&missing."="" %then
   %do;
   where
      %do va = 1 %to &nvars.;
           %scan(&classvars., &va.) is not missing
              %if &va. ne &nvars. %then
              %do;
                 and
              %end;
        %end;
   ;
   %end;
run;

data c&&combo&i.;
  set c&&combo&i.;
  _type_ = put(input("&&combo&i.", binary&nvars..), best.);
```

```
run;

%end;

data &outds.;
  set
  %do d=0 %to &high.;
  c&&combo&d.
  %end;
  ;
run;

%mend construct_non_nway;

%construct_non_nway(
  inds  = sashelp.cars,
  outds = carsnonnway,
  classvars  = origin type drivetrain cylinders,
  avar = msrp,
  missing = missing);
```

Program 4. Macro to simulate the results from a PROC SUMMARY that does not use the NWAY option.

To fully appreciate what PROC SUMMARY is doing in memory when not using the NWAY option, let's look at some code that mimics the operation using data step and hash operations.

```
data cars_hash_summary;
  length origin type drivetrain $ 13;
  length _type_ 8 _freq_ 8 n 8 sum 8 mean 8;
  length k_origin k_type k_drivetrain $ 13;
  length k_cylinders 8;

  if _n_ = 1 then do;
    declare hash h(ordered:'a');
    h.defineKey('_type_', 'k_origin', 'k_type', 'k_drivetrain', 'k_cylinders');
    h.defineData('_type_', 'k_origin', 'k_type', 'k_drivetrain', 'k_cylinders',
                 '_freq_', 'n', 'sum');
    h.defineDone();

    declare hiter hi('h');
  end;

  set sashelp.cars end=eof;

  length bit1-bit4 8;

  /* Generate all non-NWAY combinations: 0 to 15 */
  do _type_ = 0 to 15;

    /* bit flags for ORIGIN TYPE DRIVETRAIN CYLINDERS */
    bit1 = band(_type_, 1) > 0;
    bit2 = band(_type_, 2) > 0;
    bit3 = band(_type_, 4) > 0;
    bit4 = band(_type_, 8) > 0;

    /* Keep value only if CLASS var participates in this _TYPE_ */
    k_origin     = ifc(bit1, origin, '');
    k_type       = ifc(bit2, type, '');
    k_drivetrain = ifc(bit3, drivetrain, '');
    k_cylinders  = ifn(bit4, cylinders, .);
```

```
    rc = h.find();

    if rc ne 0 then do;
      _freq_ = 0;
      n      = 0;
      sum    = 0;
    end;

    _freq_ + 1;

    if not missing(msrp) then do;
      n   + 1;
      sum + msrp;
    end;

    h.replace();
  end;

  if eof then do;
    rc = hi.first();
    do while (rc = 0);

      origin     = k_origin;
      type       = k_type;
      drivetrain = k_drivetrain;
      cylinders  = k_cylinders;

      if n > 0 then mean = sum / n;
      else mean = .;

      output;
      rc = hi.next();
    end;
  end;

  keep origin type drivetrain cylinders _type_ _freq_ n sum mean;
run;
```

Program 5. Data step and hash operations to mimic results from a PROC SUMMARY that does not use the NWAY option.

It is interesting to mimic the SAS Macro above with an R function.

```
library(dplyr)
library(tidyr)
library(stringr)
library(purrr)

construct_non_nway <- function(inds, classvars, avars, missing = "") {

  nvars <- length(classvars)
  cat("nvars:", nvars, "\n")

  high <- 2^nvars - 1
  cat("high:", high, "\n")

  combo <- character(high + 1)
  for (i in 0:high) {
    # Convert integer to binary string of length nvars
    combo[i + 1] <- paste(rev(as.integer(intToBits(i))[1:nvars]), collapse = "")
    # Pad to nvars length (intToBits gives 32 bits, we take last nvars and reverse)
  }
  # Fix: need proper binary representation with leading zeros, length nvars
  combo <- vapply(0:high, function(i) {
```

```
  bits <- integer(nvars)
  val <- i
  for (b in nvars:1) {
    bits[b] <- val %% 2
    val <- val %/% 2
  }
  paste(bits, collapse = "")
}, character(1))

# List to collect all combination datasets
combo_datasets <- vector("list", high + 1)

# %do i = 0 %to &high.;
for (i in 0:high) {

  bin_str <- combo[i + 1]

  varsel <- character(0)
  for (va in 1:nvars) {
    if (substr(bin_str, va, va) != "0") {
      varsel <- c(varsel, classvars[va])
    }
  }

  cat("combo:", bin_str, "varsel:", paste(varsel, collapse = " "), "\n")


  work_data <- inds

  # Apply filter: if missing parameter is empty, filter out missing values
  # for ALL class variables
  if (missing == "") {
    for (va in 1:nvars) {
      vname <- classvars[va]
      work_data <- work_data |>
        dplyr::filter(!is.na(.data[[vname]]))
    }
  }

  if (length(varsel) > 0) {
     summary_df <- work_data |>
      dplyr::group_by(dplyr::across(dplyr::all_of(varsel))) |>
      dplyr::summarise(
        `_FREQ_` = dplyr::n(),
        N = sum(!is.na(.data[[avars[1]]])),
        SUM = sum(.data[[avars[1]]], na.rm = TRUE),
        MEAN = mean(.data[[avars[1]]], na.rm = TRUE),
        .groups = "drop"
      )
  } else {
    # No class variables: overall summary
    summary_df <- work_data |>
      dplyr::summarise(
        `_FREQ_` = dplyr::n(),
        N = sum(!is.na(.data[[avars[1]]])),
        SUM = sum(.data[[avars[1]]], na.rm = TRUE),
        MEAN = mean(.data[[avars[1]]], na.rm = TRUE)
      )
  }

  type_val <- strtoi(bin_str, base = 2)
  summary_df <- summary_df |>
    dplyr::mutate(`_TYPE_` = as.character(type_val))

  combo_datasets[[i + 1]] <- summary_df
}
```

```
  outds <- dplyr::bind_rows(combo_datasets)

  return(outds)
}

carsnonnway <- construct_non_nway(
  inds = sashelp_cars,
  classvars = c("Origin", "Type", "DriveTrain", "Cylinders"),
  avars = "MSRP",
  missing = "missing"
)
```

Program 6. R program.

CONCLUSION

While it may seem like a wasted exercise to re-invent the wheel by taking what PROC SUMMARY gives us for free, it allows us to examine the NWAY option more closely and appreciate how performing this work in memory is much faster. It is concluded to fix the memory issue by implementing appropriate options to allow the program to use more memory than to try and perform each combination of class variables with separate PROC SUMMARY calls.

CONTACT INFORMATION

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

Brian Varney
Experis
269-365-1755
brian.varney@experis.com
www.experis.com