

## D is for Dynamic, putting Dynamic back into CDISC (A simple macro utilizing PROC SQL which auto-formats continuous variables for lab tables).

Steven Black, SynteractHCR, Carlsbad, CA

### ABSTRACT

As CDISC implementation increases, the ability and need for simple dynamic code has also risen. Using PROC SQL and the SAS MACRO language coupled with proper use of ADaM guidelines, generating dynamic code has become much easier. This paper illustrates one way of auto generating the decimal format needed for tables where continuous data is arranged by parameters. This code has been especially helpful with laboratory table values where the decimal formats change between each parameter. Understanding and using these same techniques and framework, additional dynamic code can be created allowing you to put the D back into CDISC.

### INTRODUCTION

Recently I came across a situation where the decimal formatting test results needed to be based on the format of the data as it was displayed within the AVAL variable. In the past (non ADaM) datasets I could use the format specified for the specific variable needed, such as systolic blood pressure or height. Now, however, especially with lab data, the variables are stacked one on top of another, and the decimal format varies within each parameter.

Manual Solution: I could look and see for each parameter what the decimal place was and then write some code per each parameter or group those parameters with similar decimal types and hardcode their respective decimal places needed per statistic as defined by the statistical analysis. It would take some time but it can be done.

However, if there is a logically consistent pattern to it, I can macrotize the program and create reusable code. For this example the Statistical Analysis Plan (SAP) had an essential general statement stating: 'Generally, means, medians and standard deviations will be presented with an increased level of precision as follows: means and medians will be presented to one more decimal place than the raw data, and the standard deviations will be presented to two more decimal places than the raw data.' This was the key to making this all work, otherwise hardcoding may be a better alternative or additional conditional macro language could be employed.

This paper will be divided up into several steps to illustrate the process and provide the background information needed for each step. The intent of this paper is to provide a succinct example of code and to provide enough information per each step to allow others to understand and then apply this knowledge to meet their own needs.

### GETTING STARTED

First I need to know what I want the table to look like in the end. In my case, I knew that I need a table generated for each lab category (hematology, chemistry, etc.) and within each of those lab categories I would have the specific lab parameters (WBC, Basophils, AST, ALT, etc.), separated out by visit and treatment group.

PharmaSUG, Inc.  
ALPHA-DRUG-001

Page 1 of x

Table 14.3.4.1  
Hematology  
Safety Population

Parameter(units)	TimePoint	ALPHA (N= )	BETA (N= )
WBC (10 <sup>9</sup> L)	Baseline [1]		
	n	n	n
	Mean (SD)	xx.x (xx.xxx)	xx.x (xx.xxx)
	Median	xx.x	xx.x
	Min, Max	xx, xx	xx, xx
	End of Treatment		
	n	n	n
	Mean (SD)	xx.x (xx.xxx)	xx.x (xx.xxx)
	Median	xx.x	xx.x
	Min, Max	xx, xx	xx, xx
	Change from Baseline to EOT		
	n	n	n
Mean (SD)	xx.x (xx.xxx)	xx.x (xx.xxx)	
Median	xx.x	xx.x	
Min, Max	xx, xx	xx, xx	
Basophils (%)	Baseline [1]		
	n	n	n
	Mean (SD)	xx.x (xx.xxx)	xx.x (xx.xxx)
	Median	xx.x	xx.x
	Min, Max	xx, xx	xx, xx

path\program.sas date time

Programmer note: Table will include the following hematology parameters: WBC (10<sup>9</sup> L), Basophils (%), Eosinophils (%), Lymphocytes (%), Monocytes (%), Total Neutrophils (%), Bands (%), Segmented Neutrophils (%), Platelets (10<sup>9</sup> L), RBC (10<sup>12</sup> L), Hemoglobin (g/L), Hematocrit (%), MCHC (g/L), MCH (pg), MCV (fL), RDW (%). Report data in SI units described.

D is for Dynamic, putting Dynamic back into CDISC, continued

Second I need to know what data I have to work with. I know that with the ADLB dataset, the is data set up as a very long dataset with each of the laboratory categories and lab parameters set on top of one other another, as seen below. I also know that the variable that I am actually analyzing (AVAL) will have a number of different decimal places per each parameter, and that the resulting tables will also have different decimal places or formats per each statistic (mean, median, standard deviation).

Category for Lab Test	Parameter Description	Parameter (N)	Parameter Code	Analysis Timepoint Description	Analysis Timepoint Number	Analysis Value
HEMATOLOGY	RDW (%)	14	RDW	Baseline	2	15.9
HEMATOLOGY	RDW (%)	14	RDW	End of Treatment	10	15.9
HBA1C	HbA1c (%)	15	HBA1C	Baseline	2	8.6
HBA1C	HbA1c (%)	15	HBA1C	End of Treatment	10	8.5
CHEMISTRY	ALT (U/L)	16	ALT	Baseline	2	24
CHEMISTRY	ALT (U/L)	16	ALT	End of Treatment	10	23
CHEMISTRY	AST (U/L)	17	AST	Baseline	2	20
CHEMISTRY	AST (U/L)	17	AST	End of Treatment	10	19
CHEMISTRY	Alkaline Phosphatase (U/L)	18	ALP	Baseline	2	122
CHEMISTRY	Alkaline Phosphatase (U/L)	18	ALP	End of Treatment	10	107
CHEMISTRY	Albumin (g/L)	19	ALB	Baseline	2	35
CHEMISTRY	Albumin (g/L)	19	ALB	End of Treatment	10	36
CHEMISTRY	GGT (U/L)	20	GGT	Baseline	2	54

Variables that we will use in the macro from our ADLB dataset: PARAMN, AVAL, AVISITN, USUBJID, LBCAT, and TRT01PN or TRT01AN).

Prior to the Macro and PROC SQL, I do a few minor clean-up steps to the ADLB data and call this new dataset `_PRE_FREQ`.

### STEP 1 – FIND THE DECIMALS WITH PROC SQL

A brief introduction to PROC SQL: Because of many useful properties of PROC SQL it has become essential to building dynamic SAS programs. Using PROC SQL allows you to simplify and complete many steps within one call. In this paper I will touch on a number of amazing abilities of this procedure but, as in all things in SAS, there is an iceberg of possibilities. PROC SQL uses a bit more English words (underlined) than normal dataset processing.

Let's start simple:

```
proc sql;
  create table _decil as
  select paramn, aval
  from _pre_freq;
quit;
```

So in this quick step we created a new dataset (or table) and called it `_DECIL1`, then we keep or selected the variables `PARAMN` and `AVAL`, from source dataset `_PRE_FREQ`. We then end or quit the procedure. Variables are separated by commas and we only put in semi colons when we are done with the statement. This allows us to use multiple statements in one procedure which we will make use of this capability later in the macro.

Next we'll use some SAS® functions on the `AVAL` variable to capture the length of the decimal places used of for each value.

```
create table _decil as
select paramn, aval ,strip(scan(put(aval,best.),2,',')) as deci
```

I used a `PUT` statement to convert `AVAL` to character then used a `SCAN` function to look for the text after the decimal. In the dataset we would have done something similar with the code `"DECI= strip(..."` in PROC SQL we use the term `'as'` instead of the equal sign. The new variable now looks like this

Parameter (N)	Analysis Value	DECI
31	22	
31	20	
32	1.19473	19473
32	1.03328	03328
32	0.9687	9687
32	1.42076	42076

Next let's create a another variable which captures the length of the variable `DECI`, and call it `LENGTH`, but this time we need to add some additional logic and use this newly created variable `DECI`. To accomplish this we will use a `CASE` Expression.

D is for Dynamic, putting Dynamic back into CDISC, continued

```

case when calculated deci='' then 0
else length(calculated deci)
end as length

```

Using a CASE expression allows us to add 'when then' logic to our procedure. In this case, we are calculating the length of the variable DECI but first want to modify the new variable based on the presence or absence of data. Again it's near English in code: when DECI equals null then set this new variable to 0, else this new variable = length of DECI, at the end we specify what this new variable is to be called again using the 'AS' text followed by the new variable name (LENGTH in this case).

The keyword CALCULATED is needed due to the way PROC SQL process the data. The variable DECI is not contained in the \_PRE\_FREQ dataset (it is being created within the procedure), so to access this new variable we add the keyword CALCULATED prior to calling the variable. This way PROC SQL knows not to look in the dataset but in the temporary running dataset.

```

proc sql;
create table _dec1 as
select paramn, aval ,strip(scan(put(aval,best.),2,',')) as deci,
case when calculated deci='' then 0
else length(calculated deci)
end as length
from _pre_freq;
quit;

```

Parameter (N)	Analysis Value	DECI	LENGTH
31	20		0
32	1.19473	19473	5
32	1.03328	03328	5
32	0.9687	9687	4
32	1.42076	42076	5
32	1.00099	00099	5

## STEP 2 – CAPTURE THE MAX LENGTH USED PER PARAMETER

Now we need to find out what the maximum length is per parameter, again we can do this quite efficiently using PROC SQL and do it within the same procedure. We will use the same style of programming to adjust some of the max lengths to fit our needs. This next code would fit between the last semi colon and the quit statement of the previous code segment above.

```

create table _dec2 as
select paramn, max(length) as maxl,
case when calculated maxl >3 then 4
else calculated maxl
end as max_length
from _dec1
group by paramn;

```

Here we need to create a new table called \_DEC2 where we are selecting the variable PARAMN, and creating a new variable which captures the maximum value of length called MAXL. We again use the CASE expression to create a modified version of that maximum length variable resetting it to 4 if the value is greater than 3. This modification is done as to present the data in a more visually appealing way as it is difficult to fit and view decimal places greater than 4. This new variable is called MAX\_LENGTH. We are now pulling the data from \_DEC1 and use a 'group by' term to capture the max length per each PARAMN. The results of this new table look like this:

MAXL	Parameter (N)	MAX_LENGTH
2	24	2
3	25	3
0	26	0
1	27	1
0	28	0
3	29	3
3	30	3
0	31	0
5	32	4
5	33	4
4	34	4

### STEP 3 – CAPTURE THE NUMBER OF TYPES OF DECIMAL PLACES

In ADLB data there are many parameters and in these past two steps we calculated the max decimal place per parameter but we'll also need to categorize parameters into types, so we can loop through of decimal lengths. To accomplish all this we can again use PROC SQL and keep this code all within the same procedure we have created. We can place this code right before the quit statement and the last semi colon.

```
select distinct(max_length) into: type1 -: type5
from _deci2;
%let n_types=&sqllobs;
```

In the above step we are selecting the variable MAX\_LENGTH from \_DECI2 but this time we only want the DISTINCT values of the variable, similar to performing a “first dot” Or a “nodupkey” statement but it's not important if it's the first or the last observation, ensuring that there are no other values like it in the dataset. Secondly we are using PROC SQL's amazing ability to create MACRO variables. We accomplish this by using the keyword INTO in conjunction with a colon as “INTO:”. We then create our new macro variable names, in this case ranging from 1 to 5. because of my limiting the number of decimals to 4 I know that I'll not get above 5, you can also set your max number to 99 or any numerical value. I will then create macro variables type1 – type5. I want to know my highest number that the macro resolves to so I use the temporary variable &SQLOBS and store this in a permanent macro variable called N\_TYPES using a %LET statement. The values of these macro variables automatically get pushed out to the output window, if data in the output window is not wanted the NOPRINT option can be used, placed next to PROC SQL.

#### MAX\_LENGTH

```
0
1
2
3
4
```

Output Window Result of MAX\_LENGTH

### STEP 4 – GROUPING THE PARAMETERS INTO DECIMAL TYPES

Now that we know the range of decimals available and we have them stored as macro variables we can create the decimal formats for all grouped parameters at the same time. To do this we will further utilize the power of PROC SQL and the MACRO language. We will also utilize a macro DO loop to iterate through each of the decimal types. Macro DO loops need to be run within a macro, so we will place our ever growing single PROC SQL procedure within a macro called DYNAMIC.

Basic MACRO principles: %MACRO MACRO\_NAME starts a macro, %MEND ends the macro, %MACRO\_NAME runs/executes the macro. Generally macro variables created within the macro only live within that specific macro (local) unless otherwise specified using %global option. Macro DO loops have % before each key word (%do, %end, %to, %then, %else). A %DO needs a corresponding %END statement. Macro variables that are being resolved more than once need multiple ampersands.

```
%macro dynamic;
  proc sql;
    create table _decil as
    ...
    %let n_types=&sqllobs;

    %do x= 1 %to &n_types;
      select paramn into: mparams&x separated by ','
      from _deci2 having max_length =&&type&x;
    %end;

  %quit;
%mend;
%dynamic;
```

I use a macro DO loop to move through each of the types of decimal places found using the &n\_types macro variable created in the step above and using X as my macro counter variable. Within the loop I use the similar PROC SQL code for selecting the variables wanted and creating a new macro variable mparams&x which will be equal to mparams1 –mparams5 once resolved. I also use the SEPARATED BY statement to allow me to capture all of the

D is for Dynamic, putting Dynamic back into CDISC, continued

values of the PARAMN variable into one macro term. I further specify that I want these values separated by a comma. I establish where I am pulling my data from and also incorporate a HAVING statement. This HAVING statement allows for the limiting of groups of data based on the logic used. The WHERE clause works on individual rows of data, but in our example we need the PARAM to be grouped into groups based on the max length, so the HAVING clause works perfect.

Per each loop the macro variable `&&type&x` is resolved twice, for `x=1` first it is resolved from `&&type&x` to `&type1` then `&type1` resolves to `0`. When the loop completes, the macro will end and the new `mparams1 - mparams5` macro variables will contain all the parameters per decimal group and be available to use.

## STEP 5 – CREATE STATISTICAL OUTPUT

Once we have created these macro variables based off of the data in the ADLB dataset we can now calculate the statistics needed, per the SAP. This can be done using PROC UNIVARIATE, PROC MEANS, PROC TTEST, even PROC SQL! Once data from the procedure has been output we can create dynamic formats. Below is a basic example of a PROC UNIVARIATE outputting to a new dataset `_STAT_V1`. Note that the data had been previously sorted by PARAMN, TXGROUP and AVISITN.

```
proc univariate data=_pre_freq noprint;
class txgroup avisitn;
var aval;
output n=n mean=mean median=median std=std min=min max=max out=_stat_v1;
by paramn;
where avisitn in (2,10);
run;
```

### Resulting Data from Statistical Output

Parameter (N)	TXGROUP	Analysis Timepoint Number	number of nonmissing values, AVAL	the mean, AVAL	the standard deviation, AVAL	the largest value, AVAL	the median, AVAL	the smallest value, AVAL
1	1	2	38	8.2526316	2.021738193	12.9	8.2	4
1	1	10	36	8.5472222	2.460486144	14.6	8.15	3.8
1	2	2	37	7.9486486	1.966048154	12.1	7.8	5.1
1	2	10	35	7.6542857	1.743617412	11.9	7.7	3.9
2	1	2	38	0.4236842	0.404324068	2.6	0.4	0
2	1	10	36	0.4055556	0.181964588	0.8	0.4	0.1
2	2	2	37	0.3972973	0.252197549	1.5	0.3	0.1
2	2	10	35	0.3914286	0.235611131	1	0.3	0
3	1	2	38	2.0026316	1.215994053	5.4	1.8	0.4
3	1	10	36	2.3388889	1.67387963	7.6	1.9	0.3
3	2	2	37	2.5054054	2.468866779	12.3	1.6	0.4
3	2	10	35	2.3914286	2.145086236	10.6	1.7	0.1

## STEP 6 – FINALIZE THE FORMATS FROM STATISTICAL OUTPUT

In this last step we will set the statistical output datasets together and then create needed variables in proper order so that we can later transpose them as per the table shell. Much of this is dependent upon the style of the programmer however the basic principles remain. In my tables I create a `Y_AXIS` and `X_AXIS` variable these variables assist in how I transpose the data later. `Y_AXIS` will determine the order of my rows and `X_AXIS` determines the order of my columns. Below is the basic code used:

```
data _means (keep=paramn avisitn y_axis display x_axis); length display $20;
set _stat_v1 (in=a);

*** create x_axis ***;
x_axis=txgroup;

*** create y_axis for var1 ***;
y_axis=10;
display=compress(put(n,best.));
output;
y_axis=11;
display= strip(put(mean,10.1))||" ("||strip(put(std,10.2))||")";
output;
y_axis=12;
display=compress(put(median,10.1));
output;
y_axis=13;
```

D is for Dynamic, putting Dynamic back into CDISC, continued

```
display= strip(put(min,10.0))||", "||strip(put(max,10.0));
output;
run;
```

In this step I create the X\_AXIS and Y\_AXIS variables. The display variable contains the formatted values of the various statistics needed with a preset length of 20. These variables are output to the \_MEANS dataset.

Now let's add some dynamics: First we will need to loop through each of the decimal types using a macro DO loop, then we will limit the processing to only those parameters that meet that decimal type. Finally we will modify the formats used in the PUT statements based on the type used. All this code would be placed inside the %dynamic macro.

```
data _means; length display $20;
set _...

%do x= 1 %to &n_types;
  if paramn in (&&params&x) then do;
    y_axis=10;
    display=compress(put(n,best.));
    output;
    y_axis=11;
    display=strip(put(mean,10.%eval(&&type&x+1))||" ("||
strip(put(std,10.%eval(&&type&x+2))||"");
    output;
    y_axis=12;
    display=compress(put(median,10.%eval(&&type&x+1)));
    output;
    y_axis=13;
    display=strip(put(min,10.&&type&x))||", "|| strip(put(max,10.&&type&x));
    output;
  end;
%end;
run;

%mend;
%dynamic;
```

We use the macro variable &&params&x which equal a list of the parameter values per type of decimal count. Then using the values for &&type&x (values:0-4) we can determine the decimal places for each of the put statements. Using the SAP as a guide, we place the mean and median decimal place up one from the original data, standard deviation (STD) is +2, whereas the min and max equal the value found in the data. Once the macro loops through each type all parameters will have the correct format applied and we can move on to transposing the data.

## STEP 7 - MAKE IT DYNAMIC

Throughout the program we have referenced different dataset names and variables names that we can now turn into macro variables and specify them as parameters in a macro call. We have already done much of the work when we created the %DYNAMIC Macro. We will now create with the macro parameters at the top and bottom of the macro.

```
%macro dynamic (orig_data, stat_data, out_data, txgroup)

...

%mend;
%dynamic (_pre_freq, %str(_stat_v1 (in=a)), _means, trt01pn);
```

In this example we are creating a new macro variable called ORIG\_DATA and setting the value of this new parameter to \_PRE\_FREQ. In some cases character strings such as '=,%/' may be wanted within the macro parameter, using %STR option will mask these character strings during compilation of a macro. Next I'll replace those specific parameters with the macro variable assigned. Now when I run the %DYNAMIC macro I have the ability to modify the parameters to my own preference and still be able to run it.

If further values need to be run such change from baseline we can simply replace the AVAL variable with a macro

D is for Dynamic, putting Dynamic back into CDISC, continued

parameter value and add this to the macro to modify the statistic used or we can create a new statistical output dataset and reference this dataset in the macro call.

DISPLAY	Y_AVS	X_AVS	Parameter (N)	Analysis Timepoint Number	number of nonmissing values, AVAL	the mean, AVAL	the standard deviation, AVAL	the largest value, AVAL	the median, AVAL	the smallest value, AVAL
96.162	13	1	9	10	36	130.8055556	16.758485176	162	128.5	96
37	10	2	9	2	37	129.54054054	13.959454372	167	128	105
129.5(13.96)	11	2	9	2	37	129.54054054	13.959454372	167	128	105
128.0	12	2	9	2	37	129.54054054	13.959454372	167	128	105
105.167	13	2	9	2	37	129.54054054	13.959454372	167	128	105
35	10	2	9	10	35	129.45714286	14.745573328	164	129	98
129.5(14.75)	11	2	9	10	35	129.45714286	14.745573328	164	129	98
129.0	12	2	9	10	35	129.45714286	14.745573328	164	129	98
98.164	13	2	9	10	35	129.45714286	14.745573328	164	129	98
38	10	1	10	2	38	40.857894737	4.3492547718	48.7	40.55	31.6
40.86(4.349)	11	1	10	2	38	40.857894737	4.3492547718	48.7	40.55	31.6
40.55	12	1	10	2	38	40.857894737	4.3492547718	48.7	40.55	31.6
31.6.48.7	13	1	10	2	38	40.857894737	4.3492547718	48.7	40.55	31.6
36	10	1	10	10	36	40.191666667	5.1621354939	50	40.45	28.6
40.19(5.162)	11	1	10	10	36	40.191666667	5.1621354939	50	40.45	28.6
40.45	12	1	10	10	36	40.191666667	5.1621354939	50	40.45	28.6
28.6.50.0	13	1	10	10	36	40.191666667	5.1621354939	50	40.45	28.6
37	10	2	10	2	37	39.789189189	4.3704680517	50.2	39.8	32.3
39.79(4.370)	11	2	10	2	37	39.789189189	4.3704680517	50.2	39.8	32.3
39.80	12	2	10	2	37	39.789189189	4.3704680517	50.2	39.8	32.3
32.3.50.2	13	2	10	2	37	39.789189189	4.3704680517	50.2	39.8	32.3
35	10	2	10	10	35	40.071428571	4.5031734842	50.5	39.9	31
40.07(4.503)	11	2	10	10	35	40.071428571	4.5031734842	50.5	39.9	31
39.90	12	2	10	10	35	40.071428571	4.5031734842	50.5	39.9	31
31.0.50.5	13	2	10	10	35	40.071428571	4.5031734842	50.5	39.9	31
38	10	1	11	2	38	32.805263158	0.7555013973	34.7	32.85	30.6
32.81(0.756)	11	1	11	2	38	32.805263158	0.7555013973	34.7	32.85	30.6
32.85	12	1	11	2	38	32.805263158	0.7555013973	34.7	32.85	30.6
30.6.34.7	13	1	11	2	38	32.805263158	0.7555013973	34.7	32.85	30.6

## CONCLUSION

This paper presents a useful macro for dynamically creating formats for lab tables based off of the data found in the AVAL variable per parameter. It also includes specific information regarding the benefits of using PROC SQL combined with the MACRO language. By using the information and SAS code in this paper it is hoped that many other dynamic applications can be created.

## REFERENCES

Lafler Kirk Paul.2004. PROC SQL: Beyond the Basics using SAS®. Cary, NC: SAS Institute Inc.

Carpenter Arthur L.2007. "Building Dynamic Programs and Applications Using the SAS® Macro Language." SAS Seminar Publication. Vista, CA: CALOXY Inc.

## ACKNOWLEDGMENTS

Many thanks to Priya Venkata and Kurtis Voris for their willingness to listen, and provide valuable insight and feedback.

## CONTACT INFORMATION

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

Steve Black  
 SynteractHCR  
 5759 Fleet St. Suite 100  
 Carlsbad, CA 92081  
 760-268-8015  
 steven.c.black@gmail.com

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.

Other brand and product names are trademarks of their respective companies.

## APPENDIX

Full code used:

```
%macro dynamic (orig_data, stat_data, out_data, txgroup);
```

D is for Dynamic, putting Dynamic back into CDISC, continued

```
*** use sql to get max decimal places per paramn ***;

proc sql;

*** capture length of data after decimal place ***;

create table _dec1 as
select paramn, aval ,strip(scan(put(aval,best.),2,',')) as deci,
case when calculated deci=' ' then 0
else length(calculated deci)
end as length
from &orig_data;

*** capture max length per paramn if >3 then set to 4 ***;

create table _dec2
as select max(length) as maxl, paramn,
case when calculated maxl >3 then 4
else calculated maxl
end as max_length
from _dec1
group by paramn;

*** capture number of types of decimal places ***;

select distinct(max_length) into: type1 -: type5
from _dec2;
%let n_types=&sqllobs;

*** macroize paramns per decimal place type ***;

%do x= 1 %to &n_types;
    select paramn into: mparams&x separated by ','
    from _dec2 having max_length =&&type&x;
%end;

quit;

*** optional to have within the macro ***;

proc sort data=&orig_data;
by paramn &txgroup avisitn;
run;

proc univariate data=&orig_data noprint;
class &txgroup avisitn;
var aval;
output n=n mean=mean median=median std=std min=min max=max out=&stat_data;
by paramn;
where avisitn in (2,10);
run;

*** finalize statistical data ***;

data &out_data; length display $20 y_axis x_axis 3;
set &stat_data;
```

D is for Dynamic, putting Dynamic back into CDISC, continued

```
x_axis=&txgroup;

%do x= 1 %to &n_types;
  if paramn in (&&params&x) then do;
    y_axis=10;
    display=compress(put(n,best.));
    output;
    y_axis=11;
    display=strip(put(mean,10.%eval(&&type&x+1)))||" ("||
    strip(put(std,10.%eval(&&type&x+2)))||")";
    output;
    y_axis=12;
    display=compress(put(median,10.%eval(&&type&x+1)));
    output;
    y_axis=13;
    display=strip(put(min,10.&&type&x))||", "||
    strip(put(max,10.&&type&x));
    output;
  end;
%end;
run;

%mend;
%dynamic (_pre_freq, _stat_v1, _means, trt01pn);
```