Paper RIV-06

SBSBOXPLOT: A SAS® Macro for Generating Side-by-Side Boxplots

Jason A. Schoeneberger, Grant B. Morgan, Bethany A. Bell
University of South Carolina

ABSTRACT

Good research practice includes a basic exploration of data prior to engaging in more sophisticated, inferential analyses. Generating location and dispersion summary statistics such as the mean or median and standard deviation or interquartile range help describe variable distributions. Whereas these descriptive statistics can be reported in tabular or narrative format, they can also be reported visually. One efficient graphical display that clearly communicates a host of summary statistics, including the mean, median, 10^{th}, 25^{th}, 75^{th}, and 90^{th} percentiles, and the minimum and maximum values is the box-and-whisker plot (Tukey, 1977), often simply referred to as a boxplot. Thus, when examining a boxplot, a reader can quickly assess measures of central tendency and dispersion as well as the shape of the distribution for any given continuous variable. Moreover, side-by-side boxplots allow easy communication of patterns evident in data by viewing graphical displays in clusters or blocks to accentuate relationships among categorical or discrete variables or factors (e.g., examining standardized test scores by subject and student race/ethnicity or adolescent BMI by student gender and age). However, the SAS® programming necessary to generate side-by-side boxplots is not intuitive and cumbersome at best. To ease this burden, using various data management procedures in conjunction with PROC BOXPLOT, SBSBOXPLOT was developed to facilitate the generation of side-by-side boxplots for displaying distributional information disaggregated by factors important for a basic understanding of one's data. This paper provides the macro programming language, as well as results from an executed example of the macro.

Keywords: PROC BOXPLOT, BASE SAS®, SAS/GRAPH, MACRO, Reporting and Information Visualization

INTRODUCTION

Good research practice includes a basic exploration of data prior to engaging in more sophisticated, inferential analyses. Except for instances where entry is extremely well-controlled, the collection point for most data involves human input, which is subject to error. As such, data including missing values, both valid and invalid outliers, and erroneous entries can be messy. Prior exploration provides the researcher the opportunity to better understand and clean her or his data before further summary, analysis or interpretation.

A number of different strategies are available to researchers when beginning an initial exploration of newly acquired data. A procedure providing summary descriptive information, such as that obtained with PROC UNIVARIATE or PROC MEANS, allows a researcher to easily diagnose features associated with variables in her or his data, including means, standard deviations or variances, minimum and maximum values, or levels of skewness or kurtosis. This type of summary information is particularly useful when dealing with continuously distributed variables. When faced with discrete categorical data, examination of output generated by PROC FREQ quickly informs the analyst how many categories are present and the frequency with which those categories are represented in the data. Generally, both of these procedures generate numerical summaries in tabular format that may not be easily understood by consumers who may be less familiar with the data of interest.

Well-constructed graphical summaries, on the other hand, have the potential to provide clear and concise information that is more easily understood by a larger audience. Graphical summaries can also be useful during the initial exploration of data. That is, quickly allowing the researcher to diagnose potential problems or interesting trends without having to sift through the tabular data generated by the procedures discussed above. Information similar to that generated by PROC FREQ for discrete category data can also be displayed via a pie graph or bar chart using PROC GCHART. In addition, numerous graphical displays are available for summarizing continuously distributed data, including histograms (via PROC UNIVARIATE), boxplots (PROC BOXPLOT), line graphs (PROC GPLOT), or stem-and-leaf plots via the PLOT option in PROC UNIVARIATE.

Periodically, researchers are interested in exploring distributions of variables disaggregated by other variables available to them. When handling discrete categorical data, a crosstabulation table available through PROC FREQ facilitates understanding of patterns among groups. For continuously distributed data, researchers can make use of graphical displays such as scatterplots (via PROC GPLOT), comparative histograms using the NROWS command (via PROC UNIVARIATE) or multiple boxplots using the BY command (via PROC BOXPLOT). Unfortunately, this latter option does not provide a concise graphical display containing all of the information for the respective outcome and independent variables of interest in one graph. Inspection and interpretation of the results are more difficult than necessary as the number of outcome and independent variables increases. Instead, if a researcher wanted to

generate side-by-side boxplots to examine the distribution of some continuous variable (e.g., standardized test scores) as a function of two categorical variables (e.g., test subject and student race/ethnicity) extra work is required, including a rather convoluted data management process that is less than intuitive. To help ease this burden, using various data management procedures in conjunction with PROC BOXPLOT, the SAS® macro, SBSBOXPLOT, was developed to facilitate the generation of side-by-side boxplots for displaying distributional information disaggregated by factors important for a basic understanding of one's data.

MACRO SBSBOXPLOT DETAILS

SBSBOXPLOT works with cluster variables that have a maximum of five levels and has a total of ten macro arguments the user can supply to generate the side-by-side boxplot graphs. Here each argument is defined, including any rules that must be adhered to when supplying information to the macro.

path: folder location where data file for analysis resides

data_in: this argument is the library reference and name of the data file containing the data to be plotted in the boxplot graph. It is assumed that the library reference has already been established prior to running the macro. This file should be structured to contain one record per intended unit of analysis.

outcomes: the outcomes argument contains the dependent variables to be plotted within a comparison variable. Each of the outcomes should be listed, separated by a space (e.g. math reading science).

clustvar: the clustvar argument contains the cluster/diaggregation variable to be used in making comparisons among the outcome variables (i.e., the variable that will be plotted in the face of the graph).

xaxis_lbl: this argument contains the label to be applied to the x-axis of the boxplot graph. This argument should be supplied within quotes (e.g. "Ethnicity").

yaxis_lbl: this argument contains the label to be applied to the y-axis of the boxplot graph. This argument should be supplied within quotes (e.g. "Mean Standardized Score").

legend_lbl: this argument contains the label describing the legend displayed at the bottom of the boxplot graph. This argument should be supplied within quotes (e.g. "Test Subject").

box_lbl: this argument contains the labels for each of the boxes depicting the outcome data. Each label should be supplied within quotes separated by a '!' exclamation point (e.g. "Math"!"Reading"!"Science").

box_clr: this argument contains the colors for each of the boxes depicting the outcome data. Each color should be supplied separated by a space (e.g. red blue green).

vertref: this argument can contain a number within range of the scale of the outcome(s) variable. At the point on the graph supplied in this argument, a horizontal reference line will be drawn for a reference point. This can be useful for denoting cut-points or locations of desired outcomes. (e.g. for a scale of 0 to 100, a value of 80 would draw a horizontal reference line at 80). If no reference line is desired, leave blank.

flipflop: this argument allows the user to switch the variables representing the clusters and box labels, in effect flip-flopping the graph to explore interesting trends (i.e., changing the original cluster variable to be the independent variable plotted on the x-axis and the original x-axis variable to be the cluster variable). The default value for this argument is 'N', and can be turned on to 'Y' after an initial run through the macro has been executed under the default setting 'N'.

Below is an example of the SBSBOXPLOT macro, along with technical information to help the user understand the various steps within the macro. For this illustration, the outcome variable is mean standardized test scores, the independent variable is test subject (i.e., math, reading, science) and the cluster variable that appears in the face of the graph is student race/ethnicity.

A complete version of the macro, sans the technical discussions provided here, is also provided at the end of the paper. Similarly, the macro can also be downloaded from the third author's website (http://www.ed.sc.edu/bell/).

The macro begins by establishing each of the arguments described above.

```
**the macro arguments are defined;
%macro sbsboxplot
(path,data_in,outcomes,clustvar,xaxis_lbl,yaxis_lbl,legend_lbl,box_lbl,box_clr,vertr
ef,flipflop);
```

The PROC PRINTTO below sends the contents of the Output and Log files to the C-drive on the local machine in case the user needs to examine. Otherwise, only the boxplot is generated as visual output.

```
**this prints all output and log information to these locations if the user needs to
examine;
proc printto print=" %bquote(&path)SBSBOXPLOT_Output.lst" log="
%bquote(&path)SBSBOXPLOT_Log.log" new;
run;
```

The macro begins by scanning the parameters supplied by the user, creating macro calls used later in processing. In addition, the order of the outcome variables is taken into account and maintained, with box colors and labels

assigned in accordance with the order of the outcome variables. The flipflop="N" ensures that this code is utilized on an initial run of the macro. Separate coding is called for when the user wishes to exchange outcome and grouping variables.

```
**this section scans the macro arguments and creates macro calls;
**creates order of outcomes, symbol statements, box colors and box labels;
%if "&flipflop"="N" %then %do;
     %local i outcomes symbols box_color box_label n;
     %let i=1;
     %let symbols=triangle square circle diamond plus hash X % =;
     %let outone=%scan(&outcomes, &i);
     %let sym=%scan(&symbols, &i);
     %let box_color=%scan(&box_clr, &i);
     %let box_label=%scan(&box_lbl, &i,' ');
     %do %while (&outone^=);
            %let outsort&i=%nrstr(%sortcall)%str((&outone,&i)););
            %let symstat&i=%nrstr(%symcall)%str((&i,&sym););
            %let color&i=&box_color;
            %let label&i=%str(&i=&box_label);
            %let i=%eval(&i + 1);
            %let outone=%scan(&outcomes, &i);
            %let sym=%scan(&symbols, &i);
            %let box_color=%scan(&box_clr, &i);
            %let box_label=%scan(&box_lbl, &i,' ');
            %let n=%eval(&i - 1);
     %end;
```

Here a data file is created in the work directory based on the data_in argument supplied by the user during the initial SBSBOXPLOT run (when flipflop='N'). The outcome variables (outcomes argument) and cluster variable (clustvar argument) and id variable are retained. A screenshot of the resulting data file is provided below in Figure 1. The data are organized in a flat file form with one line per student (id) and test scores organized as separate variables (columns).

```
**this section of code is run under the default when flipflop=N;
**first, we create the sbs data file in the work directory with only the necessary
variables;
data sbs;
     set "%bquote(&path)%bquote(&data_in)";
     keep id &outcomes &clustvar; id+1;
     run;
```

	race	math	reading	science	id
1	Hispanic	-1.663684356	0.0432328103	1.5481116976	140
2	Black	-1.652830486	-1.677108752	-0.842328047	2
3	Black	-1.652830486	0.7484027528	-1.088213543	60
4	White	-1.649709705	-0.145513776	1.4496571541	257
5	Black	-1.624017306	-1.478848038	-1.2504645	7
6	Black	-1.624017306	0.5513437551	-1.310956144	54
7	Hispanic	-1.610333839	-0.812775521	1.7733765516	126
8	Hispanic	-1.582776438	-1.073556618	1.5907751137	115
9	White	-1.558148672	-0.274370199	0.4361102637	251
10	Black	-1.506045222	-0.412563633	-1.010271102	30
11	Black	-1.506045222	1.4986331671	-0.363764513	91
12	White	-1.490939695	0.3400445461	0.9783347887	273
13	White	-1.458976292	-0.00729288	0.9316694402	262
14	White	-1.447058656	-1.186467863	0.8040146231	219

Figure 1. Screenshot of input data file.

Data are then sorted by the generated id variable and the values of the cluster variable in ascending order. PROC TRANSPOSE is used to turn the data file on its end so that outcome values are stacked within cluster values in a hierarchically structured data file, converting the initial wide file to a tall file. This nested structure is necessary for

'tricking' SAS® into treating the data appropriately so that the side-by-side boxplots can be generated. The resulting data file can be seen in Figure 2 where there are now multiple lines per student (id) and the individual test scores are organized in a single variable labeled mean standardized score.

```
**transpose data vertically to stack outcomes of interest;
proc sort data=sbs;
    by id &clustvar;
proc transpose data=sbs out=sbsv name=outcome_type prefix=outcome_val;
    by id &clustvar;
    run;
```

	id	race	NAME OF FORMER VARIABLE	LABEL OF FORMER VARIABLE	Mean Standardized Score
1	1	Black	math	math	-1.347056082
2	1	Black	reading	reading	-1.696439105
3	1	Black	science	science	-0.450319838
4	2	Black	math	math	-1.652830486
5	2	Black	reading	reading	-1.677108752
6	2	Black	science	science	-0.842328047
7	3	Black	math	math	-0.945257836
8	3	Black	reading	reading	-1.640685029
9	3	Black	science	science	-1.054188685
10	4	Black	math	math	0.2315091227
11	4	Black	reading	reading	-1.587086046
12	4	Black	science	science	1.1700864031
13	5	Black	math	math	-0.277667444
14	5	Black	reading	reading	-1.566714685
15	5	Black	science	science	-1.33081129

Figure 2. Screenshot of transposed data file with nested structure.

Now we use PROC SQL to create a table holding unique outcome variables is created. Subsequently, a short macro created through our %SCAN routines above creates a variable denoting the order outcomes were supplied by the user.

```
**begin creating code to set up outcome codes dynamically;
**this sql will create a table of unique outcome values and associated counts;
proc sql;
     create table out_tab as
     select unique(outcome_type)
     from sbsv
     order by outcome_type;
     quit;

**this macro will create an outcome_sort variable to maintain the order of outcomes
as provided by user;
%macro sortcall (arg,arg2);

**calculate outcome_sort;
data out_tab;
     set out_tab;
     if (outcome_type="&arg") then outcome_sort=&arg2;
     run;
**end macro;
%mend sortcall;

**this code generates the macro statements from the outsort macro equal to n-
times=the number of outcomes;
%do i=1 %to &n;
     %unquote(&&outsort&i);
%end;
```

At this point, the macro creates syntax to generate outcome codes for each outcome and to define the color(s) associated with each outcome(s). Using PROC SQL, each of the values in the variables containing syntax for

creating outcome codes and color definitions are inserted into macro variables outc (for outcome codes) and outb (for box color codes). The resulting code reads “if (outcome_type=’math’) then outcome=1;” as an example.

```
**this data step will create outcomes and write code for outcomes and boxcolors;
data out_tab;
    set out_tab;
    outcode=cat("if (outcome_type='",trim(outcome_type),"') then
    outcome=",outcome_sort,";");
    boxcode=cat("if (outcome=",outcome_sort,") then
    color=",%quote(cat('"&&color',outcome_sort,'"')),";");
    run;

**this sql will create macro variables holding the code created in the previous data
step for outcome and colors;
proc sql;
    select unique(outcode), (boxcode)
    into : outc separated by "0D0A"X,
         : outb separated by "0D0A"X
    from out_tab
    order by outcode;
    quit;
```

The data step that follows creates variables necessary for formatting the side-by-side plots and makes use of the dynamic coding shown above to establish the outcome and color variables.

```
**create necessary variables for side-by-side plots;
data sbsv;
    set sbsv;
    **set length of color variable;
    length color $25;
    **creates block variable for box fill on h-axis (see cblockvar);
    block="white";
    **creates outcome variable - put macro variable &outc here;
    &outc;
    **set colors - put macro variable &outb here;
    &outb;
    run;
```

In order to create separation between the clusters, we have to trick SAS® into plotting a blank space on the x-axis of the boxplot graph between each unique cluster value. To accomplish this, we use a coding scheme that skips a consecutive number after the last outcome variable within each unique value of the cluster variable. For example, if we are plotting three outcome variables within two clusters, we need category counts such as 1, 2, 3 then 5, 6 and 7, where 4 was skipped to create separation between the first and second cluster.

```
 **create code to set up categories dynamically, with skip patterns for clustering;
 **this sql will create a table of unique combination variables, number of unique
 outcomes;
 **count of categories needed (including skips) and a count of combinations;
 proc sql;
    create table cat_tab as
    select &clustvar, outcome,
           (count(*)+(count(*)/(count(distinct(outcome)))-1)) as cat_count,
           monotonic() as combo_count
    from (select unique(&clustvar|outcome), &clustvar, outcome from sbsv)
    order by &clustvar, outcome;
    quit;
```

This piece of code here is executed when flipflop is equal to ‘N’. This creates a table in the work directory that holds the unique values of the cluster variable for use if the user wants to generate the flip-flopped graph.

```
**this sql creates a small table of unique values for use in flipflop=Y mode;
**this code should only be run when flipflop="N";
**the values are placed into a macro argument called box_lblff that will be used as
box_lbl in flipflop;
%if "&flipflop"="N" %then %do;
proc sql;
```

```
    create table cat_tabff as
    select unique(&clustvar)
    from sbsv;
    quit;
%end;
```

Here we use PROC SQL to generate the number of category counts and combination counts for use in generating the appropriate skip pattern given the data.

```
**this sql creates the total number of category counts;
proc sql;
    create table cat_tab as
    select &clustvar, outcome,
           (count(*)+(count(*)/(count(distinct(outcome)))-1)) as cat_count,
           count(distinct(outcome)) as outcome_count,
           monotonic() as combo_count
    from cat_tab
    order by &clustvar, outcome;
    quit;
```

It is possible the users may wish to plot outcome variables within either a numeric or character cluster variable. To create the skip-coding necessary to insert the spaces between clusters, we need to write the code to handle both types of variables. Here a short macro is used to identify the cluster variable type. Subsequently, code is written to create unique skip-coding when the variable is character or when it is numeric. Finally, the written code is stored in a single macro variable comboc.

```
**this macro retrieves the variable type of the clustvar for dynamically writing
code to create categories;
**here, we set clusttype=C for character and N for numeric;
%macro gettype(table);
%global clusttype;
%let dsid=%sysfunc(open(sbsv,i));
     %do i=1 %to %sysfunc(attrn(&dsid,nvars));
          %if %upcase(&clustvar) = %upcase(%sysfunc(varname(&dsid,&i))) %then %let
          clusttype=%sysfunc(vartype(&dsid,&i));
     %end;
%let rc=%sysfunc(close(&dsid));

**this data step will write the code necessary to create the groups;
**ultimately, the obs variable is used to write the code to create the appropriate
skip pattern;
data cat_code_tab;
    set cat_tab;
    by &clustvar;
    if first.&clustvar then clustcount+1;
    obs_lag=lag(combo_count);
    obs=sum(clustcount,obs_lag);
    %if "&clusttype"="C" %then %do;
          combo=cat("if (&clustvar='",trim(&clustvar),"') and (outcome=",outcome,")
          then clustnum=",obs,";");
    %end;
    %if "&clusttype"="N" %then %do;
          combo=cat("if (&clustvar=",trim(&clustvar),") and (outcome=",outcome,")
          then clustnum=",obs,";");
    %end;
    run;
%mend gettype;
%gettype(table=sbsv);

**this sql will create a macro variable holding the code created in the previous
data step;
proc sql;
    select unique(combo)
    into : comboc separated by "0D0A"X
    from cat_code_tab
    order by combo;
```

```
    quit;
```

The PROC FORMAT statement below creates the blank spaces between clusters and applies the dynamic code for the box labels. Finally, the data step applies the formats to the cluster and outcome variables.

```
**establish formats - boxgr is to make a format for the new variable that has blanks
for the space between adjacent clusters or blocks;
**this section sets labels equal to each separate supplied label along with the
number in range;
proc format;
    value boxgr
    1-50=' ';
    value outcome
    %do i=1 %to &n;
          %unquote(&&label&i)
    %end;;
```

The following data step calls the comboc macro variable that contains the dynamic code written to create the skip-codes necessary to insert spaces between each outcome-by-cluster combination. In addition, the user-supplied label for the legend, x-axis label and y-axis label are applied. The formats for the skip-pattern and box labels are also applied.

```
**use data step to call dynamically written code;
data sbsv;
    set sbsv;
    **put macro variable here;
    &comboc;
    **label variables;
    label outcome=&legend_lbl clustnum=&xaxis_lbl outcome_val1=&yaxis_lbl
    &clustvar=" ";
    **apply formats;
    format clustnum boxgr. outcome outcome.;
    run;
```

This short piece of code handles the user's input for the vertref argument. This argument allows the user to specify the placement of a horizontal reference line on the boxplot. If a value is supplied, the code is dynamically written to set a macro argument equal to the supplied value, otherwise it is set equal to missing.

```
**this section dynamically writes a macro variable for creating a horizontal
reference line;
%if %length(&vertref) > 0 %then %do;
    %let verttext=vref=&vertref. lvref=2;
    %end;
%if %length(&vertref) = 0 %then %do;
    %let verttext= ;
    %end;
```

This section of code sorts the data file in order of the cluster number and cluster variable in preparation for the PROC BOXPLOT command. AXIS statements are used to remove tick marks and establish the scale on the vertical axis. Five SYMBOL statements are used to establish different shapes and the supplied colors for each of the individual boxes within a cluster. Finally, the PROC BOXPLOT command is called, plotting the outcome variable by the cluster combination variable we created with the skip-coding. We also tell SAS® to color the x-axis blocks white and to apply the colors supplied by the user to each individual box.

```
**sort in ascending order of num variable;
proc sort data=sbsv;
    by clustnum &clustvar outcome;

**sets all font to tnr and height close to 3.5 below;
goptions ftext='times new roman' htext=2.0;

**sets no tick marks;
axis1 major=none value=none;

**this macro creates symbol statements;
%macro symcall (arg,arg2);
```

```
symbol&arg v=&arg2 c=&&color&arg r=1 h=2 w=4;
%mend symcall;

**this code creates the macro calls for the symcall macro equal to n-times=the
number of outcomes;
%do i=1 %to &n;
    %unquote(&&symstat&i);
%end;

**side-by-side boxplot;
proc boxplot data=sbsv;
    plot outcome_val1*clustnum (&clustvar)=outcome
    /continuous blockpos=3 cblockvar=block notickrep cboxes=(color) cboxfill=white
    symbolorder=data
          &verttext
          font='times new roman' height=3.5
          haxis=axis1;
          format outcome outcome.;
    run;
%mend sbsbox;
(path = C:\Stuff\USC\SESUG\2010\, data_in = sesug.racedata,outcomes =math
reading science, clustvar = race, xaxis_lbl   = "Ethnicity",yaxis_lbl =
"Mean Standardized Score",legend_lbl = "Test Subject", box_lbl =
"Math"!"Reading"!"Science", box_clr =   red blue green orange purple,
vertref =, flipflop = 'N');
```

Figure 3, below, is the side-by-side boxplot from the illustrated example above, when the flipflop argument was set to the default, 'N'. This side-by-side boxplot allows a researcher to examine mean standardized test scores for three subjects – math, reading, and science – by student race/ethnicity. Conversely, Figure 4, also below, is the side-by-side boxplot when the SBSBOXPLOT macro was run a second time and the flipflop argument = 'Y'. Through this process, now, the side-by-side boxplot shows mean standardized test scores for black, Hispanic, and white students, by test subject. That is, the variables on the x-axis and the face of the graph in Figure 3 flip-flopped in the creation of Figure 4.

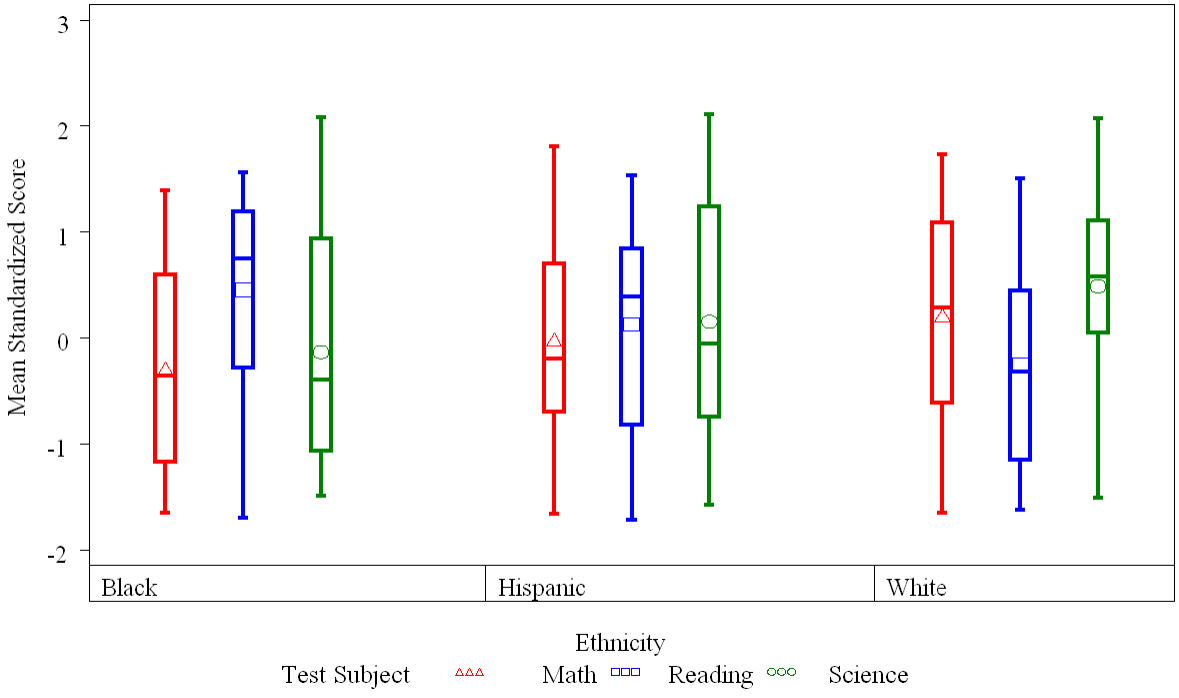

Figure 3. Side-by-Side Boxplot of subject-specific z-scores plotted within student ethnicity.

Suppose the user notices a trend in the data that may be more easily understood by swapping the outcome and grouping variables. SBSBOXPLOT has code in place to when flipflop is set equal to “Y” that will flip the outcome and grouping variables while generating the same type of plot with the user-supplied colors.

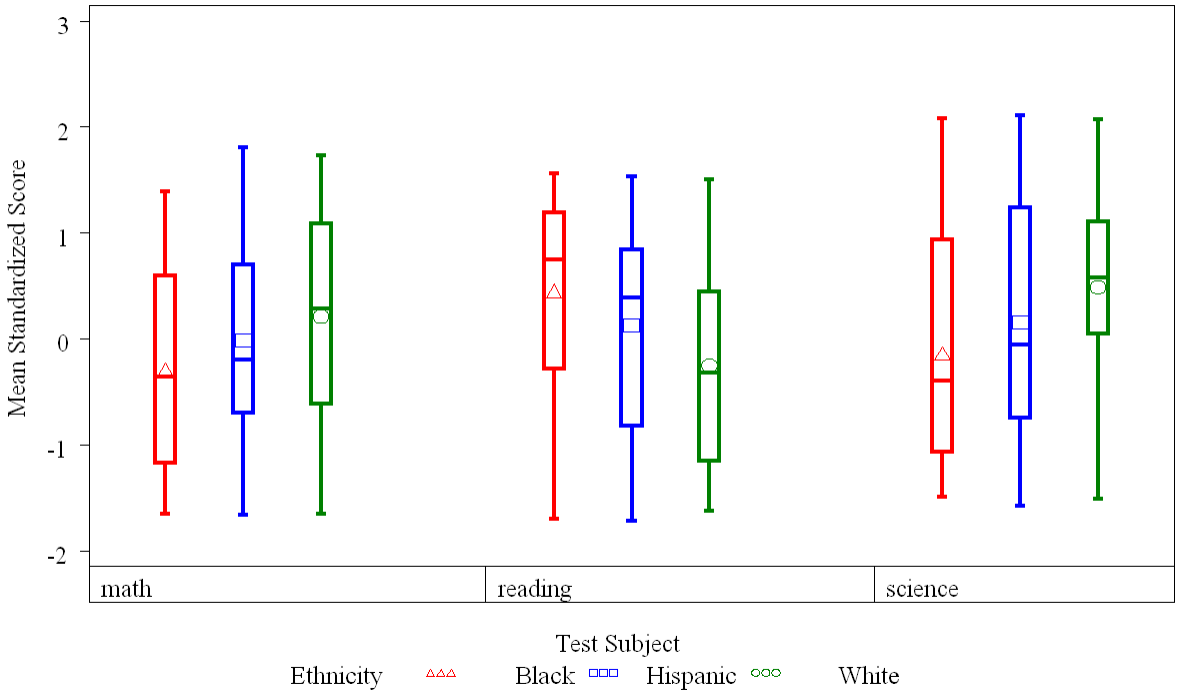

Figure 4. Side-by-Side Boxplot of ethnicity-specific z-scores plotted within subject.

CONCLUSION

Basic exploration of data at the outset of a research project is necessary for investigators to become intimately familiar with their data. Statistical graphing provides a quick and efficient method for learning about one’s data before more in-depth analyses are conducted. In addition, graphical summaries can be a clean and concise way of summarizing large quantities of outcome data that paint an interesting picture for consumers of research. The underlying justification for the development of the SBSBOXPLOT macro is that sometimes the most interesting findings occur when examining outcomes disaggregated or nested within subgroups of other variables.

A single PROC BOXPLOT command can generate an easy-to-understand visual summary of a continuous data distribution or when used with the BY STATEMENT, PROC BOXPLOT generates a series of summary boxplots, plotted on separate graphs. However, through data restructuring and SAS® syntax slight-of-hand, PROC BOXPLOT can be used to generate side-by-side boxplots. However, beginner or intermediate level SAS® users who have a need to generate such plots may not possess the technical capabilities or syntax knowledge to create their own side-by-side plots easily. The SBSBOXPLOT macro is our attempt to enable the creation of these types of plots by SAS®users at least somewhat familiar with the SAS® MACRO facility.

REFERENCES

Tukey, J. W. (1977) Exploratory Data Analysis. Addison-Wesley, Reading, MA.

CONTACT INFORMATION

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

Jason Schoeneberger
University of South Carolina
14726 Provence Lane
Charlotte, NC 28277
Phone: 704-307-9395
E-mail: schoeneb@email.sc.edu

```
%macro sbsbox
(path,data_in,outcomes,clustvar,xaxis_lbl,yaxis_lbl,
legend_lbl,box_lbl,box_clr,vertref,flipflop);
/*
Please appropriately cite the macro when used:
Schoeneberger, J. A.,Morgan, G.B., Bell, B.A., (2010).

SBSBOXPLOT: A SAS®  Macro for Generating Side-by-Side

Boxplots. Proceedings of the annual Southeastern SAS
Users Group Conference.

As you use the macro, please feel free to send feedback,
comments, and/or suggestions to,
Jason A. Schoeneberger, at schoeneb@mailbox.sc.edu or
Bethany A. Bell at bellb@mailbox.sc.edu.
*/
* +-----------------------------------------------------
--------------------------------+
Important information about the data structures
compatible with macro SBSBOXPLOT:
* +-----------------------------------------------------
--------------------------------+

**assumes data_in file is oneline per unit-of-analysis
record;
**will handle up to 10 outcomes within clusters as
unique colors specified;

*path=folder location where data file for analysis
resides;
*data_in=the data file for analysis;
*outcomes=list of dependent variables of interest,
separated by spaces;
*clustvar=the cluster variable within which boxplots are
organized. This variable will appear along the x-axis;
*xaxis_lbl=x-axis label appearing in boxplot graph;
*yaxis_lbl=y-axis label appearing in boxplot graph;
*legend_lbl=boxplot legend label appearing in graph -
i.e. what each different colored boxplot represents;
*box_lbl=boxplot labels appearing at bottom of boxplot
graph entered in corresponding order with outcomes,
separated by spaces " " and enclosed in quotes (i.e.
"Math" "Science");
*box_clr=boxplot/symbol colors enter in corresponding
order with outcomes (and as many colors as outcomes),
separated by spaces;
*vertref=location of a horizontal, dotted reference line
positioned on the vertical axis - if none needed, leave
blank;
*flipflop=the default setting 'N' generates the initial
side-by-side boxplots as intended by the user given the
parameters supplied to the arguments above. When set to
'Y', the variables submitted for clustvar and the
box_lbl are flip-flopped to reverse the plotting to
explore potentially interesting plots. When first
opening
and using SBSBOXPLOT, flipflop MUST='N';

* +-----------------------------------------------------
--------------------------------+
Output and log information printed to folder where data
for analysis is located:
* +-----------------------------------------------------
--------------------------------+
**this prints all output and log information to these
locations if the user needs to examine;
proc printto print="%bquote(&path)SBSBOXPLOT_Output.lst"
log="%bquote(&path)SBSBOXPLOT_Log.log" new;
run;

* +-----------------------------------------------------
--------------------------------+
Here code put in place for when the flipflop procedure
is enacted:
* +-----------------------------------------------------
--------------------------------+
**the following code creates a data file ready to be
tranposed when the flip-flop mechanism is turned on;
%if "&flipflop"="Y" %then %do;

      **select only variables needed to process in
      order listed in macro argument 'outcomes';
      data sbs;
            retain id &outcomes &clustvar;
            set "%bquote(&path)%bquote(&data_in)";
            **only retain necessary variables;
            keep id &outcomes &clustvar;
            id+1;
```

```
            run;

      **transpose data vertically to stack outcomes of
      interest;
      proc sort data=sbs;
            by id &clustvar;
      proc transpose data=sbs out=sbsv (drop=_label_)
      name=outcome_typeff prefix=outcome_val;
            by id &clustvar;
            run;

* +-----------------------------------------------------
--------------------------------+
Now we have to examine the new outcome variable to
determine type and handle appropriately:
* +-----------------------------------------------------
--------------------------------+
**this macro will determine the type of variable for
clustvar, and write sql code to handle accordingly;
      %global clusttype;
      %macro gettype(table);
      %let dsid=%sysfunc(open(sbsv,i));
            %do i=1 %to %sysfunc(attrn(&dsid,nvars));
      %if %upcase(&clustvar) =
      %upcase(%sysfunc(varname(&dsid,&i))) %then %let
      clusttype=%sysfunc(vartype(&dsid,&i));
            %end;
      %let rc=%sysfunc(close(&dsid));
      %mend gettype;
      %gettype(table=cat_tabff);

**these sql steps will take the values from cat_tabff
and creates labels for flipflop;
**if clustvar is a string;
      %if "&clusttype"="C" %then %do;

**here we set outcome_type equal to the cluster
variables to create the flipflop;
      data sbsv;
            set sbsv;
            outcome_type=&clustvar;
            outcome_typeff=propcase(outcome_typeff);
                  run;

      proc sql;
```

```
select unique(cat('"',compress(&clustvar),'"')),
compress(&clustvar)
      into : box_lblff separated by " ",
           : outcomeff separated by " "
      from cat_tabff;
      quit;
      %end;

      **if clustvar is a numeric;
      %if "&clusttype"="N" %then %do;

**here we set outcome_type equal to the cluster
variables to create the flipflop;
      data sbsv;
            set sbsv;
            outcome_type=compress(put(&clustvar,16.1));
            outcome_typeff=propcase(outcome_typeff);
            run;

      proc sql;
      select
      unique(cat('"',compress(put(&clustvar,16.1)),'"')
      ), &clustvar
      into : box_lblff separated by " ",
           : outcomeff separated by " "
      from cat_tabff;
      quit;
      %end;

**we then re-establish macro values to account for flip-
flop, including box_lbl to be labeled with clustvar
values;
      %let clustvar=outcome_typeff;
      %let legend_lbl2=&legend_lbl;
      %let legend_lbl=&xaxis_lbl;
      %let xaxis_lbl=&legend_lbl2;
      %let box_lbl=&box_lblff;

**this section scans the macro arguments and creates
macro calls;
**this creates order of outcomes, symbol statements, box
colors and box labels;
      %local i outcomes symbols box_color box_label n;
      %let i = 1;
      %let outcomes=&outcomeff;
```

```
	%let symbols=triangle square circle diamond plus
	hash X % =;
	%let outone = %scan(&outcomes, &i," ");
	%let sym = %scan(&symbols, &i);
	%let box_color = %scan(&box_clr, &i);
	%let box_label = %scan(&box_lbl, &i,' ');
	%do %while (&outone^=);
		%let
		outsort&i=%nrstr(%sortcall)%str((&outone,&i
		););
		%let
		symstat&i=%nrstr(%symcall)%str((&i,&sym););
		%let color&i=&box_color;
		%let label&i=%str(&i=&box_label);
		%let i = %eval(&i + 1);
		%let outone = %scan(&outcomes, &i);
		%let sym = %scan(&symbols, &i);
		%let box_color = %scan(&box_clr, &i);
		%let box_label = %scan(&box_lbl, &i,' ');
		%let n = %eval(&i - 1);
	%end;
%end;

* +-----------------------------------------------------
----------------------------------+
Here code put in place for when flipflop=N (default):
* +-----------------------------------------------------
----------------------------------+;

%if "&flipflop"="N" %then %do;

**this section scans the macro arguments and creates
macro calls;
**creates order of outcomes, symbol statements, box
colors and box labels;
	%local i outcomes symbols box_color box_label n;
	%let i=1;
	%let symbols=triangle square circle diamond plus
	hash X % =;
	%let outone=%scan(&outcomes, &i);
	%let sym=%scan(&symbols, &i);
	%let box_color=%scan(&box_clr, &i);
	%let box_label=%scan(&box_lbl, &i,' ');
	%do %while (&outone^=);
		%let
	outsort&i=%nrstr(%sortcall)%str((&outone,&i););
		%let
	symstat&i=%nrstr(%symcall)%str((&i,&sym););
		%let color&i=&box_color;
		%let label&i=%str(&i=&box_label);
		 %let i=%eval(&i + 1);
		 %let outone=%scan(&outcomes, &i);
		%let sym=%scan(&symbols, &i);
		%let box_color=%scan(&box_clr, &i);
		%let box_label=%scan(&box_lbl, &i,' ');
		%let n=%eval(&i - 1);
	%end;

**select only variables needed to process in order
listed in macro argument 'outcomes';
	data sbs;
		retain id &outcomes &clustvar;
		set "%bquote(&path)%bquote(&data_in)";
		*&data_in;
		**only retain necessary variables;
		keep id &outcomes &clustvar;
		id+1;
		run;

**transpose data vertically to stack outcomes of
interest;
	proc sort data=sbs;
		by id &clustvar;
	proc transpose data=sbs out=sbsv (drop=_label_)
	name=outcome_type prefix=outcome_val;
		by id &clustvar;
		run;

**establish arguments to ensure accurate values used
when flipflop='N';
	%let clustvar=&clustvar;
	%let legend_lbl=&legend_lbl;
	%let xaxis_lbl=&xaxis_lbl;
	%let box_lbl=&box_lbl;
%end;

**begin creating code to set up outcome codes
dynamically;
```

```
**this sql will create a table of unique outcome values
and associated counts;
proc sql;
       create table out_tab as
       select unique(outcome_type)
       from sbsv
       order by outcome_type;
       quit;

**this macro will create an outcome_sort variable to
maintain the order of outcomes as provided by user;
%macro sortcall (arg,arg2);

**calculate outcome_sort;
data out_tab;
       set out_tab;
       if (outcome_type="&arg") then outcome_sort=&arg2;
       run;

**end macro;
%mend sortcall;

**this code generates the macro statements from the
outsort macro equal to n-times=the number of outcomes;
%do i=1 %to &n;
       %unquote(&&outsort&i);
%end;

**this data step will create outcomes and write code for
outcomes and boxcolors;
data out_tab;
       set out_tab;
       outcode=cat("if
       (outcome_type='",trim(outcome_type),"') then
       outcome=",outcome_sort,";");
       boxcode=cat("if (outcome=",outcome_sort,") then
       color=",%quote(cat('"&&color',outcome_sort,'"')),
       ";");
       run;

**this sql will create macro variables holding the code
created in the previous data step for outcome and
colors;
proc sql;
       select unique(outcode), (boxcode)
       into : outc separated by '0D0A'X,
              : outb separated by '0D0A'X
       from out_tab
       order by outcode;
       quit;

**create necessary variables for side-by-side plots;
data sbsv;
       set sbsv;
       **set length of color variable;
       length color $25;
       **creates block variable for box fill on h-axis
       (see cblockvar);
       block="white";
       **create outcome variable - put macro variable
       &outc here;
       &outc;
       **set colors - put macro variable &outb here;
       &outb;
       run;

**create code to set up categories dynamically, with
skip patterns for clustering;
**this sql will create a table of unique combination
variables, number of unique outcomes;
**count of categories needed (including skips) and a
count of combinations;
proc sql;
       create table cat_tab as
       select &clustvar, outcome,

       (count(*)+(count(*)/(count(distinct(outcome)))-
       1)) as cat_count, monotonic() as combo_count
       from (select unique(&clustvar|outcome),
       &clustvar, outcome from sbsv)
       order by &clustvar, outcome;
       quit;

**this sql creates a small table of unique values for
use in flipflop=Y mode;
**this code should only be run when flipflop="N";
**the value are placed into a macro argument called
box_lblff that will be used as box_lbl in flipflop;
%if "&flipflop"="N" %then %do;
       proc sql;
```

```
            create table cat_tabff as
            select unique(&clustvar)
            from sbsv;
            quit;
%end;

* +-----------------------------------------------------
--------------------------------+
the following gettype macro will handle character or
numeric clustvar variables:
* +-----------------------------------------------------
--------------------------------+
**this macro retrieves the variable type of the clustvar
for dynamically writing code to create categories;
**here, we set clusttype=C for character and N for
numeric;
%macro gettype(table);
%global clusttype;
%let dsid=%sysfunc(open(sbsv,i));
       %do i=1 %to %sysfunc(attrn(&dsid,nvars));
            %if %upcase(&clustvar) =
%upcase(%sysfunc(varname(&dsid,&i))) %then %let
clusttype=%sysfunc(vartype(&dsid,&i));
      %end;
%let rc=%sysfunc(close(&dsid));

**this data step will write the code necessary to create
the groups;
**ultimately, the obs variable is used to write the code
to create the appropriate skip pattern;
data cat_code_tab;
      set cat_tab;
      by &clustvar;
      if first.&clustvar then clustcount+1;
      obs_lag=lag(combo_count);
      obs=sum(clustcount,obs_lag);
      %if "&clusttype"="C" %then %do;
            combo=cat("if
      (&clustvar='",trim(&clustvar),"') and
      (outcome=",outcome,") then clustnum=",obs,";");
      %end;
      %if "&clusttype"="N" %then %do;
            combo=cat("if
      (&clustvar=",trim(&clustvar),") and
      (outcome=",outcome,") then clustnum=",obs,";");
      %end;
      run;
%mend gettype;
%gettype(table=sbsv);

**this sql will create a macro variable holding the code
created in the previous data step;
proc sql;
      select unique(combo)
      into : comboc separated by "0D0A"X
      from cat_code_tab
      order by combo;
      quit;

**establish formats - boxgr is to make a format for the
new variable that has blanks for the space between
adjacent blocks;
**this section sets labels equal to each separate
supplied label along with the number in range;
proc format;
      value boxgr
      1-50=' ';
      value outcome (notsorted)
      %do i=1 %to &n;
            %unquote(&&label&i)
      %end;;

**use data step to call dynamically written code;
data sbsv;
      set sbsv;
      **put macro variable here;
      &comboc;
      **label variables;
      label outcome=&legend_lbl clustnum=&xaxis_lbl
      outcome_val1=&yaxis_lbl &clustvar=" ";
      **apply formats;
      format clustnum boxgr. outcome outcome.;
      run;

* +-----------------------------------------------------
--------------------------------+
Now we get ready to actually generate the boxplots:
* +-----------------------------------------------------
--------------------------------+
```

```
**this section dynamically writes a macro variable for
creating a horizontal reference line;
%if %length(&vertref) > 0 %then %do;
      %let verttext=vref=&vertref. lvref=2;
      %end;
%if %length(&vertref) = 0 %then %do;
      %let verttext= ;
      %end;

**sort in ascending order of num variable;
proc sort data=sbsv;
      by clustnum &clustvar outcome;

**sets all font to tnr and height close to 3.5 below;
goptions ftext='times new roman' htext=2.0;

**sets no tick marks;
axis1 major=none value=none;

* +-----------------------------------------------------
-----------------------------------------+
The following symcall macro creates symbol statements
applying symbols and colors looking like this:
symbol1 v=triangle c="&&color1" r=1 h=2 w=4;
* +-----------------------------------------------------
-----------------------------------------+
**this macro creates symbol statements;
%macro symcall (arg,arg2);

symbol&arg v=&arg2 c=&&color&arg r=1 h=2 w=4;
%mend symcall;

**this code creates the macro calls for the symcall
macro equal to n-times=the number of outcomes;
%do i=1 %to &n;
      %unquote(&&symstat&i);
%end;

**side-by-side boxplot;
proc boxplot data=sbsv;
      plot outcome_val1*clustnum (&clustvar)=outcome
            /continuous blockpos=3 cblockvar=block
      notickrep cboxes=(color) cboxfill=white
      symbolorder=data
            &verttext
            font='times new roman' height=3.5
            haxis=axis1;
            format outcome outcome.;
      run;
%mend sbsbox;
* +-----------------------------------------------------
---------------------------------+
    End of SBSBOXPLOT processing code;
* +-----------------------------------------------------
---------------------------------+;
```