Paper 203 - 2009

SAS® Macros Tool Kit for Graphical Representation of Complex Data

Ashok Chaurasia, Department of Statistics, University of Connecticut, Storrs, CT

ABSTRACT

There are many procedures and settings in SAS® that support the construction of elegant and informative graphs for data analysis. When dealing with complex data, it can be cumbersome to produce graphs that are both informative and consistent in their design. This task can be made easier with some simple macros.

This paper introduces a SAS® macro toolkit that helps construct design-consistent graphs from one complex data set to another. These macros aid the user in areas such as data reconfiguration to make the information easily accessible to SAS® procedures, control over various elements of SAS® graphs, and control over the options of SAS® procedure statements.

INTRODUCTION

My adventure with SAS® macros began at the University of Texas at San Antonio while working as a research assistant for the Statistical Consulting Center. My first research assignment was to reproduce technical graphs from an engineering standards manual, using SAS®. From this experience I realized the necessity, effectiveness, and importance of consistent/flexible SAS code when constructing graphs for complex datasets of similar caliber. The macro facility and the procedures in SAS® serve as a powerful tool for data configuration and informative graphical representation of complex datasets. This paper introduces some simple macros that I created to reduce the amount of coding (which can reach daunting lengths when dealing with large and complex datasets) and to be consistent in the design of the resulting graphs. The macros discussed in this paper are as follows: **%listlength** (returns the number of unquoted elements in a list), **%rep** (repeats numbers, strings, with or without quotes; similar to the REPEAT function in SAS®), **%ifoptions** (provides flexibility in inclusion/exclusion of options in SAS® procedures), **%Axis** (creates a customized axis statement for SAS® graphs), **%createAxes** (generates as many axis statements as needed; it makes use of **%Axis**), **%n4symbols** (returns the value of required number of symbol statements), **%symb** (generates as many symbol statements as necessary depending on the complexity of the data; it makes use of **%n4symbols**), **%TnF** (generates titles/subtitles and footnotes), **%combineAll** (this macro scans through all the sheets of an Excel® document and then combines the data from each sheet to form one complete dataset; it makes use of **%makedata**), and **%makeV2F** (this macro scans through all variables [of interest] in an Excel® sheet to create a new dataset where the scanned variables become factors levels of a new user specified variable while maintaining data integrity). After reading this paper, SAS® users should be able to readily apply these macros to facilitate their analysis.

The macros presented in this paper can be downloaded from
http://www.stat.uconn.edu/~achaurasia/Published/Ashok_Chaurasia_SAS_Conf_2009_macros.sas

%listlength

The first macro in the tool kit is `%listlength`; this macro returns the number of (unquoted) elements (separated by the space character(s)) in a list. The elements can be characters, strings or numbers. Note: this macro collapses multiple space characters between elements to a single space character.

```
%listlength(<list>);
```

Parameter	Default	Description
list	*none*	specifies the list

```
Example:       %listlength(Here is an example of $ # @ ! & % 5 6);
  Result:      13
```

This macro is very useful when we want to scan through a list of elements and for each element execute PROCs or DATA steps. For example, consider an excel® document named *grades.xls* (located in F:\) which has three sheets – *midterm1*, *midterm2*, and *final*. To read all the sheets and create SAS datasets corresponding to each sheet, we write a macro which uses the `%listlength` macro as follows:

```
%MACRO makedatasets(file=, sheets=, getnames=yes);
   %DO i=1 %TO %listlength(&sheets);
      %LET sheet=%SYSFUNC(SCANQ(&sheets, &i));
      PROC IMPORT DATAFILE=&file OUT= &sheet DBMS=EXCEL REPLACE;
         SHEET=&sheet;
         GETNAMES= &getnames;
      RUN;
   %END;
%MEND makedatasets;

%makedatasets(file="F:\grades.xls", sheets=midterm1  midterm2   final, getnames=yes);
```

The `%makedatasets` macro successfully scans through the list of sheet names and creates three SAS datasets. This is illustrated by the statements from the LOG file which are as follows:

```
NOTE: WORK.MIDTERM1 data set was successfully created.
NOTE: PROCEDURE IMPORT used (Total process time):
      real time           2.64 seconds
      cpu time            0.23 seconds

NOTE: WORK.MIDTERM2 data set was successfully created.
NOTE: PROCEDURE IMPORT used (Total process time):
      real time           0.07 seconds
      cpu time            0.04 seconds

NOTE: WORK.FINAL data set was successfully created.
NOTE: PROCEDURE IMPORT used (Total process time):
      real time           0.07 seconds
      cpu time            0.04 seconds
```

Note that the `%listlength` macro has improved the quality of the `%makedatasets` macro by automating the ending value for the %DO loop and reducing the number of arguments in the macro definition. Without the use of the `%listlength` macro, one would have to add another argument specifying the ending value for the %DO loop, which would have to be constantly changed to the number of sheets existing in an excel® file.

%rep

This macro can replicate any string (with or without quotes) or numeric values any number of times as specified by the user. This macro returns a string in which the first argument (given by the value of `what`) appears `n+1` times. The macro and its arguments are described below:

```
%rep(what=, n=);
```

Parameter	Default	Description
what	*none*[*]	specifies the string (with or without quotes) of numbers, characters, or their combination, to be replicated
n	*none*	specifies the number of times to repeat the value given by `what`; must be an integer.

Example: `%rep(what=% "% " 45 $ "$ " & " &", n=2);`

Result: `% "% " 45 $ "$ " & " &" % "% " 45 $ "$ " & " &" % "% " 45 $ "$ " & " &"`

This macro improves upon the REPEAT function (to suit our purposes) as follows:

```
%MACRO rep(what=, n=);
   %SYSFUNC( REPEAT(%BQUOTE(&what ), &n) );
%MEND rep;
```

Lets us discuss how the `%rep` macro improves upon the REPEAT function through an example.

If the %BQUOTE function was removed from the `%rep` macro definition as follows:

```
%MACRO rep(what=, n=);
   %SYSFUNC( REPEAT(&what , &n) );
%MEND rep;
```

Then,

```
%rep(what=Circle, n=2);
```

results in

```
CircleCircleCircle
```

The REPEAT function alone ignores all space characters before and after the value of the first argument and hence returns a single string (e.g. `CircleCircleCircle`). The purpose of the %BQUOTE function in the definition is to mask the value of the first argument with a single space character.

With the inclusion of the %BQUOTE function in the macro definition,

```
%rep(what=Circle, n=2);
```

results in

`Circle Circle Circle`.

The `%rep` macro has successfully added a space character in between the repetitions to produce a legible array of `n+1` elements.

Note: **`%listlength`** and **`%rep`** when executed in open code result in an error statement in the LOG file. These macros, when used within other complex macros, serve their purpose and do not result in any errors when used correctly.

[*] *none* implies that the parameter does not have a default value; the parameter requires a (valid) value when the macro is executed.

%ifoptions

This macro has two parameters and is useful in providing control over options of different SAS® procedures. This macro is used within other macros that pass a value of YES or NO to the `%ifoptions` macro; this will be clear through an example after we discuss the macro parameters.

```
%ifoptions(option=, want=);
```

Parameter	Default	Description
option	*none*	the options of the procedure where the `%ifoption` is called; it is implicit that the options must exist for the procedure.
want	YES	the value must be either YES|Y or NO|N (not case sensitive).

Example: `%ifoptions(option=REGEQN OVERLAY, want=Y);`
Result: `REGEQN OVERLAY`

Let us suppose that a macro called `%genPlots` (given below) is used on a SAS dataset called *sampledata* (with response variable `yvar` and predictors `x1` and `x2`).

```
%MACRO genPlots(data=, xvar=, yvar=);
   PROC GPLOT DATA=&data;
      PLOT &yvar*&xvar;
      SYMBOL VALUE=dot COLOR=blue INTERPOL=RL;
   RUN;
%MEND genplots;
%genPlots(data=sampledata, xvar=x1, yvar=yvalue);
%genPlots(data=sampledata, xvar=x2, yvar=yvalue);
```

Clearly, `%genPlots` generates plots with interpolation value as RL. If we require the inclusion of the regression equation on the graph for only one of the predictor variables, say `x2`, we can accomplish that by slightly modifying the `%genPlots` macro as follows:

```
%MACRO genPlots(data=, xvar=, yvar=, see_eqn=yes);
   PROC GPLOT DATA=&data;
      PLOT &yvar*&xvar / %ifoptions(option=REGEQN, want=&see_eqn);
      SYMBOL VALUE=dot COLOR=blue INTERPOL=RL;
   RUN;
%MEND genPlots;

%genPlots(data=sampledata, xvar=x1, yvar=yvalue, see_eqn=no);       /* (1) */
%genPlots(data=sampledata, xvar=x2, yvar=yvalue);                   /* (2) */
```

Note that the inclusion of **`%ifoptions`** has now provided some flexibility in the `sampledata` presentation such that the regression equation will be displayed only for variable x2 (see **(2)**, where the default value for `see_eqn=YES`) and not for the variable x1 (see **(1)**, where `see_eqn=NO`).

%Axis

This macro has various parameters and is useful in having control over the appearance of the axes[†] of plots produced in SAS®. This macro is used within the **`%createAxes`** macro (which is discussed next) that creates multiple axes statements. Let us first discuss the arguments of **`%Axis`**.

```
%Axis(n= , offset= , title= , justify= , height= , angle= , font= , color= , order=);
```

[†] For simplicity, I have only included certain elements of the AXIS statement, but one can certainly add more parameters to have complete control over the appearance of the axes.

Parameter	Default	Description
n	*none*	specifies the number for the axis. For example, when n=1 the AXIS1 statement is created.
offset [‡]	1in	this offsets both the x-axis and y-axis by the measure and unit specified by the user. Note: the value and its units should not be separate by a space character, i.e. use **1in** or **1cm** rather than **1 in** or **1 cm**.
title	“Title here”	specifies the title for the axis. MUST be in quotes.
justify	CENTER	specifies the justification of the axis label.
height	1.5	specifies the height of the axis label.
angle	90	specifies the angle of the axis label.
font	SIMPLEX	specifies the font for the axis label.
color	BLACK	specifies the color for the axis label.
order	D	specifies the order for the axis. D represents the default which is to exclude ORDER.

Example:
```
%Axis(n=2, offset=0.75in, title='1/Temperature(C)', angle=0,
color=blue);
```
Result:
```
AXIS2 OFFSET = (0.75in 0.75in) LABEL = (ANGLE=0 JUSTIFY=CENTER
HEIGHT=1.5 FONT=SIMPLEX COLOR= blue '1/Temperature(C)');
```

%createAxes

This macro creates multiple axis statements by using the previously discussed **%Axis** macro. This macro has various parameters, all of list type (except one), which are described next. NOTE: the arguments of list type must be specified, and must be of the same length. If the list elements are separated by multiple space characters, they will be replaced by a single space character when the macro is executed.

```
%createAxes(numOfAxes= , offsetlist= , titlelist= , anglelist= , justifylist= ,
            heightlist= , fontlist= , colorlist= , orderlist= );
```

Parameter	Default	Description
numOfAxes	2	specifies the number for the axis statements to be created. For example, when n=2, AXIS1 and AXIS2 are created with their respective elements.
offsetlist [§]	1in 1in	list type that specifies the offset value for both the x-axis and y-axis for each of the axis statements. Also refer to **%Axis**.
titlelist	“title 1” “title 2”	list type that specifies the title for each of the axes. Each list element must be in quotes.
anglelist	90	list type that specifies the angle for each of the axes labels.
justifylist	CENTER	list type that specifies the justification for each of the axes labels.
heightlist	1.5	list type that specifies the height for each of the axes labels.
fontlist	SIMPLEX	list type that specifies the font for each of the axes labels.
colorlist	BLUE BLACK	list type that specifies the color for each of the axes labels.

[‡] For the sake of simplicity, offset is set for both ends of the axis; one can create arguments to control the offset for each end separately.
[§] For the sake of simplicity, offset is set for both ends of the axis; one can create arguments to control the offset for each end separately.

orderlist	D$D	list type that specifies the order for each of the axes. The elements must be separated by $, for example, `0 to 20 by 5$10 to 100 by 10`. D represents the default which is to exclude ORDER.

Example:
```
%createAxes(numOfAxes=2, offsetlist=   1in    2in,
            titlelist="Put title 1   here"    "Put title 2 here",
            anglelist=  0  0  , justifylist=CENTER LEFT,
            heightlist=1.5 2.1,
            fontlist=arial simplex, colorlist=b  r,
            orderlist=10 to 20 by 1$50 to 100 by 5);
```
Result:
```
AXIS1 OFFSET = (1in 1in) ORDER = (10 to 20 by 1) LABEL = (ANGLE=0
JUSTIFY=CENTER
HEIGHT=1.5 FONT=arial COLOR= b "Put title 1   here");
AXIS2 OFFSET = (2in 2in) ORDER = (50 to 100 by 5) LABEL = (ANGLE=0
JUSTIFY=LEFT
HEIGHT=2.1 FONT=simplex COLOR= r "Put title 2 here");
```

%n4symbols

This macro calculates the number of symbol statements required for a data set.

```
%n4symbols(data= , symbvar= , nname= , library= , keep= , dropcounts= );
```

Parameter	**Default**	**Description**
data	*none*	specifies the SAS dataset for which the number of symbol statements required for graphical presentation.
symbvar	*none*	specifies the *third-variable* (see SAS online documentation on PLOT statement of PROC GPLOT), which establishes the number of required symbol statements.
nname	_n4symb_	Global macro variable name that stores the value of the number of required symbol statements.
library	WORK	specifies the library where the SAS datasets, created in the process, are stored.
keep	NO	this gives the user the option to either keep or delete the SAS datasets created in the process of calculating the number of required symbol statements. The values expected for this variable is either YES|Y or NO|N; any other value will result in an error in the LOG file.
dropcounts	0	this options allows the user to ignore observation where the value of the dataset variable, given by the `symbvar`, has counts (occurrences) less than or equal to `dropcounts`.

Example: Consider the following SAS dataset

```
Data material;
   INPUT Temp Strength Alloy $;
cards;
80 1574 Alloy1
90 1487 Alloy1
100 1200 Alloy1
80 1617 Alloy2
90 1550 Alloy2
100 1475 Alloy2
90 1550 Alloy3
100 1475 Alloy3
;
RUN;
```

For the above *material* SAS dataset, the code

```
%n4symbols(data=material, symbvar=Alloy, nname=n, library=WORK, keep=NO,
           dropcounts=2);
```

returns a value of *2* in the log file (for reference) and this value is stored in the macro variable *n*.

%symb

This macro creates multiple SYMBOL statements as required. This macro has various arguments, all of list type (except one), which are described next. NOTE: the arguments of list type, if specified, must each be of equal length (and equal to **n** – the number symbol statements). If the list elements are separated by multiple space characters then they will be replaced by a single space character when the macro is executed.

```
%symb(n= , font= , val= , color= , connect= , wd= , ht= , linetype= , repeat= );
```

Parameter	Default	Description
n	*none*	specifies the number for the required SYMBOL statements. For example, when n=2, SYMBOL1 and SYMBOL2 are created with their respective elements.
font	dflt	list type with ‘n’ elements that specifies the FONT values for each of the SYMBOL statements.
val	dot	list type with ‘n’ elements that specifies the VALUE values (such as circle, dot, etc) for each of the SYMBOL statements.
color	*see the code for default color values*	list type with ‘n’ elements that specifies the COLOR values for each of the SYMBOL statements.
connect	*none*	list type with ‘n’ elements that specifies the INTERPOL values for each of the SYMBOL statements.
wd	1	list type with ‘n’ elements that specifies the WIDTH of lines for each of the SYMBOL statements.
ht	1	list type with ‘n’ elements that specifies the HEIGHT (i.e size) of the symbol values (such as square, circle, etc) for each of the SYMBOL statements.
linetype	1	list type with ‘n’ elements that specifies the LINE type for each of the SYMBOL statements.
repeat	1	list type with ‘n’ elements that specifies if certain SYMBOL statements should be repeated.

Refer to SAS® Online documentation to understand the elements of the SYMBOL statement.

Example: From the previous example, we use the value of the macro variable *n* as follows:

```
%symb(n=&n, wd=3.2 3.2, ht=1.2 1.2);
```

Result:

```
SYMBOL1 FONT=, VALUE=dot COLOR=BLACK INTERPOL=none WIDTH=3.2 HEIGHT=1.2
LINE=1 REPEAT=1 ;
SYMBOL2 FONT=, VALUE=dot COLOR=BLUE INTERPOL=none WIDTH=3.2 HEIGHT=1.2
LINE=1 REPEAT=1 ;
```

%TnF

This macro creates a custom Title and Footnote for plots produced in SAS®. This macro is versatile because it can be included within any PROC GPLOT, or any macro that uses PROC GPLOT. This specific macro was customized to a dataset that was a compilation of several smaller datasets. Some of the parameters given below were created for catalog purposes.

```
%TnF(t1= , t2= , t3= , f1= , f2= , sheet= , t1j= , t1f= , t1h= , t2j= , t2f= ,
     t2h= , t3f= , t3h= , f1j= , f1f= , f1h= , f2j= , f2f= , f2h= );
```

Parameter	Default	Description
t1	“SAS file name”	specifies the name of the SAS file in the header.

t2	"Plot for dataset"	specifies the title of the plot. Ex: *For SET 1*
t3	"Title of plot"	specifies the main title of the plot. Ex: *Pressure vs Time*
f1	"Data File Name"	specifies the name of the dataset used in the constructing the plot. Ex:
f2	"Job Number"	specifies the job number for documentation purposes.
sheet	**.**	specifies the EXCEL® sheet corresponding to the sub-dataset.
t1j	LEFT	specifies the justification for **t1**.
t1f	SIMPLEX	specifies the font for **t1**.
t1h	1	specifies the height (size) of **t1**.
t2j	CENTER	specifies the justification for **t2**.
t2f	SIMPLEX	specifies the font for **t2**.
t2h	10pt	specifies the height (size) of **t2**.
t3f	SIMPLEX	specifies the font for **t3**.
t3h	12pt	specifies the height (size) of **t3**.
f1j	RIGHT	specifies the justification for **f1**.
f1f	SIMPLEX	specifies the font for **f1**.
f1h	1	specifies the height (size) of **f1**.
f2j	LEFT	specifies the justification for **f2**.
f2f	SIMPLEX	specifies the font for **f2**.
f2h	1	specifies the height (size) of **f2**.

Example:
```
%TnF( t1="material.sas", t2="Strength vs Temperature",
      f1="material.xls", sheet="sheet1", f2="Catalog 2" );
```

Result:
```
TITLE1 JUSTIFY=LEFT FONT=SIMPLEX HEIGHT=1 "material.sas";
TITLE2 JUSTIFY=CENTER FONT=SIMPLEX HEIGHT=12pt "Strength vs Temperature"
JUSTIFY=CENTER FONT=SIMPLEX HEIGHT=12pt "Title of plot";
FOOTNOTE1 JUSTIFY=RIGHT FONT=SIMPLEX HEIGHT=1 "Data file: "
"material.xls" JUSTIFY=RIGHT "Sheet(s): " "sheet1" ;
FOOTNOTE2 JUSTIFY=LEFT FONT=SIMPLEX HEIGHT=1 "Job: " "Catalog 2";
```

%makedata

This macro reads a specific SHEET within an EXCEL® file by using PROC IMPORT. The purpose of this macro is to read the entire data corresponding to the specific SHEET and also create a new variable (as specified by the user, default variable name is _**TYPE**_) that contains the SHEET name as its value in character format. This data is stored as a SASDATASET with the same name as the SHEET name; for example, if the sheet name called in PROC IMPORT is **grades** then the newly created SASDATASET name will be **GRADES**. The macro and its argument description are as follows:

```
%makedata(file= , sheet= , type_val= , length_type= );
```

Parameter	Default	Description
file	*none*	specifies the location of the EXCEL® file. Must be in quotes.
sheet	*none*	specifies the SHEET name in the EXCEL® file.
type_val	_typeval_	specifies the name of the new variable containing the SHEET name in character format.
length_type	*none*	specifies the length for `typeval` variable; this value, at the least, must be equal to the length of the sheet name.

Example: Consider a file called *material.xls* (located in *F*:\) with a sheet named *Alloy1* that contains data as follows:

Alloy1

```
Temp   Strength
 80       1574
 90       1487
100       1200
```

Result: The code `%makedata(file="F:\material.xls", sheet=Alloy1,`
```
                  type_val=Alloy_Type, length_type=6);
```
restructures the data as follows:

```
Alloy_Type  Temp  Strength
Alloy1        80      1574
Alloy1        90      1487
Alloy1       100      1200
```

%combineAll

In many situations, we have data from a study (with the same variables) distributed over multiple excel® sheets. The **%combineAll** macro solves this problem by scanning through the list of sheet names and executing on each the macro **%makedata**. Once SASDATSETS for each of the sheets are created, this macro combines all the datasets into one large dataset.

```
%combineAll(file= , sheets= , new_var= , new_var_length= , out_data= , keeptemps=,
            lib= );
```

Parameter	Default	Description
file	*none*	specifies the location of the EXCEL® file. Must be in quotes.
sheets	*none*	specifies the SHEET names as a list without quotes; the SHEET names must not contain any special character or spaces. Each element in the list must be separated by the single space character.
new_var	*none*	specifies the name of the new variable containing the SHEET name in character format.
new_var_length	*none*	specifies the length for `new_var` variable; this value, at the least, must be equal to the length of the sheet name
out_data	*none*	specifies the name of the final data set containing data from all the sheets.
keeptemps	NO	specifies a value YES|Y or NO|N to keep or delete the SASDATASET (corresponding to each datasheet) from the working library specified by the “lib” variable (described next).
lib	WORK	specifies the name of the final data set containing data from all the sheets.

Example: Consider a file called *material.xls* (located in *F*:\) with sheets named *Alloy1*, *Alloy2*, and *Alloy3* that contain data as follows:

Alloy1	
Temp	Strength
80	1574
90	1487
100	1200

Alloy2	
Temp	Strength
80	1617
90	1550
100	1475

Alloy3	
Temp	Strength
90	1550
100	1475

Result: The code
```
%combineAll(file="F:\material.xls", sheets=Alloy1 Alloy2 Alloy3,
            new_var= Alloy_Type, new_var_length=6,
            out_data=Alloy_combined);
```

The restructured data, *Alloy_combined*, is as follows:

```
Alloy_Type    Temp    Strength
Alloy1          80        1574
Alloy1          90        1487
Alloy1         100        1200
Alloy2          80        1617
Alloy2          90        1550
Alloy2         100        1475
Alloy3          90        1550
Alloy3         100        1475
```

%makeV2F

This macro scans through all variables (of interest as specified by the user) in an Excel® sheet to create a new dataset where the scanned variables become factors levels of a new user specified variable, while maintaining data integrity.

```
%makeV2F(data= , vars= , svar= , svar_length= , dropvars= , ddname= , newvar= , new= ,
         keeptemps= , lib=WORK );
```

Parameter	Default	Description
data	*none*	specifies the SAS dataset to be restructured
vars	*none*	specifies the variables that are to become factors levels of `svar`.
svar	_TYPE_	specifies the name of the new variable which will contain the name of the variables (as given by `vars`) as factor levels.
svar_length	*none*	specifies the character length for `svar` to reflect the maximum length of the variables names as given in `vars`.
dropvars	*none*	specifies the variable(s), that are of no concern, in the restructuring of user specified dataset, to be dropped
ddname	*none*	specifies a name for the intermediate datasets created during restructuring.
newvar	*none*	specifies the name of the new variable which will contain the values corresponding to each of the variables specified in `var`.
new	YES	specifies if the restructured data is to be stored in a new file or replace the original SAS dataset. For example, if `data`=*counting* and new=YES, the resulting new SAS dataset is named *new_counting*.
keeptemps	NO	specifies if the intermediate SAS datasets are to be kept or deleted for the working library.
lib	WORK	specifies the working library

Example: Consider the following *Rupture* SAS dataset.

```
Time  X1_temp  X2_temp  X3_temp
 10      81       91       101
 20      82       92       102
 30      83       93       103
```

Rupture contains the response variable *Time* (time to failure) with predictor variables X1_temp, X2_temp, and X2_temp which represent temperature values at rupture in alloy X1, X2 and X3 respectively.

The *Rupture* SAS dataset is restructured by the following command

```
%makeV2F(data=Rupture, vars=X1_temp  X2_temp  X3_temp, dropvars=,
         ddname=temporary, newvar=value, new=yes, keeptemps=no, svar=Type,
         svar_length=7, lib=WORK);
```

The new restructured SAS dataset called *new_Rupture* is shown below:

```
Type      Time   value
X1_temp    10      81
X1_temp    20      82
X1_temp    30      83
X2_temp    10      91
X2_temp    20      92
X2_temp    30      93
X3_temp    10     101
X3_temp    20     102
X3_temp    30     103
```

CONCLUSION

The macros listed in this paper are certainly invaluable when constructing elegant graphs: however do they really work when dealing with complicated datasets? The answer to this question is best illustrated with the graphs provided in the link below. These represent 9 out of 38 graphs produced from one excel® document which is similar (in design) to one of the excel® documents obtained from our client.[**]

http://www.stat.uconn.edu/~achaurasia/Published/Ashok_Chaurasia_SAS_Conf_2009_Graphs.pdf

All the supporting material presented in this paper can be downloaded from
http://www.stat.uconn.edu/~achaurasia/research.html

REFERENCES

1. SAS Institute. 2008. The SAS system: SAS OnlineDoc®, Version 8, HTML format. Cary, NC.
2. Introduction to SAS. 2007. UCLA: Academic Technology Services, Statistical Consulting Group. <http://www.ats.ucla.edu/stat/sas/notes2/>. (Jan. 20, 2008).

ACKNOWLEDGEMENTS

I would like to thank Dr. Kannan, Dr. Cano, and Mr. Anderson from the **University of Texas At San Antonio** (UTSA) for giving me the opportunity to work as a research assistant for UTSA's **Statistical Consulting Center**. The origins of this paper began at UTSA and I am very grateful to the faculty for their support and advice. I would also like to thank Dr. Ravishanker at my current university, **University of Connecticut**, for her advice during the preparation of this paper. Finally, I would like to thank our client for permitting us to use the design of their datasets.

CONTACT INFORMATION

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

Ashok Chaurasia
Department of Statistics
University of Connecticut
215 Glenbrook Rd. U-4120
Storrs, CT 06269
Email: ashok.chaurasia@uconn.edu
Website: http://www.stat.uconn.edu/~achaurasia/

[**] For proprietary reasons, the actual data from the client was not used in the construction of the graphs presented in this paper.