Datastep

The datastep is an essential tool for organizing data into a correct form to use with a procedure. Also, mathematical and statistical calculations can be made within the datastep as well. The purpose of this section is to examine the datastep in further detail than in previous sections. All programs and data sets used for these notes are available from my course website. New files that we have not used before are cereal_datastep.sas, general_func.sas, merge.sas, placekick_datastep.sas, placekick_datastep.csv, repeated_measure.sas, and cpt.sd2.

Permanent SAS data sets

SAS data sets are stored in libraries with "Work" being the library that we have used so far. Each data set is in a separate file stored at a temporary location on your computer (my location: C:\Users\Chris\AppData\Local\Temp\SAS Temporary Files), and these files are deleted upon closing SAS. There are occasions when you may want to keep these files (e.g., suppose a particular data set takes a long process to put into a usable form), so it can be helpful to create a *permanent* SAS data set which will not be deleted.

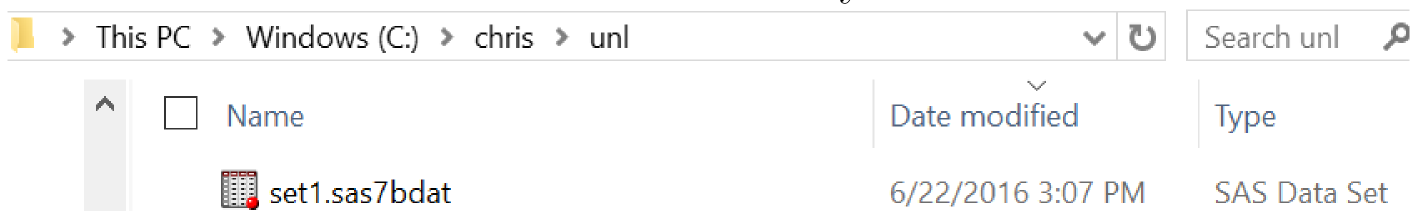
Suppose we would like to create a permanent SAS data set stored in a library named "chris". Below is my SAS code and output with `set1` containing the cereal data.

```
libname chris "C:\chris\unl";

data chris.set1;
  set set1;
run;
```

The `libname` statement creates a new library at the location specified on my hard drive. After running this first line of code, you will notice the chris library is now one of the active libraries, like work, listed in the Explorer window. Next, I simply use a datastep to create a new data set. By specifying the library first in the name of `chris.set1`, SAS creates the `set1` data set in the chris library rather than the default work library.

The previous code does not move `set1` out of the work library; rather, there are two `set1` data sets now. One could refer to the work library data set as `work.set1` if desired to differentiate between them. Also, it is interesting to note that the Log window has always referred to data sets from our previous programs in this longer manner.

Below is the actual data set file on my hard drive:



The file extension refers to SAS version 7B, which was the last time SAS updated their data file types. To use this file in a new program, one can create a new library with a `libname` statement that points to the file location where the file resides. The same chris library name does not need to be used. Then the `LibraryName.set1` data set (replace LibraryName with whatever you name the library) can be used as one would normally use any data set in the work library. The exception is that `LibraryName.` must be given prior to `set1`. The "set1.sas7bdat" file can also simply be clicked on to view it in SAS.

Conditional execution

We often encounter situations where one would like a particular set of code to run "if" a particular condition is true. Within a

datastep, this can be done using if-then-else statements.

After I originally collected my placekick data for Bilder and Loughin (*Chance*, 1998), the data was not in a form that was suitable for analysis. Some of the changes to the "raw" data that were needed included

- Create indicator variables to represent variables coded in a character format; e.g., a `FIELD` variable coded as "G" for grass and "T" for artificial turf needed to be coded as indicator variables (1 = "G", 0 = "T")
- Create an indicator variable to denote whether the placekicker was kicking in their home stadium by comparing the location variable `LOC` to the kicking team variable `TEAM`
- Convert the wind variable to an indicator format to represent windy conditions (`TYPE` = O for outdoor stadium with `SPEED` $>$15 mph) vs. non-windy conditions (`TYPE` = "D" for dome stadium or `TYPE` = "O" for outdoor stadium with `SPEED` $\leq$15 mph)

Below is how one can make these three changes:

```
proc import out=placekick1
   datafile="C:\data\placekick_datastep.csv" DBMS=CSV replace;
  getnames=yes;
  datarow=2;
run;

data placekick2;
  set placekick1;
  if field = "G" then field1 = 1;
    else field1 = 0;
  if loc = team then home = 1;
    else home = 0;
  if type = "O" and speed > 15 then wind = 1;
    else wind = 0;

  *If concerned about missing values;
```

```
  if field = "G" then field2 = 1;
    else if field = "T" then field2 = 0;
      else field2 = .;
run;

title2 "Portion of placekick data";
proc print data=placekick2(obs = 10);
  var field field1 field2 loc team home type speed wind;
run;

*Example check;
title2 "Check field variable";
proc freq data=placekick2;
  tables field*field1 / norow nocol nocum nopercent;
run;
```

Chris Bilder, STAT 850
Portion of placekick data

Obs	FIELD	field1	field2	LOC	TEAM	home	TYPE	SPEED	wind
1	G	1	1	NE	NE	1	O	11	0
2	G	1	1	NE	NE	1	O	11	0
3	G	1	1	NE	CL	0	O	11	0
4	G	1	1	NE	NE	1	O	11	0
5	T	0	0	IN	IN	1	D	.	0
6	T	0	0	IN	CN	0	D	.	0
7	T	0	0	IN	CN	0	D	.	0
8	T	0	0	IN	IN	1	D	.	0
9	T	0	0	IN	CN	0	D	.	0
10	T	0	0	IN	CN	0	D	.	0

Chris Bilder, STAT 850
Check field variable

The FREQ Procedure

Frequency	Table of FIELD by field1		
	field1		
FIELD	0	1	Total
G	0	682	682
T	743	0	743
Total	743	682	1425

Comments:

- Notice the use of `and` in one of the if-then statements. An `or` can be used for other purposes too. Also, parentheses can be used to make sure particular conditions are evaluated first. For example, suppose there are 3 variables named `x1`, `x2`, and `x3` in a data set named `set1`. Then a potential if-then-else statement could be

```
data set2;
  set set1;
```

```
  if x3 > 1 and (x1 > 1 or x2 >1) then x4 = 1;
    else x4 = 0;
run;
```

- SAS denotes missing values for numerical values in any data set by a period. There are many missing values in this data set. If there is concern about how the missing values are treated when performing an if-then-else statement, a second "nested" if-then-else statement can be given. Of course, nested if-then-else statements can be used for other purposes too.
- Various checks of the data should always be made to make sure the code worked as intended.
- The change variable that we saw in a previous section of the notes was a binary variable denoting lead-change (1) versus non-lead-change (0) placekicks. Successful lead-change placekicks are those that change which team is winning the game. How could this variable be created here for this original version of the data set? The variables that will help include `sc_team` (score of kicking team), `sc_opp` (score of non-kicking team), and `PAT` (="Y" for point after touchdowns which are worth 1 point if successful and ="N" for field goals which are worth three points if successful).

Additional notes:

- There may be instances when a set of commands need to be completed when a condition is satisfied. These commands can be included within a do-end statement. For example, suppose if `field = "G"`, a set of commands are needed. The syntax within a datastep would be similar to

```
if field = "G" then do;
  field1 = 1;
  SecondCommand = "Y"; *Just an example;
  end;
else field1 = 0;
```

- In some situations, many nested if-then-else statements may need to be used to take care of a large number of conditions. Alternatively, a select-when statement can be used to simplify the code. Please see the example in the placekicking program.

Re-organizing data

Horizontal concatenation

Data sets with the same variables can be horizontally concatenated in a datastep by putting their names in the `set` statement. The gpa_graphics.sas program illustrates how this can be done to get a data set in the correct form for a plot. Below is the code and output illustrating the process.

```
title2 "The HS and College GPA data set";
proc print data=set1(obs=5);
run;

*There is not a good way to do side-by-side box plots with HS
  and College in the current form of the data set. Instead, the
  data needs to be put in a form with
*  school   GPA
*  HS       3.04
*  HS       2.35
*  ...
*  HS       2.88
*  College  3.1
*  ...
*  College  2.6;

*Below is how I change the data set;
data HSset(drop=College) Collegeset(drop=HS);
  set set1;
run;

data HSset;
  set HSset;
  rename HS = GPA;
```

```
  school = "HS";
run;

data Collegeset;
  set Collegeset;
  rename College = GPA;
  school = "College";
run;

data set2;
  set Collegeset HSset;
run;

title2 "New form of data set used for box plots";
proc print data=set2;
run;
```

Chris Bilder, STAT 850
The HS and College GPA data set

Obs	HS	College
1	3.04	3.10
2	2.35	2.30
3	2.70	3.00
4	2.55	2.45
5	2.83	2.50

Chris Bilder, STAT 850
New form of data set used for box plots

Obs	GPA	school
1	3.10	College
2	2.30	College
3	3.00	College
4	2.45	College
5	2.50	College

<Output excluded>

35	2.22	HS
36	1.98	HS
37	2.88	HS
38	4.00	HS
39	2.28	HS
40	2.88	HS

Notice the use of the `rename` statement in the datasteps.

Create multiple data sets from one datastep

An `output` statement in a datastep can be used to create new data sets. One situation where this can be useful is with splitting a data set into multiple parts depending on a variable value. For example, suppose we would like separate cereal data sets based on the shelf. The cereal_graphics.sas program illustrated how to do this when showing how to get the data into a different form for plotting purposes.

```
title2 "Cereal data adjusted for serving size";
proc print data=set1(obs = 5);
run;

data shelf1 shelf2 shelf3 shelf4;
  set set1;
  if shelf = 1 then output shelf1;
  if shelf = 2 then output shelf2;
  if shelf = 3 then output shelf3;
  if shelf = 4 then output shelf4;
run;
```

Chris Bilder, STAT 850
Cereal data adjusted for serving size

Obs	ID	Shelf	Cereal	sugar	fat	sodium
1	1	1	Kellog's Razzle Dazzle Rice Crispies	0.35714	0.000000	6.0714
2	2	1	Post Toasties Corn Flakes	0.07143	0.000000	9.6429
3	3	1	Kellogg's Corn Flakes	0.07143	0.000000	10.7143
4	4	1	Food Club Toasted Oats	0.06250	0.062500	8.7500
5	5	1	Frosted Cheerios	0.43333	0.033333	7.0000
6	6	1	Food Club Frosted Flakes	0.35484	0.000000	5.8065
7	7	1	Capn Crunch	0.44444	0.055556	7.4074
8	8	1	Capn Crunch's Peanut Butter Crunch	0.33333	0.092593	7.4074
9	9	1	Post Honeycomb	0.37931	0.017241	7.5862
10	10	1	Food Club Crispy Rice	0.06061	0.000000	10.0000
11	11	2	Rice Crispies Treats	0.30000	0.050000	6.3333

These four separate data sets are then merged together using a `merge` statement to form a new data set that is used for plotting.

```
data set2;
  merge shelf1(keep=sugar fat rename=(sugar=sugar1 fat=fat1))
        shelf2(keep=sugar fat rename=(sugar=sugar2 fat=fat2))
        shelf3(keep=sugar fat rename=(sugar=sugar3 fat=fat3))
        shelf4(keep=sugar fat rename=(sugar=sugar4 fat=fat4));
run;

title2 "Reformulated data set";
proc print data=set2(obs = 3);
run;
```

Chris Bilder, STAT 850
Reformulated data set

Obs	sugar1	fat1	sugar2	fat2	sugar3	fat3	sugar4	fat4
1	0.35714	0	0.30000	0.050000	0.31481	0.018519	0.34545	0.018182
2	0.07143	0	0.55556	0.018519	0.20000	0.050000	0.10000	0.000000
3	0.07143	0	0.46875	0.031250	0.06452	0.000000	0.34545	0.036364

Merge data sets by a variable

The merging of the data sets above represented a simple vertical concatenation. In other cases, one may want to merge data *by* a particular variable. This can be done thru using both the `merge` and `by` statements. Below are two simple data sets which are merged together (merge.sas):

```
data set1;
  input name$ response1;
  datalines;
  "a" 1
  "b" 2
  "c" 3
  "d" 4
  "e" 5
  "f" 6
  ;
run;

data set2;
  input name$ response2;
  datalines;
  "a" 10
  "a" 11
  "b" 20
  "c" 30
  "d" 40
  "e" 50
  ;
run;

data merg_set1;
  merge set1 set2;
  by name;
run;

title2 "Merged set";
proc print data=merg_set1;
run;
```

Chris Bilder, STAT 850
Merged set

Obs	name	response1	response2
1	"a"	1	10
2	"a"	1	11
3	"b"	2	20
4	"c"	3	30
5	"d"	4	40
6	"e"	5	50
7	"f"	6	.

Transpose a data set

Similar to transposing a matrix, one may need to transpose part of or all of a data set. This may occur when one needs to put a data set in the correct format for a SAS procedure to use. In particular, this often occurs when dealing with repeated measures data (more than one observation is taken on the same experimental unit). The procedure `proc transpose` provides a convenient tool for this purpose.

As an example, suppose a pharmaceutical company is conducting clinical trials on a new drug used to treat schizophrenia patients. Healthy male volunteers were given either 0, 3, 9, 18, 36, or 72mg of the drug. Before the drug was given (time = 0) and 1, 2, 3, and 4 hours after, a psychometric test called the Continuous Performance Test (CPT) was administered to each volunteer. The CPT involves the following:

- A subject sits in front a computer screen
- Randomly generated numbers appear on the computer screen
- Each image is slightly blurred
- One number appears every second for 480 seconds

- Subjects are required to press a button whenever the number 0 appears

We want to examine the number of hits (i.e., the number of correct responses). The response variable is the change in the number of hits from time 0. For example, patient 101 had

Time 0 hits - Time 1 hits = -9

Does the number of hits change after the drug is administered? If the number of hits goes down, this could mean the drug causes drowsiness, blurred vision, or some other detrimental effect. If the number of hits go up, possibly the drug acts as some type of stimulant.

Below is what a portion of the data looks like (repeated_measure.sas):

```
libname chris "C:\data";

data set1;
  set chris.cpt;
run;

title2 "The original data set";
proc print data=set1(obs=10);
run;
```

Chris Bilder, STAT 850
The original data set

Obs	PATIENT	TIME	DOSE	C_HITS
1	101	1	0	-9
2	101	2	0	-11
3	101	3	0	6
4	101	4	0	-1
5	205	1	0	4
6	205	2	0	-4
7	205	3	0	10
8	205	4	0	9
9	302	1	0	-5
10	302	2	0	0

The response variable is `c_hits`.

The goal is create a new form of the data such that all `c_hits` values are put in a row for the same patient. Below is the process:

```
proc sort data=set1;
  by patient dose;
run;

*Create 4 variables for the time 1,2,3,4 changes;
proc transpose data=set1 out=set2 prefix=time name=response;
  var c_hits;
  by patient dose;
run;

proc sort data=set2;
  by dose;
run;

title2 "The transposed data";
proc print data=set2(obs=5);
run;

title2 "Means over time for each dose group";
proc means data=set2 mean;
```

```
  class dose;
  var time1-time4;
run;
```

Chris Bilder, STAT 850
The transposed data

Obs	PATIENT	DOSE	response	time1	time2	time3	time4
1	101	0	C_HITS	-9	-11	6	-1
2	205	0	C_HITS	4	-4	10	9
3	302	0	C_HITS	-5	0	-5	-9
4	404	0	C_HITS	-4	2	0	0
5	102	3	C_HITS	-2	2	2	-1

Chris Bilder, STAT 850
Means over time for each dose group

The MEANS Procedure

DOSE	N Obs	Variable	Mean
0	4	time1 time2 time3 time4	-3.5000000 -3.2500000 2.7500000 -0.2500000
3	3	time1 time2 time3 time4	-2.3333333 2.0000000 7.6666667 14.0000000
9	4	time1 time2 time3 time4	9.5000000 6.2500000 32.5000000 19.0000000
18	4	time1 time2 time3 time4	0 4.5000000 1.2500000 6.5000000
36	4	time1 time2 time3 time4	-2.0000000 0 23.2500000 27.2500000
72	3	time1 time2 time3 time4	10.6666667 2.6666667 40.0000000 26.0000000

Note that to find the means at each time point, one can use `var time1-time4` in `proc means` rather than `var time1 time2 time3 time4`. This type of syntax can be used in any procedure. Also, additional code is given in the program to show how a plot of the means can be constructed.

General functions

Summary

Simple mathematical operations can be performed within a datastep (general_func.sas):

```
data set1;
  input x1 x2 x3;
  datalines;
  1 2 3
  4 5 6
  ;
run;

data set2;
  set set1;
  sum1 = x1 + x2 + x3;
  sum2 = sum(x1, x2, x3);
  sum3 = sum(of x1-x3);
  sqrt1 = sqrt(x1);
  max1 = max(x1, x2, x3);
  max2 = max(of x1-x3);
run;

title2 "Illustrate the sum and max functions";
proc print data=set2;
run;
```

Chris Bilder, STAT 850
Illustrate the sum and max functions

Obs	x1	x2	x3	sum1	sum2	sum3	sqrt1	max1	max2
1	1	2	3	6	6	6	1	3	3
2	4	5	6	15	15	15	2	6	6

Help for these and other functions is available at

Base SAS
Base SAS Software in the Windowing Environment
Base SAS 9.4 Procedures Guide, Third Edition
Base SAS 9.4 Procedures Guide: Statistical Procedures, Third E
Base SAS 9.4 Procedures Guide: High-Performance Procedure
SAS 9.4 Language Reference: Concepts, Third Edition
SAS 9.4 Component Objects Reference, Second Edition
SAS 9.4 Data Set Options: Reference, Second Edition
SAS 9.4 Formats and Informats: Reference
SAS 9.4 Functions and CALL Routines: Reference, Third Editio
Title Page
What's New in SAS 9.4 Functions and CALL Routines
About This Book
SAS Functions and CALL Routines
Definitions of Functions and CALL Routines
Syntax
Using Functions and CALL Routines
Function Compatibility with SBCS, DBCS, and MBCS C
Using Random-Number Functions and CALL Routines
Using SYSRANDOM and SYSRANEND Macro Variables
Date and Time Intervals
Pattern Matching Using Perl Regular Expressions (PRX)
Using Perl Regular Expressions in the DATA Step
Writing Perl Debug Output to the SAS Log
Perl Artistic License Compliance
Base SAS Functions for Web Applications
Dictionary of SAS Functions and CALL Routines
SAS Functions and CALL Routines Documented in Oth
SAS Functions and CALL Routines by Category
ABS Function
ADDR Function

Previous Page | Next Page

Dictionary of SAS Functions and CALL Routines

SAS Functions and CALL Routines Documented in Other SAS Publications

SAS Functions and CALL Routines by Category

A B C D E F G H I J K L M N O P Q R S T U V

ABS Function
ADDR Function
ADDRLONG Function
AIRY Function
ALLCOMB Function
ALLPERM Function
ANYALNUM Function
ANYALPHA Function
ANYCNTRL Function
ANYDIGIT Function
ANYFIRST Function
ANYGRAPH Function
ANYLOWER Function
ANYNAME Function
ANYPRINT Function

Probability distributions - quantiles, probabilities, and random number generation

Statistical software packages have made the use of "statistical tables" obsolete for a long time! Below are examples of how to calculate quantiles and probabilities from a standard normal distribution.

```
*Quantile from a standard normal distribution;
data set1;
```

```
  input area_to_left;
  quant1 = probit(area_to_left);
  quant2 = quantile("normal", area_to_left , 0, 1);
  datalines;
    0.975
  ;
run;

title2 "Standard normal quantiles";
proc print data=set1;
run;

*Probability from a standard normal distribution;
data set2;
  set set1;
  prob1 = probnorm(quant1);
  prob2 = CDF("normal", quant1 , 0, 1);
run;

title2 "Standard normal probabilities";
proc print data=set2;
run;
```

Chris Bilder, STAT 850
Standard normal quantiles

Obs	area_to_left	quant1	quant2
1	0.975	1.95996	1.95996

Chris Bilder, STAT 850
Standard normal probabilities

Obs	area_to_left	quant1	quant2	prob1	prob2
1	0.975	1.95996	1.95996	0.975	0.975

Comments:

- Notice how functions were included in the initial datastep which included `datalines`.

- With each probability distribution, there is a distribution-specific function (`probit`, `probnorm`) and a more general function (`quantile`, `CDF`) that can be used with a large number of probability distributions.
- The use of `PDF("normal", quant1, 0, 1)` finds the density value (height of curve, $f(x)$) for the standard normal density function.

Simulating a sample from a population characterized by a particular probability distribution is a very useful tool for statistical research and for illustrating statistical theory in an educational setting. For example, one can estimate the "true" confidence level for a confidence interval through simulating many samples via Monte Carlo simulation. We will examine this in more detail shortly. For now, below is an example of how to simulate an observation from a standard normal distribution through using the `rand` function.

```
*No seed set;
data ranset1;
  x = rand("normal", 0, 1);
run;

title2 "Simulated observation from a normal distribution";
proc print data=ranset1;
run;

*Set a seed so that observation can be reproduced;
data ranset2;
  call streaminit(7812);
  x = rand("normal", 0, 1);
run;

title2 "Simulated observation from a normal distribution, set a
  seed first";
proc print data=ranset2;
run;
```

Below is the output from running the code twice:

Chris Bilder, STAT 850
Simulated observation from a normal distribution

Obs	x
1	0.86163

Chris Bilder, STAT 850
Simulated observation from a normal distribution, set a seed first

Obs	x
1	-0.71818

Chris Bilder, STAT 850
Simulated observation from a normal distribution

Obs	x
1	0.036729

Chris Bilder, STAT 850
Simulated observation from a normal distribution, set a seed first

Obs	x
1	-0.71818

The difference between the two runs is that `ranset2` always contains the same observed value. This is because a seed number was set prior to using the `rand` function. This seed number intializes the random number generator so that the same observed value will occur each time. Seed numbers should always be set prior to any type of random number generation for reproducibility of results purposes. Note that a *call routine* was used to set the seed number. These are similar to functions, but a variable cannot be assigned a resulting value.

Comments:

- The `rand` function can be used with many other probability distributions. Please see the help for this function.
- Similar to what we saw with finding quantiles and probabilities, there are distribution-specific functions available for simulating observations. For example, the `rannor(<seed number>)` and `normal(<seed number>)` could be used to simulate observations from a population characterized by a standard normal distribution.
- In most cases, one will want to simulate more than just one observation. The next section discusses how to simulate more than one!

Loops

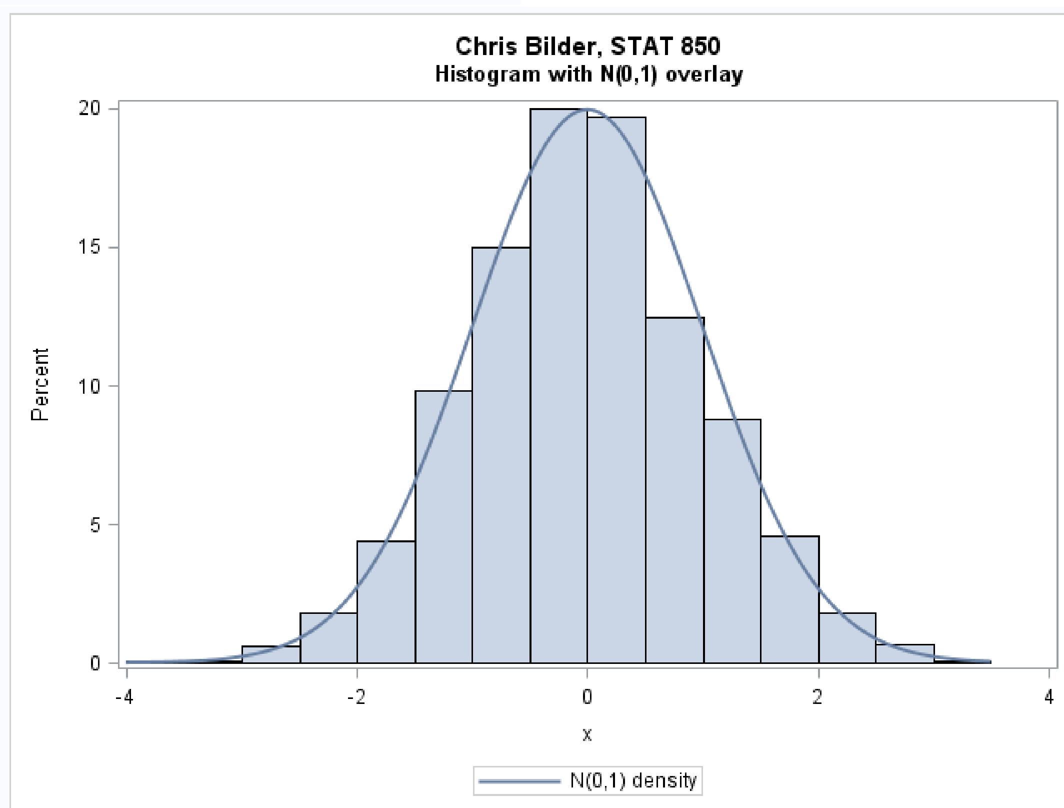
Loops are used to repeat the same set of code a number of times (i.e., iterations). The main way this is done in SAS is through a do-end statement. Below is a simple example illustrating the process through simulating 1,000 observations from a population characterized by a standard normal distribution:

```
data ranset3;
  call streaminit(1221);
  do i = 1 to 1000;
    x = rand("normal", 0, 1);
    output;
  end;
run;

title2 "First few simulated observations";
proc print data=ranset3(obs=5);
run;

title2 "Histogram with N(0,1) overlay";
proc sgplot data=ranset3;
  histogram x;
```

```
  density x / type=normal(mu=0 sigma=1) legendlabel="N(0,1)
     density";
run;
```

Chris Bilder, STAT 850
First few simulated observations

Obs	i	x
1	1	0.39450
2	2	-1.62930
3	3	-0.07863
4	4	-1.26361
5	5	-0.04255



Comments:

- The use of do-end is somewhat similar to how it was used with if-then-else earlier.
- The `i` in the do-end statement becomes a variable in the data set.
- The `output` statement is used as was shown earlier in the

notes, but no new data set is specified after it. This causes the observations to be put into the current data set. Without any `output` statement, only the last observation simulated would be put into the data set.

A somewhat more complex example involves the same type of data simulation again as part of a Monte Carlo simulation to estimate the true confidence level of a t-distribution based confidence interval for a population mean. The main goal for any confidence interval is for this estimated true confidence level to be close to the *stated* confidence level. Thus, if the stated confidence level is 95% for a particular type of interval, we would like this interval to contain or "cover" the parameter value approximately 95% of the time.

Here's a summary of the algorithm used for the Monte Carlo simulation:

- Simulate 1,000 data sets
- Calculate the confidence interval for each data set
- Check if each confidence interval contains the true population mean μ

The percentage of times overall that the confidence interval contains μ is the *estimated* true confidence level. Below is my SAS code used to implement this algorithm when using a 95% confidence interval.

```
data ranset4;
  call streaminit(1221);
  do simnumb = 1 to 1000;
    do n = 1 to 10;
      x = rand("normal", 0, 1);
      output;
    end;
   end;
run;
```

```
title2 "First few simulated observations";
proc print data=ranset4(obs=12);
run;

proc means data=ranset4 noprint mean std;
  var x;
  by simnumb;
  output out=out_set1 mean=mean std=sd;
run;

*Calculate 95% intervals using t_1-alpha/2,n-1 and check if
   interval contains mu=0;
data out_set2;
  set out_set1;
  lower = mean - quantile("t", 0.975, 10-1)*sd/sqrt(10);
  upper = mean + quantile("t", 0.975, 10-1)*sd/sqrt(10);
  if lower < 0 and upper > 0 then check1 = 1;
    else check1 = 0;

  *Another way to check;
  check2 = 0;
  if lower < 0 then if upper > 0 then check2 = 1;

  *One more way to check;
  if lower < 0 then if upper > 0 then check3 = 1;
      else check3 = 0;
    else check3 = 0;
run;

title2 "Estimated true confidence level";
proc means data=out_set2 mean;
  var check;
run;
```

Chris Bilder, STAT 850
First few simulated observations

Obs	simnumb	n	x
1	1	1	0.39450
2	1	2	-1.62930
3	1	3	-0.07863
4	1	4	-1.26361
5	1	5	-0.04255
6	1	6	-0.02415
7	1	7	-1.52169
8	1	8	-1.88170
9	1	9	-0.71359
10	1	10	0.78851
11	2	1	-1.66221
12	2	2	2.64056

Chris Bilder, STAT 850
Estimated true confidence level

The MEANS Procedure

Variable	Mean
check1	0.9560000
check2	0.9560000
check3	0.9560000

Comments:

- Notice the use of nested do-end statements. The `simnumb` variable denotes the number of the simulated data set.
- The estimate true confidence level is 0.956 which is quite close to the stated 95% confidence level. Why should this be expected in this situation?

Additional items

- To create a variable in a data set that represents the observation number, one can have SAS use the row number of the data set instead if the data is arranged in an appropriate way. The syntax `_n_` represents the row number in SAS. Thus,

  ```
  data set2;
    set set1;
    obs = _n_;
  run;
  ```

  will include the row number as the new `obs` variable in the data set.

- Sounds can be made by using a sound call routine. This can be useful to let you know when a long running program has been completed. Please see my general_func.sas program for an example.

- There are a number of time and date functions available. These can be helpful to use before and after a set of long running code to help track the amount of time the code takes to run. Below is a simple datastep used to find the current time.

  ```
  data time;
    time = hour(time());
    minute = minute(time());
    second = second(time());
    month = month(today());
    day = day(today());
    year = year(today());
  run;
  ```