



SAS® Tech Report

MAY 13, 2008

Dear Readers,

I'm very excited about this, my first, issue of the SAS® *Tech Report*. I've introduced myself to my colleagues, delved into the world of content found on support.sas.com, and subscribed to a few SAS® news and support feeds. Now I have the pleasure of introducing myself to you. All of this is with one goal in mind: I want to make sure that I continue to bring to you the content that you need and want. So, don't hesitate to [send me a note](#) and tell me what's on your mind.

In this edition of the *SAS Tech Report*, we'll continue to highlight the top papers from SAS Global Forum. Greg McLean's paper will show you how to add extra precision to your GUI applications, and Candice Riley reports a method for accessing the SAS environment and sharing data from multiple platforms.

There's still time to register for [PharmaSUG 2008 – the Pharmaceutical Industry SAS® Users Group Conference](#). The conference, held June 1-4, focuses on education and training. The keynote address, "Life Sciences and Analytics: SAS 2.0," will be given by David Handelsman, R&D Manager of the new SAS Health and Life Sciences Global Practice. And lastly, although it may seem far into the future, it's time to begin thinking about attending [M2008 – the 11th annual SAS® Data Mining Conference](#). The conference will be held at Caesars Palace in Las Vegas, Oct. 27-28, with pre-conference workshops Oct. 26 and post-conference training starting Oct. 29. Check out the links for other upcoming events as well.

Waynette Tubbs

Editor, *SAS Tech Report*

SAS News

Global Forum Wrap: Highlighting Great Papers from the April Event

Over the next few issues we will be highlighting some of the most interesting papers presented at the SAS Global Forum. Get a list of [award-winning papers](#) or check out the [complete list](#) of 401 papers.

Great Forum Paper Series: Picture Perfect – Centering Frames in SAS/AF®

Graphical user interface (GUI) applications, when done with precision, seem an almost invisible part of the user experience. But what if the GUI is off-center or has overlapping frames? Greg McLean's paper illustrates his simple method for centering GUI frames to improve user productivity, acceptability and ease of use.

<http://www2.sas.com/proceedings/forum2008/009-2008.pdf>

Great Forum Paper Series: Getting SAS® to Play Nice with Others: Connecting SAS® to Microsoft SQL Server Using an ODBC Connection

Candice Riley shows how to use SAS/ACCESS®, an ODBC connection and a couple of third-party tools to connect a Microsoft SQL Server from both PC and Linux environments to SAS and seamlessly share data without ever having to leave the SAS environment.

<http://www2.sas.com/proceedings/forum2008/135-2008.pdf>

Create a New Data Set for each BY-Group Within the Larger Data Set

This sample uses macro logic to determine the number of unique values of a variable (the BY variable) and creates a new data set for each. The resulting data set names will be the BY variable value.

Limitations: The sample code does not allow for BY values of longer than 32 positions, numeric BY values or BY values that contain characters that are not permitted in SAS data set names.

If your data contains any of the above, you must add program statements to convert your BY values into valid SAS data set names.

FULLCODE:

```
/* Example 1 - Use macro logic to create a new data set for each BY-Group in */  
/* an existing data set. */
```

```
/* Create sample data */
```

```
data test;  
  input color $ num;  
datalines;  
blue 1  
blue 2  
blue 3  
green 4  
green 5  
red 6  
red 7  
red 8  
;
```

```
/* Create a new macro variable, VARn, for each BY-Group and a */  
/* counter of the number of new macro variables created. */
```

```
data _null_;  
  set test end=eof;  
  by color;  
  /* On the first member of the BY-Group, create a new macro variable VARn */  
  /* and increment the counter FLAG. */  
  if first.color then do;  
    flag+1;  
    call symput('var'||put(flag,8. -L),color);  
  end;  
  /* On the last observation of the data set, create a macro variable to */  
  /* contain the final value of FLAG. */  
  if eof then call symput('tot',put(flag,8. -L));  
run;
```

```
/* Create a macro to generate the new data sets. Dynamically produce data set names */  
/* on the DATA statement, using subsetting criteria to create the new data sets */  
/* based upon the value of the BY variable. */
```

```
%macro groups(dsn,byvar);  
  data %do i=1 %to &tot;  
    &&var&i  
  %end;;  
  set &dsn;
```

```

    %do i=1 %to &tot;
      if &byvar="&&var&i" then output &&var&i;
    %end;
run;
%mend groups;

/* Call the macro GROUPS. Specify the name of the data set to be split */
/* in the first macro parameter and the name of the BY variable in the */
/* second parameter. */

%groups(test,color)

proc print data=blue;
  title 'Blue';
run;

proc print data=green;
  title 'Green';
run;

proc print data=red;
  title 'Red';
run;

/* Example 2 - Use CALL EXECUTE to pass a parameter to a macro in order to */
/* create a new data set for each BY-Group in an existing data */
/* set. The output is identical to the output created by */
/* Example 1 above. */

/* Compile the macro BREAK. The parameter BYVAL will be generated below in */
/* the CALL EXECUTE. */

%macro break(byval);
  data &byval;
    set test(where=(color="&byval"));
  run;
%mend;

/* Use the same TEST data set created for Example 1. */

data _null_;
  set test;
  by color;
  if first.color then
    call execute(%nrstr('%break('||trim(color)||')'));

run;

```

RESULTS:

Blue

Obs	color	num
-----	-------	-----

1	blue	1
2	blue	2

3 blue 3

Green

Obs color num

1 green 4
2 green 5

Red

Obs color num

1 red 6
2 red 7
3 red 8

Create multiple SAS data sets from one SAS data set based upon the value of the BY variable.

Type: Sample

Topic: SAS Reference ==> Statements ==> File-handling ==> SET
SAS Reference ==> DATA Step
SAS Reference ==> Macro
Data Management ==> Manipulation and Transformation ==> BY-group processing

Date Modified: 2008-05-06 10:32:44

Date Created: 2007-02-14 03:04:16

Operating System and Release Information

Product Family	Product	Host	Starting Release	Ending Release
SAS System	Base SAS	All	8.2 TS2M0	n/a

When Do I Use a WHERE Statement vs. an IF Statement to Subset a Data Set?

When programming in SAS, there is almost always more than one way to accomplish a task. Beginning programmers may think that there is no difference between using the WHERE statement and the IF statement to subset your data set. Knowledgeable programmers know that depending on the situation, sometimes one statement is more appropriate than the other. For example, if your subset condition includes automatic variables or new variables created within the DATA step, then you must use the IF statement instead of the WHERE statement. This tip shows you how and when to apply the WHERE and IF statements to get correct and reliable results. It also reviews the similarities as well as the differences between these two SAS programming approaches. Detail differences in program efficiency between the two approaches will not be covered in this tip.

The following code creates the sample data set that is used in the examples. The data set contains test scores of three classes from three students.

```
data exam;
  input name $ class $ score ;
  cards;
Tim math 9
Tim history 8
Tim science 7
Sally math 10
Sally science 7
Sally history 10
John math 8
John history 8
John science 9
;
run;
```

Key Differences between WHERE and IF Conditions to Subset Data Sets

Below is a table that summarizes the key differences between WHERE and IF Conditions. The examples following this table show some of these differences. The source for this table is *Sharpening Your SAS Skills*, CRC Press (www.crcpress.com), April 2005.

Subset Data set	WHERE	IF
No Difference between WHERE and IF Conditions		
Using variables in data set	X	X
Using SET, MERGE, or UPDATE statement if within the DATA step*	X	X
Must use IF Condition		
Accessing raw data file using INPUT statement		X
Using automatic variables such as _N_, FIRST.BY, LAST.BY		X
Using newly created variables in data set		X
In combination with data set options such as OBS		X

=, POINT = , FIRSTOBS =		
To conditionally execute statement		X
Must use WHERE Condition		
Using special operators such as LIKE or CONTAINS	X	
Directly using any SAS Procedure	X	
More efficiently**	X	
Using index, if available	X	
When subsetting as a data set option	X	
When subsetting using PROC SQL	X	
Be careful which you use!		
When merging data sets***	SUBSET BEFORE MERGING	SUBSET AFTER MERGING

Notes:

* WHERE condition requires one of these statements if used within the DATA step. In addition, the variable specified in the WHERE condition must exist in all data sets.

** OBS = data set option is compatible with the WHERE statement in SAS version 8.1 and higher.

When OBS = is used with the IF statement, SAS first subsets the data set based on the number of observations in the OBS = option and then applies the IF subset condition. When OBS = is used with the WHERE statement, SAS first applies the WHERE subset condition and then restricts the output data set to contain the maximum of observations as specified in the OBS = option.

*** The Colon Modifier (:) works with the IF statement to compare shorter text with longer text.

**** WHERE condition may be more efficient because SAS is not required to read all observations from the input data set.

***** Results may be different depending on the data sets being merged. In general, use the IF condition to subset the data set after merging the data sets.

First example

In the first example, the data set contains both a WHERE and IF statement to subset the data set. The records in the new data set, student1, will contain those records that meet both WHERE and IF conditions.

```
data student1;
  set exam;
```

* Can use WHERE condition because NAME variable is a data set variable;

* WHERE condition requires all data set variables;
where name = 'Tim' or name = 'Sally';

* Create CLASSNUM variable;
if class = 'math' then classnum = 1;

```

else if class = 'science' then classnum = 2;
else if class = 'history' then classnum = 3;

* Use IF condition because CLASSNUM variable was created within the DATA step;
if classnum = 2;
run;

proc print data = student1;
run;

```

As you can see below, both records meet both conditions. Subset condition 1 required Tim's or Sally's scores. Since either the WHERE or IF statement could have been used in condition 1, using the WHERE statement would be more efficient. Subset condition 2 required the CLASSNUM variable to equal 2 or any science score. The IF statement must be used in condition 2 because the CLASSNUM variable is created in the DATA step.

The results...

Obs	name	class	score	classnum
1	Tim	science	7	2
2	Sally	science	7	2

Second example

In the second example, the exam data set is sorted by the NAME variable. The DATA step uses the IF statement to keep the FIRST.NAME record, because the WHERE statement can not be applied to FIRST.BY or LAST.BY variables. Variables in the WHERE statement must exist in the data set and can not be temporary variables.

```

proc sort data = exam out=student2;
  by name;
run;

data student2;
  set student2;
  by name;

* Use IF condition because NAME is the BY variable;
if first.name;

run;

proc print data = student2;
run;

```

As you can see below, there is only one record for each student. This meets the subset condition to keep only the first record for each unique value of the student's name. Since the data set stored the math test scores as the first record for each student, this is the only subject in the student2 data set.

The results...

Obs	name	class	score
1	John	math	8
2	Sally	math	7
3	Tim	math	9

Third example

The third example uses the two sample data sets below to show the difference in using WHERE and IF statements when merging data sets:

```
data school;          data school_data;
  input name $ class $ score ;    input name $ class $ score ;
cards;          cards;
A math 10          A math 10
B history 10        B history 8
C science 10        C science 7
;          ;
run;          run;
```

Below are two DATA steps using WHERE or IF statement:

```
data school_where;    data school_if;
  merge school school_data;    merge school school_data;
  by name;          by name;

  * subsets BEFORE merging;    * subsets AFTER merging;
  where score = 10;    if score = 10;
run;          run;
```

Since the WHERE statement applies the subset condition before merging the data sets, all records from the school data set are selected and only the first record from the school_data data set is selected. Because of this, all records from the first data set are kept in the school_where data set. In general, you will want to use the IF statement to apply the subset condition after merging the data sets. This approach will first merge the two data sets and then apply the subset condition to result in the first record of both school and school_data sets.

Below are the two different data sets from using WHERE and IF statements. Be careful to use the correct subset method since the results could be very different.

```
school_where data set    school_if data set
A math 10          A math 10
B history 10
C science 10
```

Make sure you apply the following rules when determining which approach to take when subsetting your data set using the DATA step. If your subset condition does not meet the requirements below, then the WHERE and IF statements should produce identical results. For cases such as this, use the WHERE statement since it is more efficient. Note that having both WHERE and IF statements within the same DATA step has a cumulative effect.

- Can use WHERE statement when only specifying data set variables
- Use IF statement when specifying automatic variables or new variables created within DATA step
- Use IF statement when specifying FIRST.BY or LAST. BY variables
- Use IF statement when specifying data set options such as OBS = , POINT = or FIRSTOBS =
- In general, use IF statement when merging data sets to apply subset condition after merging data set
- Use WHERE statement when specifying indexes

About the Author

Sunil Gupta is the principal consultant and trainer at Gupta Programming (www.GuptaProgramming.com). You can find this tip along with other useful techniques in Sunil's

latest book, *Sharpening Your SAS Skills* (available from [SAS Press](http://www.saspress.com) and from www.crcpress.com).

Operating System and Release Information

Product Family	Product	System	Reported Release	Fixed Release*
SAS System	Base SAS	All	n/a	

* For software releases that are not yet generally available, the Fixed Release is the software release in which the problem is planned to be fixed.

Jackknife and Bootstrap Analyses

PURPOSE:

The %JACK macro does jackknife analyses for simple random samples, computing approximate standard errors, bias-corrected estimates, and confidence intervals assuming a normal sampling distribution.

The %BOOT macro does elementary nonparametric bootstrap analyses for simple random samples, computing approximate standard errors, bias-corrected estimates, and confidence intervals assuming a normal sampling distribution. Also, for regression models, the %BOOT macro can resample either observations or residuals.

The %BOOTCI macro computes several varieties of confidence intervals that are suitable for sampling distributions that are not normal.

REQUIREMENTS:

The macros require only Base SAS software. However, to apply them a statistic computed by a procedure in another product, such as a procedure in SAS/STAT software, that product will also be required.

USAGE:

Follow the instructions in the Downloads tab of this sample to save the %JACK and %BOOT macro definitions. Replace the text within quotes in the following statement with the location of the macro definitions file on your system. In your SAS program or in the SAS editor window, specify this statement to define the macros and make them available for use:

```
%inc "<location of your file containing the JACK and BOOT macros>";
```

Following this statement, you may call the %JACK and %BOOT macros. See the Results tab for an example.

To use the %JACK or %BOOT macros, you must write a macro called %ANALYZE to do the data analysis that you want to bootstrap. The %ANALYZE macro must have two arguments:

```
DATA= the name of the input data set to analyze  
OUT= the name of the output data set containing the statistics  
for which you want to compute bootstrap distributions.
```

The %ANALYZE macro is run once by the %JACK or %BOOT macros to analyze the original data set. Then the resampled data sets are generated, and the %JACK or %BOOT macros run the %ANALYZE macro again to analyze each resample. There are two ways to analyze the resamples:

```
with a macro loop  
with BY processing
```

If you don't do anything special, a macro loop will be used. A macro loop takes much more computer time than BY processing but requires less disk space. It is usually better to use BY processing unless you have run out of disk space.

To use BY processing for the resamples, you must write the %ANALYZE macro to use BY processing. But instead of having ordinary BY statements in the %ANALYZE macro, you must use the %BYSTMT macro instead, which automatically generates an appropriate BY statement or not; that is, for the analysis of the resamples, the %BYSTMT macro generates a BY statement, but for the analysis of the original data, the %BYSTMT macro produces no BY statement (actually, it produces an empty BY statement, which has the same effect as not having a BY statement).

If the %ANALYZE macro uses the %BYSTMT macro, the %JACK and %BOOT macros create a huge data set (actually a view) containing all of the resampled data sets, and analyze all the resamples by running the %ANALYZE macro once with BY processing. If the %ANALYZE macro does not use the %BYSTMT macro, the %JACK and %BOOT macros execute a macro loop that generates and analyzes the resamples one at a time.

BY Processing in the Analysis

Ordinarily, you do not need to worry about what happens inside the %BYSTMT macro. However, if the analysis of the original data set requires BY processing, you will have to modify the %BYSTMT macro to include your BY variable(s) as well as the variable that the %JACK and %BOOT macros use to distinguish the resamples (which is referred to as &by in the %BYSTMT macro). If you're not in a hurry, you may find it simpler to forego use of the %BYSTMT macro.

DETAILS:

Introduction

The %JACK macro does jackknife analyses for simple random samples, computing approximate standard errors, bias-corrected estimates, and confidence intervals assuming a normal sampling distribution.

The %BOOT macro does elementary nonparametric bootstrap analyses for simple random samples, computing approximate standard errors, bias-corrected estimates, and confidence intervals assuming a normal sampling distribution. Also, for regression models, the %BOOT macro can resample either observations or residuals.

The %BOOTCI macro computes several varieties of confidence intervals that are suitable for sampling distributions that are not normal.

In order to use the %JACK or %BOOT macros, you need to know enough about the SAS macro language to write simple macros yourself. See *The SAS Guide to Macro Processing* for information on the SAS macro language.

This document does not explain how the jackknife and bootstrap are performed or how the various confidence intervals are computed, but does provide some advice and caveats regarding usage. For an elementary introduction, see Dixon in the bibliography below. There is a thorough exposition in E&T (see references below) that should be accessible to anyone who has done a year or more of statistical study.

There is a widespread myth that bootstrapping is a magical spell to perform valid statistical inference on *anything*. S&T dispell this myth very effectively and very technically. For an elementary demonstration of the dangers of bootstrapping, see the "Cautionary Example" below.

The Jackknife

The jackknife works only for statistics that are smooth functions of the data. Statistics that are not smooth functions of the data, such as quantiles, may yield inconsistent jackknife estimates. The best results are obtained with statistics that are linear functions of the data. For highly nonlinear statistics, the jackknife can be inaccurate. See S&T, chapter 2, for a detailed discussion of the validity of the jackknife.

The Bootstrap

Bootstrap estimates of standard errors are valid for many commonly-used statistics, generally requiring no major assumptions other than simple random sampling and finite variance. There do exist some statistics for which the standard error estimates will fail, such as the maximum or minimum. The bootstrap standard error is consistent for some nonsmooth statistics such as the median. However, the bootstrap standard error may not be consistent even for very smooth

statistics when the population distribution has very heavy tails. Inconsistency of the usual bootstrap estimators can often be remedied by using a resample size $m(n)$ that is smaller than the sample size n , so that $m(n) \rightarrow \infty$ and $m(n)/n \rightarrow 0$ as $n \rightarrow \infty$. Theoretical results on the consistency of the bootstrap standard error are not extensive. See S&T, chapter 3, for details.

The bootstrap estimates of bias provided by the %BOOT macro are valid under simple random sampling for many commonly-used *plug-in* estimators. A *plug-in* estimator is one that uses the same formula to compute an estimate from a sample that is used to compute a parameter from the population. For example, if the sample variance is computed with a divisor of n (VARDEF=N), it is a plug-in estimate; if it is computed with a divisor of $n-1$ (VARDEF=DF, the default), it is *not* a plug-in estimate. R^2 is a plug-in estimator; adjusted R^2 is not. Estimating the bias of a non-plug-in estimator requires special treatment; see "Bias Estimation" below. If you are using an estimator that is known to be unbiased, use the BIASCORR=0 argument with %BOOT. See E&T, chapter 10, for more discussion of bootstrap estimation of bias.

The approximate normal confidence intervals computed by the %BOOT macro are valid if both the bias and standard error estimates are valid and if the sampling distribution is approximately normal. For non-normal sampling distributions, you should use the %BOOTCI macro, which requires a much larger number of resamples for adequate approximation. If you plan to use only %BOOT, 200 resamples will typically be enough. If you plan to use %BOOTCI, 1000 or more resamples are likely to be needed for a 90% confidence interval; greater confidence levels require even more resamples. The proper use of bootstrap confidence intervals is a matter of considerable controversy; see S&T, chapter 4, for a review.

The %BOOT macro does balanced resampling when possible. Balanced resampling yields more accurate approximations to the ideal bootstrap estimators of bias and standard errors than does uniform resampling. Of course, both balanced resampling and uniform resampling produce approximations that converge to the same ideal bootstrap estimators as the number of resamples goes to infinity. Balanced resampling is of little benefit with %BOOTCI. See Hall, appendix II, for a discussion of balanced resampling and other methods from improving the computational efficiency of the bootstrap.

Output Data Sets

If the %ANALYZE macro uses the %BYSTMT macro, two output data sets are created by the %JACK macro:

JACKDATA

contains the jackknife resamples. The variable `_SAMPLE_` gives the resample number, and `_OBS_` gives the original observation number.

JACKDIST

contains the resampling distributions of the statistics in the OUT= data set created by the %ANALYZE macro. The variable `_SAMPLE_` gives the resample number. Two similar data sets are also created by the %BOOT macro when the %BYSTMT macro is used:

BOOTDATA

contains the bootstrap resamples. The variable `_SAMPLE_` gives the resample number, and `_OBS_` gives the original observation number.

BOOTDIST

contains the resampling distributions of the statistics in the OUT= data set created by the %ANALYZE macro. The variable `_SAMPLE_` gives the resample number.

In addition, the %JACK macro creates a data set JACKSTAT and the %BOOT macro creates a data set BOOTSTAT regardless of whether the %BYSTMT macro is used. These data sets

contain the approximate standard errors, bias-corrected estimates, and 95% confidence intervals assuming a normal sampling distribution. The %BOOTCI macro creates a data set BOOTCI containing the confidence intervals.

If the OUT= data set contains more than one observation per BY group, you must specify a list of ID= variables when you run the %JACK or %BOOT macros. These ID= variables identify observations that correspond to the same statistic in different BY groups. For many procedures, these ID= variables would naturally be _TYPE_ and _NAME_, but those names are *not* allowed to be used as ID= variables--you must use the RENAME= data set option to rename them. (Renaming variables can be tricky. You must use the *old* name with the DROP= and KEEP= data set options, but you must use the *new* name with the WHERE= data set option.)

Bias Estimation

The sample correlation is a plug-in estimator and hence is suitable for the bias estimator in %BOOT. The sample variance computed with a divisor of n-1 is not a plug-in estimator and therefore requires special treatment. In some procedures, you can use the VARDEF= option to obtain a plug-in estimate of the variance. The default value of VARDEF= is DF, which yields the usual adjustment for degrees of freedom, instead of the plug-in estimate. For example:

```
title2 'The unbiased variance estimator is not a plug-in estimator';
proc means data=law var vardef=df;
  var LSAT GPA;
run;
```

The following %ANALYZE macro could be used to jackknife the unbiased variance estimator, but the bootstrap over-corrects for the nonexistent bias:

```
title2 'Estimating the bias of the unbiased estimator of variance';
%macro analyze(data=,out=);
  proc means noprint data=&data vardef=df;
    output out=&out(drop=_freq__type_) var=var_LSAT var_GPA;
    var LSAT GPA;
    %bystmt;
  run;
%mend;

title3 'The jackknife computes the correct bias of zero';
%jack(data=law)

title3 'The bootstrap over-corrects for bias';
%boot(data=law,random=123)
```

By specifying VARDEF=N instead of VARDEF=DF, you can tell the MEANS procedure to compute a plug-in estimate of the variance:

```
title2 'Estimating the bias of the plug-in estimator of variance';
%macro analyze(data=,out=);
  proc means noprint data=&data vardef=n;
    output out=&out(drop=_freq__type_) var=var_LSAT var_GPA;
    var LSAT GPA;
    %bystmt;
  run;
%mend;
```

With the above %ANALYZE macro, %JACK yields an exact bias correction, while the bias-corrected estimates from %BOOT are very close to the unbiased estimates:

```
title3 'Jackknife Analysis';
%jack(data=law)
```

```
title3 'Bootstrap Analysis';
%boot(data=law,random=123)
```

If the procedure you are using supports the VARDEF= option to produce plug-in estimates, you can use the %VARDEF macro to obtain correct bootstrap bias estimates of the corresponding non-plug-in estimates. The %VARDEF macro generates a VARDEF= option with a value of either N or DF as appropriate for use with the %BOOT macro (The %JACK macro ignores the %VARDEF macro). In the %ANALYZE macro, use %VARDEF in the procedure statement where the VARDEF= option would be syntactically correct. For example:

```
title2 'Estimating the bias of the unbiased variance estimator';
%macro analyze(data=,out=);
proc means noprint data=&data %vardef;
  output out=&out(drop=_freq_ _type_) var=var_LSAT var_GPA;
  var LSAT GPA;
  %bystmt;
run;
%mend;

title3 'Bootstrap Analysis';
%boot(data=law,random=123)
```

The variance estimator using VARDEF=DF is unbiased, so the bias correction estimated by bootstrapping is much smaller than in the previous example, in which the biased plug-in estimator was used.

Confidence Intervals

The normal bootstrap confidence interval computed by %BOOT or %BOOTSE is accurate only for statistics with an approximately normal sampling distribution. The %BOOTCI macro provides the most commonly used types of bootstrap confidence intervals that:

are suitable for statistics with nonnormal sampling distributions and require only a single level of resampling.

You must run %BOOT before %BOOTCI, and it is advisable to specify at least 1000 resamples in %BOOT for a 90% confidence interval. For a higher level of confidence or for the BC and BCa methods, even more resamples should be used.

The terminology for bootstrap confidence intervals is confused. The keywords used with the %BOOTCI macro follow S&T:

Keyword	Terms from the references
-----	-----
PCTL or	"bootstrap percentile" in S&T;
PERCENTILE	"percentile" in E&T;
	"other percentile" in Hall;
	"Efron's 'backwards' percentile" in Hjorth
HYBRID	"hybrid" in S&T;
	no term in E&T;
	"percentile" in Hall;
	"simple" in Hjorth
T	"bootstrap-t" in S&T and E&T;

"percentile-t" in Hall;
"studentized" in Hjorth

BC "BC" in all

BCA "BCa" in S&T, E&T, and Hjorth; "ABC" in Hall
(cannot be used for bootstrapping residuals in
regression models)

There is considerable controversy concerning the use of bootstrap confidence intervals. To fully appreciate the issues, it is important to read S&T and Hall in addition to E&T. Asymptotically in simple random samples, the T and BCa methods work better than the traditional normal approximation, while the percentile, hybrid, and BC methods have the same accuracy as the traditional normal approximation. In small samples, things get much more complicated:

The percentile method simply uses the $\alpha/2$ and $1-\alpha/2$ percentiles of the bootstrap distribution to define the interval. This method performs well for quantiles and for statistics that are unbiased and have a symmetric sampling distribution. For a statistic that is biased, the percentile method amplifies the bias. The main virtue of the percentile method and the closely related BC and BCa methods is that the intervals are equivariant under transformation of the parameters. One consequence of this equivariance is that the interval cannot extend beyond the possible range of values of the statistic. In some cases, however, this property can be a vice--see the "Cautionary Example" below.

The BC method corrects the percentile interval for bias--median bias, not mean bias. The correction is performed by adjusting the percentile points to values other than $\alpha/2$ and $1-\alpha/2$. If a large correction is required, one of the percentile points will be very small; hence a very large number of resamples will be required to approximate the interval accurately. See the "Cautionary Example" below.

The BCa method corrects the percentile interval for bias and skewness. This method requires an estimate of the acceleration, which is related to the skewness of the sampling distribution. The acceleration can be estimated by jackknifing for simple random samples which, of course, requires extra computation. For bootstrapping residuals in regression models, no general method for estimating the acceleration is known. If the acceleration is not estimated accurately, the BCa interval will perform poorly. The length of the BCa interval is not monotonic with respect to α (Hall, pp 134-135, 137). For large values of the acceleration and large α , the BCa interval is excessively short. The BCa interval is no better than the BC interval for nonsmooth statistics such as the median.

The HYBRID method is the reverse of the percentile method. While the percentile method amplifies bias, the HYBRID method automatically adjusts for bias and skewness. The HYBRID method works well if the standard error of the statistic does not depend on any unknown parameters; otherwise, the T method works better if a good estimate of the standard error is available. Of all the methods in %BOOTCI, the HYBRID method seems to be the least likely to yield spectacularly wrong results, but often suffers from low coverage in relatively easy cases. The HYBRID method and the closely related T method are not equivariant under transformation of the parameters.

The T method requires an estimate of the standard error (or a constant multiple thereof) of each statistic being bootstrapped. This requires more work from the user. If the standard errors are not estimated accurately, the T method may perform poorly. In simulation studies, T intervals are often found to be very long. E&T (p 160) claim that the T method is erratic and sensitive to outliers.

Numerous other methods exist for bootstrap confidence intervals that require nested resampling, i.e., each resample of the original sample is itself resampled multiple times. Since the total number of resamples required is typically 25,000 or more, these methods are extremely expensive and have not yet been implemented in the %BOOT and %BOOTCI macros.

The following example replicates the nonparametric confidence intervals shown in E&T, p 183. This example analyzes the variances of two variables, A and B, while E&T analyze only A. E&T do not show the hybrid interval, the normal ("standard") interval with bias correction, or the jackknife interval.

```

title 'Spatial Test Data from Efron and Tibshirani, pp 180 & 183';
data spatial;
  input a b @@;
cards;
48 42 36 33 20 16 29 39 42 38 42 36 20 15 42 33 22 20 41 43 45 34
14 22 6 7 0 15 33 34 28 29 34 41 4 13 32 38 24 25 47 27 41 41
24 28 26 14 30 28 41 40
;

%macro analyze(data=,out=);
  proc means noprint data=&data vardef=n;
    output out=&out(drop=_freq__type_) var=var_a var_b;
    var a b;
    %bystmt;
  run;
%mend;

title2 'Jackknife Interval with Bias Correction';
%jack(data=spatial,alpha=.10);

title2 'Normal ("Standard") Confidence Interval with Bias Correction';
%boot(data=spatial,alpha=.10,samples=2000,random=123);

title2 'Normal ("Standard") Confidence Interval without Bias Correction';
%bootse(alpha=.10,biascorr=0);

title2 'Efron"s Percentile Confidence Interval';
%bootci(percentile,alpha=.10)

title2 'Hybrid Confidence Interval';
%bootci(hybrid,alpha=.10)

title2 'BC Confidence Interval';
%bootci(bc,alpha=.10)

title2 'BCa Confidence Interval';
%bootci(bca,alpha=.10)

title2 'Resampling with Computation of Studentizing Statistics';
%macro analyze(data=,out=);
  proc means noprint data=&data vardef=n;
    output out=&out(drop=_freq__type_)
      var=var_a var_b kurtosis=kurt_a kurt_b;
    var a b;
    %bystmt;

```

```

run;
data &out;
  set &out;
  stud_a=var_a*sqrt(kurt_a+2);
  stud_b=var_b*sqrt(kurt_b+2);
  drop kurt_a kurt_b;
run;
%mend;

%boot(data=spatial,stat=var_a var_b,samples=2000,random=123);

title2 'T Confidence Interval';
%bootci(t,stat=var_a var_b,student=stud_a stud_b,alpha=.10)

```

If you want to compute *all* the varieties of confidence intervals, you can use the %ALLCI macro:

```

title2 'All Jackknife and Bootstrap Confidence Intervals';
%allci(stat=var_a var_b,student=stud_a stud_b,alpha=.10)

```

Bootstrapping Regression Models

In regression models, there are two main ways to do bootstrap resampling, depending on whether the predictor variables are random or fixed. Stine provides an elementary introduction to bootstrapping regressions, including discussion of outliers, robust estimators, and heteroscedasticity.

If the predictors are random, you resample observations just as you would for any simple random sample. This method is usually called "bootstrapping pairs".

If the predictors are fixed, the resampling process should keep the same values of the predictors in every resample and change only the values of the response variable by resampling the residuals. To do this with the %BOOT macro, you must do a preliminary analysis in which you fit the regression model using the complete sample and create an output data set containing residuals and predicted values; it is this output data set that is used as input to the %BOOT macro. You must also specify the name of the residual variable and provide an equation for computing the response variable from the residual and predicted values.

```

title 'Cement Hardening Data from Hjorth, p 31';
data cement;
  input x1-x4 y;
  label x1='3CaOAl2O3'
        x2='3CaOSiO2'
        x3='4CaOAl2O3Fe2O3'
        x4='2CaOSiO2';
cards;
7 26 6 60 78.5
1 29 15 52 74.3
11 56 8 20 104.3
11 31 8 47 87.6
7 52 6 33 95.9
11 55 9 22 109.2
3 71 17 6 102.7
1 31 22 44 72.5
2 54 18 22 93.1
21 47 4 26 115.9
1 40 23 34 83.8
11 66 9 12 113.3
10 68 8 12 109.4

```

```

;

proc reg data=cement;
  model y=x1-x4;
  output out=cemout r=resid p=pred;
run;

%macro analyze(data=,out=);
  options nonotes;
  proc reg data=&data noprint
    outest=&out(drop=Y _IN_ _P_ _EDF_);
    model y=x1-x4/selection=rsquare start=4;
    %bystmt;
  run;
  options notes;
%mend;

title2 'Resampling Observations';
title3 '(bias correction for _RMSE_ is wrong)';
%boot(data=cement,random=123)

title2 'Resampling Residuals';
title3 '(bias correction for _RMSE_ is wrong)';
%boot(data=cemout,residual=resid,equation=y=pred+resid,random=123)

```

Either method of resampling for regression models (observations or residuals) can be used regardless of the form of the error distribution. However, residuals should be resampled only if the errors are independent and identically distributed and if the functional form of the model is correct to within a reasonable approximation. If these assumptions are questionable, it is safer to resample observations.

In the above example, R^2 is a plug-in estimator, so the bias correction is appropriate. The root mean squared error, `_RMSE_`, is not a plug-in estimator, so the bias correction for `_RMSE_` is wrong. Unfortunately, the REG procedure does not support the `VARDEF=` option. `_RMSE_` is not *very* biased, so you could choose to ignore the bias and run the `%BOOTSE` macro to compute the standard error without a bias correction:

```

title2 'Resampling Observations';
title3 'Without bias correction';
%bootse(stat=_rmse_,biacorr=0)

```

To get the proper bias correction for `_RMSE_`, you have to use a DATA step that checks the macro variable `&VARDEF` and unadjusts for degrees of freedom when `&VARDEF=N`. You must also invoke the `%VARDEF` macro, but since you don't want to generate a `VARDEF=` option in this case, just assign the value returned by `%VARDEF` to an unused macro variable:

```

%macro analyze(data=,out=);
  options nonotes;
  proc reg data=&data noprint outest=&out;
    model y=x1-x4/selection=rsquare start=4;
    %bystmt;
  run;
  %let junk=%vardef;
  data &out(drop=y _in_ _p_ _edf_);
    set &out;
    _mse_=_rmse_**2;

```

```

%if &vardef=N %then %do;
  _mse_=_mse_*_edf_/(_edf+_p_);
  _rmse_=sqrt(_mse_);
%end;
label _mse_='Mean Squared Error';
run;
options notes;
%mend;

title2 'Resampling Observations';
%boot(data=cement,random=123)

```

Note that `_MSE_` is an unbiased estimate, so its estimated bias is very small. `_RMSE_` is slightly biased and thus has a larger estimated bias.

REFERENCES:

Dixon

Dixon, P.M. (1993), "The bootstrap and the jackknife: Describing the precision of ecological indices," in Scheiner, S.M. and Gurevitch, J., eds., Design and Analysis of Ecological Experiments, New York: Chapman & Hall, pp 290-318.

E&T

Efron, B. and Tibshirani, R.J. (1993), An Introduction to the Bootstrap, New York: Chapman & Hall.

Hall

Hall, P. (1992), The Bootstrap and Edgeworth Expansion, New York: Springer-Verlag.

Hjorth

Hjorth, J.S.U. (1994), Computer Intensive Statistical Methods, London: Chapman & Hall.

S&T

Shao, J. and Tu, D. (1995), The Jackknife and Bootstrap, New York: Springer-Verlag.

Stine

Stine, R. (1990), "An introduction to bootstrap methods: Examples and ideas," Sociological Methods & Research, 18, 243-291.

These sample files and code examples are provided by SAS Institute Inc. "as is" without warranty of any kind, either express or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. Recipients acknowledge and agree that SAS Institute shall not be liable for any damages whatsoever arising out of their use of this material. In addition, SAS Institute will provide no support for the materials contained herein.

Type: Sample

Date Modified: 2007-10-26 11:29:45

Date Created: 2005-01-13 15:02:40

Product Family	Product	Host	Starting Release	Ending Release
SAS System	SAS/STAT	All	n/a	n/a

FASTats: Frequently Asked-For Statistics

Bookmark this statistics resource so that you'll easily locate it the next time you're in a quandary.

Online at: <http://support.sas.com/kb/30/333.html>.

Events

PharmaSug

<http://www.pharmasug.org>

June 1 – 4
Atlanta

F2008 Conference

<http://www.sas.com/events/fx/>

June 2 – 3
Cary, NC

Discovery 2008

<http://www.sas.com/events/jmp/>

June 16 – 17
Cary, NC

Certification Exam Fair

<http://support.sas.com/certify/promo.html>

July 1 – 31

M2008 – 11th Annual SAS® Data Mining Conference

<http://www.sas.com/events/dmconf/index.html>

October 27 – 28
Las Vegas