bootstrap — Bootstrap sampling and estimation

Description Quick start Menu Syntax Options
Remarks and examples Stored results Methods and formulas References Also see

Description

`bootstrap` performs nonparametric bootstrap estimation of specified statistics (or expressions) for a Stata command or a user-written program. Statistics are bootstrapped by resampling the data in memory with replacement. `bootstrap` is designed for use with nonestimation commands, functions of coefficients, or user-written programs. To bootstrap coefficients, we recommend using the `vce(bootstrap)` option when allowed by the estimation command.

Quick start

Bootstrap the mean of `v1` returned by `summarize` in `r(mean)`

```
bootstrap mean=r(mean): summarize v1
```

Bootstrap the statistic `r(mystat)` returned by program `myprog1`

```
bootstrap stat=r(mystat): myprog1 v1
```

Same as above, but use 100 replications

```
bootstrap stat=r(mystat), reps(100): myprog1 v1
```

Same as above, and save the results from each replication in `mydata.dta`

```
bootstrap stat=r(mystat), reps(100) saving(mydata): myprog1 v1
```

Bootstrap a difference in coefficients estimated by `regress`

```
bootstrap diff=(_b[x2]-_b[x1]): regress y x1 x2 x3
```

Bootstrap the coefficients stored in `e(b)` by `myprog2`

```
bootstrap _b: myprog2 y x1 x2 x3
```

Same as above, but with bootstrap samples taken independently within strata identified by `svar`

```
bootstrap _b, strata(svar): myprog2 y x1 x2 x3
```

Resample clusters defined by `cvar` and create `newcvar` identifying resampled clusters

```
bootstrap _b, cluster(cvar) idcluster(newcvar): myprog2 y x1 x2 x3
```

Menu

Statistics > Resampling > Bootstrap estimation

Syntax

bootstrap *exp_list* [, *options eform_option*] : *command*

options	Description
Main	
reps(#)	perform # bootstrap replications; default is reps(50)
fweights(*varlist*)	perform bootstrap replications using frequency weight variables
iweights(*varlist*)	perform bootstrap replications using importance weight variables
Options	
strata(*varlist*)	specify variables identifying strata
size(#)	draw samples of size #; default is _N
cluster(*varlist*)	specify variables identifying resampling clusters
idcluster(*newvar*)	create new cluster ID variable
saving(*filename*, ...)	save results to *filename*; save statistics in double precision; save results to *filename* every # replications
bca	compute acceleration for BC_a confidence intervals
ties	adjust BC/BCa confidence intervals for ties
mse	use MSE formula for variance estimation
Reporting	
level(#)	set confidence level; default is level(95)
notable	suppress table of results
noheader	suppress table header
nolegend	suppress table legend
verbose	display the full table legend
nodots	suppress replication dots
dots(#)	display dots every # replications
noisily	display any output from *command*
trace	trace *command*
title(*text*)	use *text* as title for bootstrap results
display_options	control columns and column formats, row spacing, line width, display of omitted variables and base and empty cells, and factor-variable labeling
eform_option	display coefficient table in exponentiated form
Advanced	
nodrop	do not drop observations
nowarn	do not warn when e(sample) is not set
force	do not check for *weights* or svy commands; seldom used
reject(*exp*)	identify invalid results
rseed(#)	set random-number seed to #
group(*varname*)	ID variable for groups within cluster()
jackknifeopts(*jkopts*)	options for jackknife; see [R] **jackknife**
coeflegend	display legend instead of statistics

command is any command that follows standard Stata syntax. *weights* are not allowed in *command*.

`collect` and `svy` are allowed; see **[U] 11.1.10 Prefix commands**.

`group()`, `jackknifeopts()`, and `coeflegend` do not appear in the dialog box.

See **[U] 20 Estimation and postestimation commands** for more capabilities of estimation commands.

exp_list contains	(*name*: *elist*)
	elist
	eexp
elist contains	*newvar* = (*exp*)
	(*exp*)
eexp is	*specname*
	[*eqno*]*specname*
specname is	`_b`
	`_b[]`
	`_se`
	`_se[]`
eqno is	##
	name

exp is a standard Stata expression; see **[U] 13 Functions and expressions**.

Distinguish between `[ ]`, which are to be typed, and [ ], which indicate optional arguments.

Options

Main

`reps(`#`)` specifies the number of bootstrap replications to be performed. The default is 50. A total of 50–200 replications are generally adequate for estimates of standard error and thus are adequate for normal-approximation confidence intervals; see Mooney and Duval (1993, 11). Estimates of confidence intervals using the percentile or bias-corrected methods typically require 1,000 or more replications.

`fweights(`*varlist*`)` specifies frequency weight variables used to perform each bootstrap replication. These variables are typically generated by `rwgen bsample`. This option cannot be combined with `reps()`, `strata()`, `size()`, `cluster()`, `idcluster()`, or `iweights()`.

`iweights(`*varlist*`)` specifies importance weight variables used to perform each bootstrap replication. These variables are typically generated by `rwgen bayes`. This option cannot be combined with `reps()`, `strata()`, `size()`, `cluster()`, `idcluster()`, or `fweights()`.

Options

`strata(`*varlist*`)` specifies the variables that identify strata. If this option is specified, bootstrap samples are taken independently within each stratum.

`size(`#`)` specifies the size of the samples to be drawn. The default is `_N`, meaning to draw samples of the same size as the data. If specified, # must be less than or equal to the number of observations within `strata()`.

If `cluster()` is specified, the default size is the number of clusters in the original dataset. For unbalanced clusters, resulting sample sizes will differ from replication to replication. For cluster sampling, # must be less than or equal to the number of clusters within `strata()`.

`cluster(`*varlist*`)` specifies the variables that identify resampling clusters. If this option is specified, the sample drawn during each replication is a bootstrap sample of clusters.

`idcluster(`*newvar*`)` creates a new variable containing a unique identifier for each resampled cluster. This option requires that `cluster()` also be specified.

`saving(`*filename*[, *suboptions*]`)` creates a Stata data file (`.dta` file) consisting of (for each statistic in *exp_list*) a variable containing the replicates.

`double` specifies that the results for each replication be saved as `double`s, meaning 8-byte reals. By default, they are saved as `float`s, meaning 4-byte reals. This option may be used without `saving()` to compute the variance estimates by using double precision.

`every(#)` specifies that results be written to disk every #th replication. `every()` should be specified only in conjunction with `saving()` when *command* takes a long time for each replication. This option will allow recovery of partial results should some other software crash your computer. See [P] **postfile**.

`replace` specifies that *filename* be overwritten if it exists. This option does not appear in the dialog box.

`bca` specifies that `bootstrap` estimate the acceleration of each statistic in *exp_list*. This estimate is used to construct BC_a confidence intervals. Type `estat bootstrap, bca` to display the BC_a confidence interval generated by the `bootstrap` command.

`ties` specifies that `bootstrap` adjust for ties in the replicate values when computing the median bias used to construct BC and BCa confidence intervals.

`mse` specifies that `bootstrap` compute the variance by using deviations of the replicates from the observed value of the statistics based on the entire dataset. By default, `bootstrap` computes the variance by using deviations from the average of the replicates.

Reporting

`level(#)`; see [R] **Estimation options**.

`notable` suppresses the display of the table of results.

`noheader` suppresses the display of the table header. This option implies `nolegend`. This option may also be specified when replaying estimation results.

`nolegend` suppresses the display of the table legend. This option may also be specified when replaying estimation results.

`verbose` specifies that the full table legend be displayed. By default, coefficients and standard errors are not displayed. This option may also be specified when replaying estimation results.

`nodots` and `dots(#)` specify whether to display replication dots. By default, one dot character is displayed for each successful replication. An “x” is displayed if *command* returns an error or if any value in *exp_list* is missing. You can also control whether dots are displayed using `set dots`; see [R] **set**.

`nodots` suppresses display of the replication dots.

`dots(#)` displays dots every # replications. `dots(0)` is a synonym for `nodots`.

`noisily` specifies that any output from *command* be displayed. This option implies the `nodots` option.

`trace` causes a trace of the execution of *command* to be displayed. This option implies the `noisily` option.

`title(`*text*`)` specifies a title to be displayed above the table of bootstrap results. The default title is the title stored in `e(title)` by an estimation command, or if `e(title)` is not filled in, `Bootstrap results` is used. `title()` may also be specified when replaying estimation results.

display_options: `noci`, `nopvalues`, `noomitted`, `vsquish`, `noemptycells`, `baselevels`, `allbaselevels`, `nofvlabel`, `fvwrap(#)`, `fvwrapon(`*style*`)`, `cformat(`%*fmt*`)`, `pformat(`%*fmt*`)`, `sformat(`%*fmt*`)`, and `nolstretch`; see [R] **Estimation options**.

eform_option causes the coefficient table to be displayed in exponentiated form; see [R] ***eform_option***.

command determines which of the following are allowed (`eform(`*string*`)` and `eform` are always allowed):

eform_option	Description
`eform(`*string*`)`	use *string* for the column title
`eform`	exponentiated coefficient, *string* is `exp(b)`
`hr`	hazard ratio, *string* is `Haz. ratio`
`shr`	subhazard ratio, *string* is `SHR`
`irr`	incidence-rate ratio, *string* is `IRR`
`or`	odds ratio, *string* is `Odds ratio`
`rrr`	relative-risk ratio, *string* is `RRR`

Advanced

`nodrop` prevents observations outside `e(sample)` and the `if` and `in` qualifiers from being dropped before the data are resampled.

`nowarn` suppresses the display of a warning message when *command* does not set `e(sample)`.

`force` suppresses the restriction that *command* not specify weights or be a `svy` command. This is a rarely used option. Use it only if you know what you are doing.

`reject(`*exp*`)` identifies an expression that indicates when results should be rejected. When *exp* is true, the resulting values are reset to missing values.

`rseed(#)` sets the random-number seed. Specifying this option is equivalent to typing the following command prior to calling `bootstrap`:

```
. set seed #
```

When either `fweights()` or `iweights()` is specified, `rseed()` is ignored.

The following options are available with `bootstrap` but are not shown in the dialog box:

`group(`*varname*`)` re-creates *varname* containing a unique identifier for each group across the resampled clusters. This option requires that `idcluster()` also be specified.

This option is useful for maintaining unique group identifiers when sampling clusters with replacement. Suppose that cluster 1 contains 3 groups. If the `idcluster(newclid)` option is specified and cluster 1 is sampled multiple times, `newclid` uniquely identifies each copy of cluster 1. If `group(newgroupid)` is also specified, `newgroupid` uniquely identifies each copy of each group.

`jackknifeopts(`*jkopts*`)` identifies options that are to be passed to `jackknife` when it computes the acceleration values for the BC_a confidence intervals; see [R] **jackknife**. This option requires the `bca` option and is mostly used for passing the `eclass`, `rclass`, or `n(#)` option to `jackknife`.

`coeflegend`; see [R] **Estimation options**.

Remarks and examples

Remarks are presented under the following headings:

Introduction

With few assumptions, bootstrapping provides a way of estimating standard errors and other measures of statistical precision (Efron 1979; Efron and Stein 1981; Efron 1982; Efron and Tibshirani 1986; Efron and Tibshirani 1993; also see Davison and Hinkley [1997]; Guan [2003]; Mooney and Duval [1993]; Poi [2004]; and Stine [1990]). It provides a way to obtain such measures when no formula is otherwise available or when available formulas make inappropriate assumptions. Cameron and Trivedi (2022, chap. 12) discuss many bootstrapping topics and demonstrate how to do them in Stata.

To illustrate bootstrapping, suppose that you have a dataset containing N observations and an estimator that, when applied to the data, produces certain statistics. You draw, with replacement, N observations from the N-observation dataset. In this random drawing, some of the original observations will appear once, some more than once, and some not at all. Using the resampled dataset, you apply the estimator and collect the statistics. This process is repeated many times; each time, a new random sample is drawn and the statistics are recalculated.

This process builds a dataset of replicated statistics. From these data, you can calculate the standard error by using the standard formula for the sample standard deviation

$$\widehat{\mathrm{se}} = \left\{ \frac{1}{k-1} \sum (\widehat{\theta}_i - \overline{\theta})^2 \right\}^{1/2}$$

where $\widehat{\theta}_i$ is the statistic calculated using the ith bootstrap sample and k is the number of replications. This formula gives an estimate of the standard error of the statistic, according to Hall and Wilson (1991). Although the average, $\overline{\theta}$, of the bootstrapped estimates is used in calculating the standard deviation, it is not used as the estimated value of the statistic itself. Instead, the original observed value of the statistic, $\widehat{\theta}$, is used, meaning the value of the statistic computed using the original N observations.

You might think that $\overline{\theta}$ is a better estimate of the parameter than $\widehat{\theta}$, but it is not. If the statistic is biased, bootstrapping exaggerates the bias. In fact, the bias can be estimated as $\overline{\theta} - \widehat{\theta}$ (Efron 1982, 33). Knowing this, you might be tempted to subtract this estimate of bias from $\widehat{\theta}$ to produce an unbiased statistic. The bootstrap bias estimate has an indeterminate amount of random error, so this unbiased estimator may have greater mean squared error than the biased estimator (Mooney and Duval 1993; Hinkley 1978). Thus, $\widehat{\theta}$ is the best point estimate of the statistic.

The logic behind the bootstrap is that all measures of precision come from a statistic's sampling distribution. When the statistic is estimated on a sample of size N from some population, the sampling distribution tells you the relative frequencies of the values of the statistic. The sampling distribution, in turn, is determined by the distribution of the population and the formula used to estimate the statistic.

Sometimes the sampling distribution can be derived analytically. For instance, if the underlying population is distributed normally and you calculate means, the sampling distribution for the mean is also normal but has a smaller variance than that of the population. In other cases, deriving the sampling distribution is difficult, such as when means are calculated from nonnormal populations. Sometimes, as in the case of means, it is not too difficult to derive the sampling distribution as the sample size goes to infinity ($N \to \infty$). However, such asymptotic distributions may not perform well when applied to finite samples.

If you knew the population distribution, you could obtain the sampling distribution by simulation: you could draw random samples of size N, calculate the statistic, and make a tally. Bootstrapping does precisely this, but it uses the observed distribution of the sample in place of the true population distribution. Thus, the bootstrap procedure hinges on the assumption that the observed distribution is a good estimate of the underlying population distribution. In return, the bootstrap produces an estimate, called the bootstrap distribution, of the sampling distribution. From this, you can estimate the standard error of the statistic, produce confidence intervals, etc.

The accuracy with which the bootstrap distribution estimates the sampling distribution depends on the number of observations in the original sample and the number of replications in the bootstrap. A crudely estimated sampling distribution is adequate if you are going to extract, say, only a standard error. A better estimate is needed if you want to use the 2.5th and 97.5th percentiles of the distribution to produce a 95% confidence interval. To extract many features simultaneously about the distribution, an even better estimate is needed. Generally, replications on the order of 1,000 produce very good estimates, but only 50–200 replications are needed for estimates of standard errors. See Poi (2004) for a method to choose the number of bootstrap replications.

Using bootstrap

Typing

```
. bootstrap exp_list, reps(#): command
```

executes *command* multiple times, bootstrapping the statistics in *exp_list* by resampling observations (with replacement) from the data in memory # times. This method is commonly referred to as the nonparametric bootstrap.

command defines the statistical command to be executed. Most Stata commands and user-written programs can be used with `bootstrap`, as long as they follow standard Stata syntax; see **[U] 11 Language syntax**. If the `bca` option is supplied, *command* must also work with `jackknife`; see [R] **jackknife**. The `by` prefix may not be part of *command*.

exp_list specifies the statistics to be collected from the execution of *command*. If *command* changes the contents in `e(b)`, *exp_list* is optional and defaults to `_b`.

Because bootstrapping is a random process, if you want to be able to reproduce results, set the random-number seed by specifying the `rseed(#)` option or by typing

```
. set seed #
```

where # is a seed of your choosing, before running `bootstrap`; see [R] **set seed**.

Another way to reproduce results is to use frequency weights (`fweights()`) or importance weights (`iweights()`). The weight variables specified in these options are typically generated using either the `rwgen bsample` or `rwgen bayes` command. When either `fweights()` or `iweights()` is specified, `rseed()` is ignored.

Regression coefficients

▷ Example 1

Let's say that we wish to compute bootstrap estimates for the standard errors of the coefficients from the following regression:

```
. use https://www.stata-press.com/data/r19/auto
(1978 automobile data)

. regress mpg weight gear foreign

      Source |       SS           df       MS      Number of obs   =        74
-------------+----------------------------------   F(3, 70)        =     46.73
       Model |  1629.67805         3  543.226016   Prob > F        =    0.0000
    Residual |  813.781411        70  11.6254487   R-squared       =    0.6670
-------------+----------------------------------   Adj R-squared   =    0.6527
       Total |  2443.45946        73  33.4720474   Root MSE        =    3.4096

------------------------------------------------------------------------------
         mpg | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
      weight |   -.006139   .0007949    -7.72   0.000    -.0077245   -.0045536
  gear_ratio |   1.457113   1.541286     0.95   0.348    -1.616884     4.53111
     foreign |  -2.221682   1.234961    -1.80   0.076    -4.684735    .2413715
       _cons |   36.10135   6.285984     5.74   0.000     23.56435    48.63835
------------------------------------------------------------------------------
```

To run the bootstrap, we simply prefix the above regression command with the `bootstrap` command (specifying its options before the colon separator). We must set the random-number seed before calling `bootstrap`.

```
. bootstrap, reps(100) rseed(1): regress mpg weight gear foreign
(running regress on estimation sample)

Bootstrap replications (100): .........10.........20.........30.........40......
> ...50.........60.........70.........80.........90.........100 done

Linear regression                                       Number of obs =     74
                                                        Replications  =    100
                                                        Wald chi2(3)  = 167.13
                                                        Prob > chi2   = 0.0000
                                                        R-squared     = 0.6670
                                                        Adj R-squared = 0.6527
                                                        Root MSE      = 3.4096

------------------------------------------------------------------------------
             |   Observed   Bootstrap                         Normal-based
         mpg | coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
      weight |   -.006139   .0006063   -10.13   0.000    -.0073273   -.0049507
  gear_ratio |   1.457113   1.367917     1.07   0.287    -1.223954    4.138181
     foreign |  -2.221682   1.169727    -1.90   0.058    -4.514305     .070942
       _cons |   36.10135    5.20581     6.93   0.000     25.89815    46.30455
------------------------------------------------------------------------------
```

The displayed confidence interval is based on the assumption that the sampling (and hence bootstrap) distribution is approximately normal (see *Methods and formulas* below). Because this confidence interval is based on the standard error, it is a reasonable estimate if normality is approximately true, even for a few replications. Other types of confidence intervals are available after `bootstrap`; see [R] **bootstrap postestimation**.

We could instead supply names to our expressions when we run `bootstrap`. For example,

```
. bootstrap diff=(_b[weight]-_b[gear]): regress mpg weight gear foreign
```

would bootstrap a statistic, named `diff`, equal to the difference between the coefficients on `weight` and `gear_ratio`.

◁

❑ Technical note

`regress`, like many estimation commands, allows the `vce(bootstrap)` option. For any estimation command that allows this option, we recommend using `vce(bootstrap)` over `bootstrap` because the estimation command automatically handles clustering and other model-specific details for you.

❑

Expressions

▷ Example 2

When we use `bootstrap`, the list of statistics can contain complex expressions, as long as each expression is enclosed in parentheses. For example, to bootstrap the range of a variable `x`, we could type

```
. bootstrap range=(r(max)-r(min)), reps(1000): summarize x
```

Of course, we could also bootstrap the minimum and maximum and later compute the range.

```
. bootstrap max=r(max) min=r(min), reps(1000) saving(mybs): summarize x

. use mybs, clear
(bootstrap: summarize)

. generate range = max - min

. bstat range, stat(19.5637501)
```

The difference between the maximum and minimum of `x` in the sample is 19.5637501.

The `stat()` option to `bstat` specifies the observed value of the statistic (`range`) to be summarized. This option is useful when, as shown above, the statistic of ultimate interest is not specified directly to `bootstrap` but instead is calculated by other means.

Here the observed values of `r(max)` and `r(min)` are stored as characteristics of the dataset created by `bootstrap` and are thus available for retrieval by `bstat`; see [R] **bstat**. The observed range, however, is unknown to `bstat`, so it must be specified.

◁

Combining bootstrap datasets

You can combine two datasets from separate runs of `bootstrap` by using append (see [D] **append**) and then get the bootstrap statistics for the combined datasets by running `bstat`. The runs must have been performed independently (having different starting random-number seeds), and the original dataset, command, and bootstrap statistics must have been all the same.

A note about macros

In example 2, we executed the command

```
. bootstrap max=r(max) min=r(min), reps(1000) saving(mybs): summarize x
```

We did not enclose `r(max)` and `r(min)` in single quotes, as we would in most other contexts, because it would not produce what was intended:

```
. bootstrap ‘r(max)’ ‘r(min)’, reps(1000) saving(mybs): summarize x
```

To understand why, note that `‘r(max)’`, like any reference to a local macro, will evaluate to a literal string containing the contents of `r(max)` *before* `bootstrap` is even executed. Typing the command above would appear to Stata as if we had typed

```
. bootstrap 14.5441234 33.4393293, reps(1000) saving(mybs): summarize x
```

Even worse, the current contents of `r(min)` and `r(max)` could be empty, producing an even more confusing result. To avoid this outcome, refer to statistics by name (for example, `r(max)`) and not by value (for example, `‘r(max)’`).

Achieved significance level

▷ Example 3

Suppose that we wish to estimate the *achieved significance level* (ASL) of a test statistic by using the bootstrap. ASL is another name for p-value. An example is

$$\mathrm{ASL} = \Pr\left(\widehat{\theta}^* \ge \widehat{\theta} | H_0\right)$$

for an upper-tailed, alternative hypothesis, where H_0 denotes the null hypothesis, $\widehat{\theta}$ is the observed value of the test statistic, and $\widehat{\theta}^*$ is the random variable corresponding to the test statistic, assuming that H_0 is true.

Here we will compare the mean miles per gallon (`mpg`) between foreign and domestic cars by using the two-sample t test with unequal variances. The following results indicate the p-value to be `0.0034` for the two-sided test using Satterthwaite’s approximation. Thus, assuming that mean `mpg` is the same for foreign and domestic cars, we would expect to observe a t statistic more extreme (in absolute value) than `3.1797` in about 0.3% of all possible samples of the type that we observed. Thus, we have evidence to reject the null hypothesis that the means are equal.

```
. use https://www.stata-press.com/data/r19/auto
(1978 automobile data)

. ttest mpg, by(foreign) unequal

Two-sample t test with unequal variances
------------------------------------------------------------------------------
   Group |     Obs        Mean    Std. err.   Std. dev.   [95% conf. interval]
---------+--------------------------------------------------------------------
Domestic |      52    19.82692     .657777    4.743297    18.50638    21.14747
 Foreign |      22    24.77273     1.40951    6.611187    21.84149    27.70396
---------+--------------------------------------------------------------------
Combined |      74     21.2973    .6725511    5.785503     19.9569    22.63769
---------+--------------------------------------------------------------------
    diff |           -4.945804    1.555438               -8.120053   -1.771556
------------------------------------------------------------------------------
    diff = mean(Domestic) - mean(Foreign)                         t =  -3.1797
H0: diff = 0                     Satterthwaite's degrees of freedom =  30.5463

    Ha: diff < 0                 Ha: diff != 0                 Ha: diff > 0
 Pr(T < t) = 0.0017         Pr(|T| > |t|) = 0.0034          Pr(T > t) = 0.9983
```

We also place the value of the test statistic in a scalar for later use.

```
. scalar tobs = r(t)
```

Efron and Tibshirani (1993, 224) describe an alternative to Satterthwaite's approximation that estimates the ASL by bootstrapping the statistic from the test of equal means. Their idea is to recenter the two samples to the combined sample mean so that the data now conform to the null hypothesis but that the variances within the samples remain unchanged.

```
. summarize mpg, meanonly

. scalar omean = r(mean)

. summarize mpg if foreign==0, meanonly

. replace mpg = mpg - r(mean) + scalar(omean) if foreign==0
variable mpg was int now float
(52 real changes made)

. summarize mpg if foreign==1, meanonly

. replace mpg = mpg - r(mean) + scalar(omean) if foreign==1
(22 real changes made)

. sort foreign

. by foreign: summarize mpg

--------------------------------------------------------------------------------
-> foreign = Domestic

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
         mpg |         52     21.2973    4.743297   13.47037   35.47038

--------------------------------------------------------------------------------
-> foreign = Foreign

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
         mpg |         22     21.2973    6.611187   10.52457   37.52457
```

Each sample (foreign and domestic) is a stratum, so the bootstrapped samples must have the same number of foreign and domestic cars as the original dataset. This requirement is facilitated by the `strata()` option to `bootstrap`. By typing the following, we bootstrap the test statistic using the modified dataset and save the values in `bsauto2.dta`:

```
. keep mpg foreign

. set seed 1

. bootstrap t=r(t), rep(1000) strata(foreign) saving(bsauto2) nodots: ttest mpg,
> by(foreign) unequal

warning: ttest does not set e(sample), so no observations will be excluded
         from the resampling because of missing values or other reasons. To
         exclude observations, press Break, save the data, drop any
         observations that are to be excluded, and rerun bootstrap.

Bootstrap results

Number of strata = 2                                   Number of obs =    74
                                                       Replications  = 1,000

      Command: ttest mpg, by(foreign) unequal
            t: r(t)

──────────────┬────────────────────────────────────────────────────────────────
              │   Observed   Bootstrap                         Normal-based
              │ coefficient  std. err.      z    P>|z|     [95% conf. interval]
──────────────┼────────────────────────────────────────────────────────────────
            t │   1.75e-07   1.051867     0.00   1.000    -2.061622    2.061622
──────────────┴────────────────────────────────────────────────────────────────
```

We can use the data in `bsauto2.dta` to estimate ASL via the fraction of bootstrap test statistics that are more extreme than `3.1797`.

```
. use bsauto2, clear
(bootstrap: ttest)

. generate indicator = abs(t)>=abs(scalar(tobs))

. summarize indicator, meanonly

. display "ASLboot = " r(mean)
ASLboot = .004
```

The result is $\mathrm{ASL}_{\mathrm{boot}} = 0.004$. Assuming that the mean `mpg` is the same between foreign and domestic cars, we would expect to observe a t statistic more extreme (in absolute value) than `3.1797` in about 0.4% of all possible samples of the type we observed. This finding is still strong evidence to reject the hypothesis that the means are equal.

◁

Bootstrapping a ratio

▷ Example 4

Suppose that we wish to produce a bootstrap estimate of the ratio of two means. Because `summarize` stores results for only one variable, we must call `summarize` twice to compute the means. Actually, we could use `collapse` to compute the means in one call, but calling `summarize` twice is much faster. Thus, we will have to write a small program that will return the results we want.

We write the program below and save it to a file called `ratio.ado` (see **[U] 17 Ado-files**). Our program takes two variable names as input and saves them in the `local` macros `y` (first variable) and `x` (second variable). It then computes one statistic: the mean of `‘y’` divided by the mean of `‘x’`. This value is returned as a scalar in `r(ratio)`. `ratio` also returns the ratio of the number of observations used to the mean for each variable.

```
program myratio, rclass
        version 19.5      // (or version 19 if you do not have StataNow)
        args y x
        confirm var ‘y’
        confirm var ‘x’
        tempname ymean
        summarize ‘y’, meanonly
        scalar ‘ymean’ = r(mean)
        return scalar n_‘y’ = r(N)
        summarize ‘x’, meanonly
        return scalar n_‘x’ = r(N)
        return scalar ratio = ‘ymean’/r(mean)
end
```

Remember to test any newly written commands before using them with `bootstrap`.

```
. use https://www.stata-press.com/data/r19/auto, clear
(1978 automobile data)

. summarize price

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
       price |         74    6165.257    2949.496       3291      15906

. scalar mean1=r(mean)

. summarize weight

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
      weight |         74    3019.459    777.1936       1760       4840

. scalar mean2=r(mean)

. di scalar(mean1)/scalar(mean2)
2.0418412

. myratio price weight

. return list

scalars:
              r(ratio) =  2.041841210168278
           r(n_weight) =  74
            r(n_price) =  74
```

The results of running `bootstrap` on our program are

```
. use https://www.stata-press.com/data/r19/auto
(1978 automobile data)

. set seed 1

. bootstrap ratio=r(ratio), reps(1000) nowarn nodots: myratio price weight

Bootstrap results                                       Number of obs =    74
                                                        Replications  = 1,000

      Command: myratio price weight
        ratio: r(ratio)

------------------------------------------------------------------------------
             |   Observed   Bootstrap                         Normal-based
             | coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
       ratio |   2.041841   .0953559    21.41   0.000     1.854947    2.228735
------------------------------------------------------------------------------
```

As mentioned previously, we should specify the `saving()` option if we wish to save the bootstrap dataset.

◁

Warning messages and e(sample)

`bootstrap` works well with weights specified in either `fweights()` or `iweights()`. When neither of these options is specified, `bootstrap` determines the presence of weights by parsing the prefixed command with standard syntax. However, commands like `stcox` and `streg` require that weights be specified in `stset`, and some user commands may allow weights to be specified by using an option instead of the standard syntax. Both cases pose a problem for `bootstrap` because it cannot determine the presence of weights under these circumstances. In these cases, we can only assume that you know what you are doing.

`bootstrap` does not know which variables of the dataset in memory matter to the calculation at hand. You can speed their execution by dropping unnecessary variables because, otherwise, they are included in each bootstrap sample.

You should thus drop observations with missing values. Leaving in missing values causes no problem in one sense because all Stata commands deal with missing values gracefully. It does, however, cause a statistical problem. Bootstrap sampling is defined as drawing, with replacement, samples of size N from a set of N observations. `bootstrap` determines N by counting the number of observations in memory, not counting the number of nonmissing values on the relevant variables. The result is that too many observations are resampled; the resulting bootstrap samples, because they are drawn from a population with missing values, are of unequal sizes.

If the number of missing values relative to the sample size is small, this will make little difference. If you have many missing values, however, you should first drop the observations that contain them.

▷ Example 5

To illustrate, we use the previous example but replace some of the values of `price` with missing values. The number of values of `price` used to compute the mean for each bootstrap is not constant. This is the purpose of the warning message.

```
. use https://www.stata-press.com/data/r19/auto
(1978 automobile data)

. replace price = . if inlist(_n,1,3,5,7)
(4 real changes made, 4 to missing)

. set seed 1

. bootstrap ratio=r(ratio) np=r(n_price) nw=r(n_weight), reps(100) nodots:
> myratio price weight

warning: myratio does not set e(sample), so no observations will be excluded
         from the resampling because of missing values or other reasons. To
         exclude observations, press Break, save the data, drop any
         observations that are to be excluded, and rerun bootstrap.

Bootstrap results                                       Number of obs =  74
                                                        Replications  = 100

      Command: myratio price weight
        ratio: r(ratio)
           np: r(n_price)
           nw: r(n_weight)

------------------------------------------------------------------------------
             |   Observed   Bootstrap                         Normal-based
             | coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
       ratio |   2.063051   .0981706    21.01   0.000     1.870641    2.255462
          np |         70   2.071939    33.78   0.000     65.93908    74.06092
          nw |         74          .        .       .            .           .
------------------------------------------------------------------------------
```

◁

Bootstrapping statistics from data with a complex structure

Here we describe how to bootstrap statistics from data with a complex structure, for example, longitudinal or panel data, or matched data. `bootstrap`, however, is not designed to work with complex survey data. It is important to include all necessary information about the structure of the data in the `bootstrap` syntax to obtain correct bootstrap estimates for standard errors and confidence intervals.

`bootstrap` offers several options identifying the specifics of the data. These options are `strata()`, `cluster()`, `idcluster()`, and `group()`. The usage of `strata()` was described in example 3 above. Below, we demonstrate several examples that require specifying the other three options.

▷ Example 6

Suppose that `auto.dta` in example 1 above are clustered by `rep78`. We want to obtain bootstrap estimates for the standard errors of the difference between the coefficients on `weight` and `gear_ratio`, taking into account clustering.

We supply the `cluster(rep78)` option to `bootstrap` to request resampling from clusters rather than from observations in the dataset.

```
. use https://www.stata-press.com/data/r19/auto, clear
(1978 automobile data)

. keep if rep78<.
(5 observations deleted)

. bootstrap diff=(_b[weight]-_b[gear]), rseed(1) cluster(rep78): regress mpg
> weight gear foreign
(running regress on estimation sample)

Bootstrap replications (50): .........10.........20.........30.........40.......
> ..50 done

Linear regression                                       Number of obs = 69
                                                        Replications  = 50

      Command: regress mpg weight gear foreign
         diff: _b[weight]-_b[gear]

                                    (Replications based on 5 clusters in rep78)
------------------------------------------------------------------------------
             |   Observed   Bootstrap                         Normal-based
             | coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
        diff |   -1.910396   2.538558    -0.75   0.452    -6.885879    3.065087
------------------------------------------------------------------------------
```

We drop missing values in `rep78` before issuing the command because `bootstrap` does not allow missing values in `cluster()`. See the section above about using `bootstrap` when variables contain missing values.

We can also obtain these same results by using the following syntax:

```
. bootstrap diff=(_b[weight]-_b[gear]), rseed(1): regress mpg weight gear foreign,
> vce(cluster rep78)
```

When only clustered information is provided to the command, `bootstrap` can pick up the `vce(cluster` *clustvar*`)` option from the main command and use it to resample from clusters.

◁

▷ Example 7

Suppose now that we have matched data and want to use `bootstrap` to obtain estimates of the standard errors of the exponentiated difference between two coefficients (or, equivalently, the ratio of two odds ratios) estimated by `clogit`. Consider the example of matched case–control data on birthweight of infants described in example 2 of [R] **clogit**.

The infants are paired by being matched on mother's age. All groups, defined by the `pairid` variable, have 1:1 matching. `clogit` requires that the matching information, `pairid`, be supplied to the `group()` (or, equivalently, `strata()`) option to be used in computing the parameter estimates. Because the data are matched, we need to resample from groups rather than from the whole dataset. However, simply supplying the grouping variable `pairid` in `cluster()` is not enough with `bootstrap`, as it is with clustered data.

```
. use https://www.stata-press.com/data/r19/lowbirth2, clear
(Applied Logistic Regression, Hosmer & Lemeshow)

. bootstrap ratio=exp(_b[smoke]-_b[ptd]), rseed(1) cluster(pairid):
> clogit low lwt smoke ptd ht ui i.race, group(pairid)
(running clogit on estimation sample)

Bootstrap replications (50): .........10.........20.........30.........40.......
> ..50 done

Bootstrap results                                       Number of obs = 112
                                                        Replications  =  50

      Command: clogit low lwt smoke ptd ht ui i.race, group(pairid)
        ratio: exp(_b[smoke]-_b[ptd])

                                   (Replications based on 56 clusters in pairid)
------------------------------------------------------------------------------
             |   Observed   Bootstrap                         Normal-based
             | coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
       ratio |   .6654095   2.043274     0.33   0.745    -3.339334    4.670153
------------------------------------------------------------------------------
```

For the syntax above, imagine that the first pair was sampled twice during a replication. Then, the bootstrap sample has four subjects with `pairid` equal to one, which clearly violates the original 1:1 matching design. As a result, the estimates of the coefficients obtained from this bootstrap sample will be incorrect.

Therefore, in addition to resampling from groups, we need to ensure that resampled groups are uniquely identified in each of the bootstrap samples. The `idcluster(`*newcluster*`)` option is designed for this. It requests that at each replication `bootstrap` create the new variable, *newcluster*, containing unique identifiers for all resampled groups. Thus, to make sure that the correct matching is preserved during each replication, we need to specify the grouping variable in `cluster()`, supply a variable name to `idcluster()`, and use this variable as the grouping variable with `clogit`, as we demonstrate below.

```
. bootstrap ratio=exp(_b[smoke]-_b[ptd]), rseed(1) cluster(pairid)
> idcluster(newpairid): clogit low lwt smoke ptd ht ui i.race, group(newpairid)
(running clogit on estimation sample)

Bootstrap replications (50): .........10.........20.........30.........40.......
> ..50 done

Bootstrap results                                       Number of obs = 112
                                                        Replications  =  50

      Command: clogit low lwt smoke ptd ht ui i.race, group(newpairid)
        ratio: exp(_b[smoke]-_b[ptd])

                                   (Replications based on 56 clusters in pairid)
------------------------------------------------------------------------------
             |   Observed   Bootstrap                         Normal-based
             | coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
       ratio |   .6654095   1.156848     0.58   0.565    -1.601972    2.932791
------------------------------------------------------------------------------
```

Note the difference between the estimates of the bootstrap standard error for the two specifications of the `bootstrap` syntax.

◁

❑ Technical note

Similarly, when you have panel (longitudinal) data, all resampled panels must be unique in each of the bootstrap samples to obtain correct bootstrap estimates of statistics. Therefore, both `cluster(`*panelvar*`)` and `idcluster(`*newpanelvar*`)` must be specified with `bootstrap`, and `i(`*newpanelvar*`)` must be used with the main command. Moreover, you must clear the current `xtset` settings by typing `xtset, clear` before calling `bootstrap`.

❑

▷ Example 8

Continuing with our birthweight data, suppose that we have more information about doctors supervising women's pregnancies. We believe that the data on the pairs of infants from the same doctor may be correlated and want to adjust standard errors for possible correlation among the pairs. `clogit` offers the `vce(cluster` *clustvar*`)` option to do this.

Let's add a cluster variable to our dataset. One thing to keep in mind is that to use `vce(cluster` *clustvar*`)`, groups in `group()` must be nested within clusters.

```
. use https://www.stata-press.com/data/r19/lowbirth2, clear
(Applied Logistic Regression, Hosmer & Lemeshow)

. set seed 12345

. by pairid, sort: egen byte doctor = total(int(2*runiform()+1)*(_n == 1))

. clogit low lwt smoke ptd ht ui i.race, group(pairid) vce(cluster doctor)

Iteration 0:  Log pseudolikelihood = -26.768693
Iteration 1:  Log pseudolikelihood = -25.810476
Iteration 2:  Log pseudolikelihood = -25.794296
Iteration 3:  Log pseudolikelihood = -25.794271
Iteration 4:  Log pseudolikelihood = -25.794271

Conditional (fixed-effects) logistic regression         Number of obs =    112
                                                        Wald chi2(1)  =      .
                                                        Prob > chi2   =      .
Log pseudolikelihood = -25.794271                       Pseudo R2     = 0.3355

                                 (Std. err. adjusted for 2 clusters in doctor)
------------------------------------------------------------------------------
             |               Robust
         low | Coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
         lwt |  -.0183757   .0020314    -9.05   0.000    -.0223571   -.0143942
       smoke |   1.400656   .3067525     4.57   0.000     .7994322     2.00188
         ptd |   1.808009   .2092246     8.64   0.000     1.397936    2.218082
          ht |   2.361152   1.410341     1.67   0.094    -.4030665     5.12537
          ui |   1.401929   .9406248     1.49   0.136    -.4416617     3.24552
             |
        race |
      Black  |   .5713643   .9992656     0.57   0.567     -1.38716    2.529889
      Other  |  -.0253148   .8453206    -0.03   0.976    -1.682113    1.631483
------------------------------------------------------------------------------
```

To obtain correct bootstrap standard errors of the exponentiated difference between the two coefficients in this example, we need to make sure that both resampled clusters and groups within resampled clusters are unique in each of the bootstrap samples. To achieve this, `bootstrap` needs the information about clusters in `cluster()`, the variable name of the new identifier for clusters in `idcluster()`, and the information about groups in `group()`. We demonstrate the corresponding syntax of `bootstrap` below.

```
. bootstrap ratio=exp(_b[smoke]-_b[ptd]), rseed(1) cluster(doctor)
> idcluster(uidoctor) group(pairid): clogit low lwt smoke ptd ht ui i.race,
> group(pairid)
(running clogit on estimation sample)

Bootstrap replications (50): .........10.........20.........30.........40.......
> ..50 done

Bootstrap results                                       Number of obs = 112
                                                        Replications  =  50

      Command: clogit low lwt smoke ptd ht ui i.race, group(pairid)
        ratio: exp(_b[smoke]-_b[ptd])

                                   (Replications based on 2 clusters in doctor)
------------------------------------------------------------------------------
             |   Observed   Bootstrap                         Normal-based
             | coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
       ratio |   .6654095   .1459234     4.56   0.000     .3794048    .9514142
------------------------------------------------------------------------------
```

In the above syntax, although we specify `group(pairid)` with `clogit`, it is not the group identifiers of the original `pairid` variable that are used to compute parameter estimates from bootstrap samples. The way `bootstrap` works is that, at each replication, the clusters defined by `doctor` are resampled and the new variable, `uidoctor`, uniquely identifying resampled clusters is created. After that, another new variable uniquely identifying the (`uidoctor`, `group`) combination is created and renamed to have the same name as the grouping variable, `pairid`. This newly defined grouping variable is then used by `clogit` to obtain the parameter estimates from this bootstrap sample of clusters. After all replications are performed, the original values of the grouping variable are restored.

◁

❑ Technical note

The same logic must be used when running `bootstrap` with commands designed for panel (longitudinal) data that allow specifying the `cluster(`*clustervar*`)` option. To ensure that the combination of (*clustervar*, *panelvar*) values are unique in each of the bootstrap samples, `cluster(`*clustervar*`)`, `idcluster(`*newclustervar*`)`, and `group(`*panelvar*`)` must be specified with `bootstrap`, and `i(`*panelvar*`)` must be used with the main command.

❑

❑ Technical note

When the `bootstrap` prefix is used with a user-defined program and when the expression list is `_b`, `bootstrap` calls

```
set coeftabresults off
```

before entering the replication loop to prevent Stata from performing unnecessary calculations. This means that, provided option `noisily` is not specified, estimation commands will not build or post the coefficient table matrix `r(table)`.

If your program calls an estimation command and needs `r(table)` to exist to perform properly, then your program will need to call

```
set coeftabresults on
```

before calling other estimation commands.

❑

Bootstrapping statistics using frequency or importance weights

We can use frequency weights or importance weights when bootstrapping statistics by specifying the `fweights()` or `iweights()` option with the `bootstrap` prefix.

If your data have a complex structure, for example, if they are clustered or stratified, you can use the `cluster()`, `idcluster()`, and `strata()` options with `rwgen bsample` to generate replicate weights that reflect the complexities of your data. Then, when you specify these replicate weight variables in the `fweights()` option of the `bootstrap` prefix, `bootstrap` will use this information about the structure of your data to produce correct bootstrap estimates.

Clustered and matched case–control data are supported only when using the `fweights()` option of the `bootstrap` prefix; these data structures are not supported when using the `iweights()` option.

When you specify the `fweights()` option, the `bootstrap` prefix calculates the sum of the frequencies for each replication and compares it with the number of observations in the original data used to obtain point estimates. If the numbers do not match, a missing value of the estimate is posted for that replication. To include the results for those replications in the final bootstrap confidence interval calculation, you can specify the `force` option. However, you should use this option only if you understand the implications.

Additionally, suppose you have generated frequency weight variables with `rwgen bsample` by specifying the `cluster()`, `idcluster()`, or `strata()` option. When you specify these variables in the `fweights()` option of the `bootstrap` prefix, `bootstrap` will check whether the number of clusters is consistent with the original dataset. You can ignore such checks by specifying the `force` option.

If the `iweights()` option is specified, the `bootstrap` prefix verifies that the number of missing values is the same as in the original dataset used to obtain point estimates. You can ignore such checks by specifying the `force` option.

In this section, we demonstrate how to generate replicate weights with `rwgen bsample` and how to use them when bootstrapping statistics. For more examples, see [R] **rwgen**. Also see [R] **bayesboot** for examples of Bayesian bootstrap.

▷ Example 9

Consider a variation of the auto dataset that contains frequency weights and importance weights. In `wauto.dta`, we have variables `f1-f100`, which are replicate weights generated by random sampling with replacement. In practice, you might obtain these replicate weights from the data source or generate them yourself with the `rwgen bsample` command.

We wish to compute bootstrap estimates for the standard errors of the coefficients from a linear regression using frequency weights. Therefore, we specify the frequency weight variables in the `fweights()` option:

```
. use https://www.stata-press.com/data/r19/wauto, clear
(1978 automobile data with replicate weights)

. bootstrap, fweights(f1-f100): regress mpg weight gear i.foreign
(running regress on estimation sample)

Bootstrap replications (100): .........10.........20.........30.........40.....
> ....50.........60.........70.........80.........90.........100 done

Linear regression                                      Number of obs =     74
                                                       Replications  =    100
                                                       Wald chi2(3)  = 130.33
                                                       Prob > chi2   = 0.0000
                                                       R-squared     = 0.6670
                                                       Adj R-squared = 0.6527
                                                       Root MSE      = 3.4096

------------------------------------------------------------------------------
             |   Observed   Bootstrap                         Normal-based
         mpg | coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
      weight |   -.006139   .0006435    -9.54   0.000    -.0074002   -.0048778
  gear_ratio |   1.457113   1.495479     0.97   0.330    -1.473972    4.388199
             |
     foreign |
    Foreign  |  -2.221682   1.323902    -1.68   0.093    -4.816482    .3731185
       _cons |   36.10135   5.560168     6.49   0.000     25.20362    46.99908
------------------------------------------------------------------------------
```

Because the frequency weights are predetermined, we do not need to specify the number of replications with the `reps()` option or the random-number seed. The command above will issue the `regress` command with `[fw=f1]` for the first replication, `[fw=f2]` for the second replication, and so on up through the 100th replication.

Similarly, if we wanted to apply importance weights when bootstrapping our statistics, we could specify the replicate weights `w1-w100` in the `iweights()` option:

```
. bootstrap, iweights(w1-w100): regress mpg weight gear i.foreign
(running regress on estimation sample)

Bootstrap replications (100): .........10.........20.........30.........40.....
> ....50.........60.........70.........80.........90.........100 done

Linear regression                                      Number of obs =     74
                                                       Replications  =    100
                                                       Wald chi2(3)  = 191.58
                                                       Prob > chi2   = 0.0000
                                                       R-squared     = 0.6670
                                                       Adj R-squared = 0.6527
                                                       Root MSE      = 3.4096

------------------------------------------------------------------------------
             |   Observed   Bootstrap                         Normal-based
         mpg | coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
      weight |   -.006139   .0005861   -10.48   0.000    -.0072877   -.0049904
  gear_ratio |   1.457113   1.454702     1.00   0.317     -1.39405    4.308276
             |
     foreign |
    Foreign  |  -2.221682    1.02959    -2.16   0.031     -4.23964   -.2037225
       _cons |   36.10135    5.48608     6.58   0.000     25.34883    46.85387
------------------------------------------------------------------------------
```

The command above will issue the `regress` command with `[iw=w1]` for the first replication, `[iw=w2]` for the second replication, and so on up through the 100th replication.

◁

▷ Example 10

Below, we continue working with `wauto.dta` and generate a categorical variable based on the values of `displacement`. We wish to bootstrap the difference in coefficients estimated by `regress` using frequency weights. After generating the categorical variable `dlevel`, we specify this variable as the cluster identifier and `foreign` as the strata identifier when generating frequency weights with `rwgen bsample`. Therefore, for each replication, we get a bootstrap sample of clusters within each stratum of `foreign`. Our frequency weight variables will be named `u1`, `u2`, ..., `u100`.

```
. use https://www.stata-press.com/data/r19/wauto
(1978 automobile data with replicate weights)

. generate dlevel = floor(displacement / 50)

. rwgen bsample u, reps(100) rseed(19) cluster(dlevel) strata(foreign)
```

We can now bootstrap the difference in coefficients estimated by `regress`, using the frequency weights to ensure that we obtain correct bootstrap estimates that account for the complexity of our data.

```
. bootstrap diff=(_b[weight] - _b[gear]), fweights(u1-u100): ///
> regress mpg weight gear
(running regress on estimation sample)

Bootstrap replications (100): .........10.........20.........30.........40.....
> ....50.........60.........70.........80.........90.........100 done

Linear regression

Number of strata = 2                                   Number of obs =  74
                                                       Replications  = 100

      Command: regress mpg weight gear
         diff: _b[weight] - _b[gear]
                                    (Replications based on 11 clusters in dlevel)

------------------------------------------------------------------------------
             |   Observed   Bootstrap                         Normal-based
             | coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
        diff |  -.1054525     1.38734    -0.08   0.939    -2.824589    2.613684
------------------------------------------------------------------------------
```

The estimation results are calculated based on replications that account for the clusters defined by `dlevel` and the strata variable `foreign`. These results are equivalent to the output we would obtain with the following command:

```
bootstrap diff=(_b[weight] - _b[gear]), reps(100) rseed(19) ///
   cluster(dlevel) strata(foreign): regress mpg weight gear
```

◁

Bradley Efron (1938–) was born in 1938 in Minnesota and studied mathematics and statistics at Caltech and Stanford; he has lived in northern California since 1960. He has worked on empirical Bayes, survival analysis, exponential families, bootstrap and jackknife methods, and confidence intervals, in conjunction with applied work in biostatistics, astronomy, and physics.

Efron is a member of the US National Academy of Sciences and was awarded the US National Medal of Science in 2005. He is by any standards one of the world's leading statisticians: his work ranges from deep and elegant contributions in theoretical statistics to pathbreaking involvement in a variety of practical applications.

Stored results

`bootstrap` stores the following in `e()`:

Scalars

`e(N)`	sample size
`e(N_reps)`	number of complete replications
`e(N_misreps)`	number of incomplete replications
`e(N_strata)`	number of strata
`e(N_clust)`	number of clusters
`e(k_eq)`	number of equations in `e(b)`
`e(k_exp)`	number of standard expressions
`e(k_eexp)`	number of extended expressions (that is, `_b`)
`e(k_extra)`	number of extra equations beyond the original ones from `e(b)`
`e(level)`	confidence level for bootstrap CIs
`e(bs_version)`	version for `bootstrap` results
`e(rank)`	rank of `e(V)`

Macros

`e(cmdname)`	command name from *command*
`e(cmd)`	same as `e(cmdname)` or `bootstrap`
`e(command)`	*command*
`e(cmdline)`	command as typed
`e(prefix)`	`bootstrap`
`e(title)`	title in estimation output
`e(rwtype)`	replicate weight type
`e(rwvarlist)`	replicate weight variables
`e(strata)`	strata variables
`e(cluster)`	cluster variables
`e(rngstate)`	random-number state used
`e(size)`	from the `size(#)` option
`e(exp#)`	expression for the #th statistic
`e(ties)`	`ties`, if specified
`e(mse)`	`mse`, if specified
`e(vce)`	`bootstrap`
`e(vcetype)`	title used to label Std. err.
`e(properties)`	`b V`

Matrices

`e(b)`	observed statistics
`e(b_bs)`	bootstrap estimates
`e(reps)`	number of nonmissing results
`e(bias)`	estimated biases
`e(se)`	estimated standard errors
`e(z0)`	median biases
`e(accel)`	estimated accelerations
`e(ci_normal)`	normal-approximation CIs
`e(ci_percentile)`	percentile CIs
`e(ci_bc)`	bias-corrected CIs
`e(ci_bca)`	bias-corrected and accelerated CIs
`e(V)`	bootstrap variance–covariance matrix
`e(V_modelbased)`	model-based variance

Functions

`e(sample)`	marks estimation sample

When *exp_list* is `_b`, `bootstrap` will also carry forward most of the results already in `e()` from *command*.

In addition to the above, the following is stored in `r()`:

Matrices	
`r(table)`	matrix containing the coefficients with their standard errors, test statistics, p-values, and confidence intervals

Note that results stored in `r()` are updated when the command is replayed and will be replaced when any r-class command is run after the estimation command.

Methods and formulas

Let $\widehat{\theta}$ be the observed value of the statistic, that is, the value of the statistic calculated with the original dataset. Let $i = 1, 2, \dots, k$ denote the bootstrap samples, and let $\widehat{\theta}_i$ be the value of the statistic from the ith bootstrap sample.

When the `mse` option is specified, the standard error is estimated as

$$\widehat{\mathrm{se}}_{\mathrm{MSE}} = \left\{ \frac{1}{k} \sum_{i=1}^{k} (\widehat{\theta}_i - \widehat{\theta})^2 \right\}^{1/2}$$

Otherwise, the standard error is estimated as

$$\widehat{\mathrm{se}} = \left\{ \frac{1}{k-1} \sum_{i=1}^{k} (\widehat{\theta}_i - \overline{\theta})^2 \right\}^{1/2}$$

where

$$\overline{\theta} = \frac{1}{k} \sum_{i=1}^{k} \widehat{\theta}_i$$

The variance–covariance matrix is similarly computed. The bias is estimated as

$$\widehat{\mathrm{bias}} = \overline{\theta} - \widehat{\theta}$$

Confidence intervals with nominal coverage rates $1 - \alpha$ are calculated according to the following formulas. The normal-approximation method yields the confidence intervals

$$\left[\, \widehat{\theta} - z_{1-\alpha/2}\, \widehat{\mathrm{se}},\ \widehat{\theta} + z_{1-\alpha/2}\, \widehat{\mathrm{se}} \,\right]$$

where $z_{1-\alpha/2}$ is the $(1-\alpha/2)$th quantile of the standard normal distribution. If the `mse` option is specified, `bootstrap` will report the normal confidence interval using $\widehat{\mathrm{se}}_{\mathrm{MSE}}$ instead of $\widehat{\mathrm{se}}$. `estat bootstrap` only uses $\widehat{\mathrm{se}}$ in the normal confidence interval.

The percentile method yields the confidence intervals

$$\left[\,\theta^*_{\alpha/2},\ \theta^*_{1-\alpha/2}\,\right]$$

where θ^*_p is the pth quantile (the $100p$th percentile) of the bootstrap distribution $(\widehat{\theta}_1, \dots, \widehat{\theta}_k)$.

Let

$$z_0 = \Phi^{-1}\{\#(\widehat{\theta}_i \le \widehat{\theta})/k\}$$

where $\#(\widehat{\theta}_i \le \widehat{\theta})$ is the number of elements of the bootstrap distribution that are less than or equal to the observed statistic and Φ is the standard cumulative normal. z_0 is known as the median bias of $\widehat{\theta}$. When the `ties` option is specified, z_0 is estimated as $\#(\widehat{\theta}_i < \widehat{\theta}) + \#(\widehat{\theta}_i = \widehat{\theta})/2$, which is the number of elements of the bootstrap distribution that are less than the observed statistic plus half the number of elements that are equal to the observed statistic.

Let

$$a = \frac{\sum_{i=1}^n (\overline{\theta}_{(\cdot)} - \widehat{\theta}_{(i)})^3}{6\{\sum_{i=1}^n (\overline{\theta}_{(\cdot)} - \widehat{\theta}_{(i)})^2\}^{3/2}}$$

where $\widehat{\theta}_{(i)}$ are the leave-one-out (jackknife) estimates of $\widehat{\theta}$ and $\overline{\theta}_{(\cdot)}$ is their mean. This expression is known as the jackknife estimate of acceleration for $\widehat{\theta}$. Let

$$p_1 = \Phi\left\{ z_0 + \frac{z_0 - z_{1-\alpha/2}}{1 - a(z_0 - z_{1-\alpha/2})} \right\}$$

$$p_2 = \Phi\left\{ z_0 + \frac{z_0 + z_{1-\alpha/2}}{1 - a(z_0 + z_{1-\alpha/2})} \right\}$$

where $z_{1-\alpha/2}$ is the $(1-\alpha/2)$th quantile of the normal distribution. The bias-corrected and accelerated (BC_a) method yields confidence intervals

$$\left[\,\theta^*_{p_1},\ \theta^*_{p_2}\,\right]$$

where θ^*_p is the pth quantile of the bootstrap distribution as defined previously. The bias-corrected (but not accelerated) method is a special case of BC_a with $a = 0$.

References

Ängquist, L. 2010. Stata tip 92: Manual implementation of permutations and bootstraps. *Stata Journal* 10: 686–688.

Cameron, A. C., and P. K. Trivedi. 2022. *Microeconometrics Using Stata*. 2nd ed. College Station, TX: Stata Press.

Davison, A. C., and D. V. Hinkley. 1997. *Bootstrap Methods and Their Application*. Cambridge: Cambridge University Press. https://doi.org/10.1017/CBO9780511802843.

Dorta, M., and G. Sánchez. 2021. Bootstrap unit-root test for random walk with drift: The bsrwalkdrift command. *Stata Journal* 21: 39–50.

Efron, B. 1979. Bootstrap methods: Another look at the jackknife. *Annals of Statistics* 7: 1–26. https://doi.org/10.1214/aos/1176344552.

———. 1982. *The Jackknife, the Bootstrap and Other Resampling Plans*. Philadelphia: Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9781611970319.

Efron, B., and C. Stein. 1981. The jackknife estimate of variance. *Annals of Statistics* 9: 586–596. https://doi.org/10.1214/aos/1176345462.

Efron, B., and R. J. Tibshirani. 1986. Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy. *Statistical Science* 1: 54–77. https://doi.org/10.1214/ss/1177013815.

———. 1993. *An Introduction to the Bootstrap*. New York: Chapman and Hall/CRC. https://doi.org/10.1201/9780429246593.

Field, C. A., and A. H. Welsh. 2007. Bootstrapping clustered data. *Journal of the Royal Statistical Society*, B ser., 69: 369–390. https://doi.org/10.1111/j.1467-9868.2007.00593.x.

Guan, W. 2003. From the help desk: Bootstrapped standard errors. *Stata Journal* 3: 71–80.

Hall, P., and S. R. Wilson. 1991. Two guidelines for bootstrap hypothesis testing. *Biometrics* 47: 757–762. https://doi.org/10.2307/2532163.

Hinkley, D. V. 1978. Improving the jackknife with special reference to correlation estimation. *Biometrika* 65: 13–22. https://doi.org/10.1093/biomet/65.1.13.

Holmes, S., C. N. Morris, and R. J. Tibshirani. 2003. Bradley Efron: A conversation with good friends. *Statistical Science* 18: 268–281. https://doi.org/10.1214/ss/1063994981.

Huber, C. 2013. Measures of effect size in Stata 13. *The Stata Blog: Not Elsewhere Classified*. https://blog.stata.com/2013/09/05/measures-of-effect-size-in-stata-13/.

Mooney, C. Z., and R. D. Duval. 1993. *Bootstrapping: A Nonparametric Approach to Statistical Inference*. Newbury Park, CA: Sage.

Ng, E. S.-W., R. Grieve, and J. R. Carpenter. 2013. Two-stage nonparametric bootstrap sampling with shrinkage correction for clustered data. *Stata Journal* 13: 141–164.

Poi, B. P. 2004. From the help desk: Some bootstrapping techniques. *Stata Journal* 4: 312–328.

Roodman, D., J. G. MacKinnon, M. Ø. Nielsen, and M. D. Webb. 2019. Fast and wild: Bootstrap inference in Stata using boottest. *Stata Journal* 19: 4–60.

Royston, P., and W. Sauerbrei. 2009. Bootstrap assessment of the stability of multivariable models. *Stata Journal* 9: 547–570.

Ruhe, C. 2019. Bootstrap pointwise confidence intervals for covariate-adjusted survivor functions in the Cox model. *Stata Journal* 19: 185–199.

Schwarz, C. 2019. lsemantica: A command for text similarity based on latent semantic analysis. *Stata Journal* 19: 129–142.

Stine, R. 1990. "An introduction to bootstrap methods: Examples and ideas". In *Modern Methods of Data Analysis*, edited by J. Fox and J. S. Long, 353–373. Newbury Park, CA: Sage.

Also see

[R] **bootstrap postestimation** — Postestimation tools for bootstrap

[R] **bayesboot** — Bayesian bootstrap estimation

[R] **jackknife** — Jackknife estimation

[R] **permute** — Permutation tests

[R] **rwgen** — Generate replicate weights for bootstrap estimation

[R] **simulate** — Monte Carlo simulations

[R] **set rngstream** — Specify the stream for the stream random-number generator

[SVY] **svy bootstrap** — Bootstrap for survey data

[U] 13.5 Accessing coefficients and standard errors

[U] 13.6 Accessing results from Stata commands

[U] 20 Estimation and postestimation commands

For suggested citations, see the FAQ on citing Stata documentation.