splitsample — Split data into random samples

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

Description

`splitsample` splits data into random samples based on a specified number of samples and specified proportions for each sample. Splitting can also be done based on clusters. Sample splitting can also be balanced across specified variables. Balanced splitting can be used for matched treatment assignment.

Quick start

Split data into two random samples of equal sizes and generate sample ID variable `svar` with values 1 and 2

```
splitsample, generate(svar)
```

Same as above, but with sample ID variable `svar` having values 0 and 1

```
splitsample, generate(svar) values(0 1)
```

Split data into three random samples of equal sizes and generate sample ID variable `svar` with values 1, 2, and 3

```
splitsample, generate(svar) nsplit(3)
```

Same as above, but with sample ID variable `svar` equal to missing (`.`) whenever any of `y` or `x1-x100` have missing values

```
splitsample y x1-x100, generate(svar) nsplit(3)
```

Split data into three random samples with the first sample having 25% of the observations, the second having 25%, and the third having 50%

```
splitsample, generate(svar) split(0.25 0.25 0.5)
```

Same sample split as above, but specify the split using ratios rather than proportions

```
splitsample, generate(svar) split(1 1 2)
```

Same as above, but maintain the specified sample-size ratios in each group defined by the variables `agegrp` and `gender`

```
splitsample, generate(svar) split(1 1 2) balance(agegrp gender)
```

Same as above, but randomly round sample sizes when samples within an `agegrp` by `gender` group cannot be chosen to satisfy the specified sample-size ratios exactly

```
splitsample, generate(svar) split(1 1 2) balance(agegrp gender) rround
```

Split data into three samples based on clusters defined by `clustvar`

```
splitsample, generate(svar) nsplit(3) cluster(clustvar)
```

Same as above, but maintain the specified sample proportions based on clusters in each group defined by the variables agegrp and gender, randomly round cluster sample sizes, and display a table showing the cluster sample sizes

```
splitsample, generate(svar) nsplit(3) cluster(clustvar) ///
  balance(agegrp gender) rround show
```

Menu

Data > Create or change data > Other variable-creation commands > Split data into random samples

Syntax

```
splitsample [varlist] [if] [in], generate(newvar [, replace]) [options]
```

varlist is checked for missing values, and the sample ID variable *newvar* is set to missing for observations where any variable in *varlist* is missing. _all or * may be specified for *varlist*.

options	Description
Main	
* generate(*newvar* [, replace])	create new sample ID variable; optionally replace existing variable
nsplit(#)	split into # random samples of equal size
split(*numlist*)	specify *numlist* of proportions or ratios for the split
rround	randomly round sample sizes when an exact split cannot be made
values(*numlist*)	specify *numlist* of values for sample ID variable
cluster(*clustvar*)	split by clusters defined by *clustvar*, not observations
balance(*balvars*)	split each group defined by the distinct values of *balvars* independently based on the specified sample proportions
Advanced	
strok	evaluate string variables in *varlist* for missing values; by default, string variables are ignored
rseed(#)	specify random-number seed
show	display a table showing the sample sizes of the split
percent	display percentages in the table showing the split

*generate() is required.

collect is allowed; see **[U] 11.1.10 Prefix commands**.

Options

Main

generate(*newvar* [, replace]) creates a new variable containing ID values for the random samples. The variable *newvar* is valued 1, 2, ...by default. The option values(*numlist*) can be used to specify different ID values. generate() is required.

replace allows any existing variable named *newvar* to be replaced.

nsplit(#) splits the data into # random samples of equal size, or as close to equal as possible. If neither nsplit() nor split() is specified, the data are split into two samples.

`split(`*numlist*`)` is an alternative to `nsplit()` for specifying the split. This option splits the data into samples whose sizes are proportional to the values of *numlist*. The values of *numlist* can be any positive number. You can specify proportions that sum to 1, or you can specify integers that define ratios for the sample sizes. Regardless of whether you specify decimals less than 1 or integers, the proportions of the split are given by the values in *numlist* divided by their sum.

`rround` specifies that sample sizes be randomly rounded when an exact split cannot be made. When an exact split can be made, this option does nothing. When `split(`*numlist*`)` is specified with `rround`, *numlist* must consist of integers, and the integers should contain no common factors. For instance, use `split(1 1 2)`, not `split(25 25 50)`. See *Methods and formulas* for an explanation.

By default, the sample sizes of the splits are calculated using a deterministic rounding formula. That is, if you repeat the splitting with a different random-number seed, you will get exactly the same sample sizes. Specifying `rround` creates randomly rounded sample sizes such that the expected values of the sample sizes match the specified split proportions exactly.

The option `rround` is designed for use with the `balance()` option when the number of observations in each of the balance groups is small. When group sizes are small (especially when smaller than the number of splits), `rround` ensures that the overall actual sample split proportions closely match the specified split proportions.

`values(`*numlist*`)` specifies that *numlist* be used for the values of the sample ID variable rather than the default of 1, 2, ... The number of values in *numlist* must correspond to the number of samples into which the data are split and must be ascending nonnegative integers.

`cluster(`*clustvar*`)` specifies that the data be split by the clusters defined by *clustvar*. That is, all observations in a cluster are kept together in the same split sample. The proportions of the split are based on numbers of clusters, not numbers of observations. *clustvar* can be a numeric or string variable.

`balance(`*balvars*`)` specifies that each group defined by the distinct values of *balvars* be split independently based on the specified sample proportions. This ensures a balanced, or roughly balanced, distribution of the *balvars* values across the split samples. When the number of observations (or clusters) in each group is about the same as (or smaller than) the number of split samples, the option `rround` is recommended. *balvars* can be numeric or string variables.

Advanced

`strok` (applies only when a *varlist* is specified) specifies to check any string variables in *varlist* for missing values. For observations with missing values, the generated sample ID variable is set to missing. By default, string variables in *varlist* are ignored.

`rseed(#)` sets the random-number seed. This option can be used to reproduce results. `rseed(#)` is equivalent to typing `set seed #` prior to running `splitsample`. See [R] **set seed**.

`show` displays a table showing the sample sizes of the split. When `cluster()` is specified, it shows the numbers of clusters in the samples. When `balance(`*balvars*`)` is specified, it displays a table in which each row corresponds to a distinct set of values of *balvars* and shown across the columns are the numbers of observations (or clusters) belonging to each split sample for that balance group.

`percent` specifies to display percentages rather than the number of observations (or clusters) in the table. `percent` can only be specified with the option `show`.

Remarks and examples

splitsample is useful for dividing data into training, validation, and testing samples for machine learning and automated model-building procedures such as those performed by the lasso, stepwise, and nestreg commands.

splitsample with the options balance() and rround can also be used to do random treatment assignment with matching. See example 3.

Example 1: Splitting by observations

Let's create a dataset with 101 observations and run splitsample without any options except the required option giving the name of the sample ID variable to generate. Then we tabulate the newly created variable.

```
. set obs 101
Number of observations (_N) was 0, now 101.
. splitsample, generate(svar)
. tabulate svar
```

svar	Freq.	Percent	Cum.
1	51	50.50	50.50
2	50	49.50	100.00
Total	101	100.00	

By default, splitsample splits the data into two samples, with the samples as equal in size as possible.

The option nsplit(#) can be used to split the data into as many samples as you want—in this case, three samples.

```
. splitsample, generate(svar, replace) nsplit(3)
. tabulate svar
```

svar	Freq.	Percent	Cum.
1	34	33.66	33.66
2	33	32.67	66.34
3	34	33.66	100.00
Total	101	100.00	

The option split(*numlist*) can be specified in place of nsplit() to split the data into any proportions you want. Here we specify that we want 25% of the observations in sample 1, 25% in sample 2, and 50% in sample 3.

```
. splitsample, generate(svar, replace) split(0.25 0.25 0.50) show
```

svar	Freq.	Percent	Cum.
1	25	24.75	24.75
2	26	25.74	50.50
3	50	49.50	100.00
Total	101	100.00	

It split the data as close as it could to 25% : 25% : 50%. The option show displayed the tabulation for us.

◁

▷ Example 2: Splitting by clusters

splitsample can also split the data by clusters. Let's create a cluster variable clustvar and split the data into three samples with proportions 25% : 25% : 50% for the numbers of clusters. We also specify the option show, which gives a convenient tabulation by numbers of clusters rather than numbers of observations.

```
. set seed 12345
. generate clustvar = runiformint(1, 20)
. splitsample, generate(svar, replace) split(0.25 0.25 0.50) cluster(clustvar)
> show
```

svar	Freq.	Percent	Cum.
1	5	25.00	25.00
2	5	25.00	50.00
3	10	50.00	100.00
Total	20	100.00	

Total is number of clusters.

Because we had 20 clusters, the split into 25% : 25% : 50% yielded cluster sample sizes that met the specified proportions exactly.

The resulting split by number of observations is, of course, different.

```
. tabulate svar
```

svar	Freq.	Percent	Cum.
1	34	33.66	33.66
2	21	20.79	54.46
3	46	45.54	100.00
Total	101	100.00	

When splitting by clusters, the size of each cluster is ignored.

◁

▷ Example 3: Balanced splitting and treatment assignment

splitsample can split the data independently within groups using the option balance(). Let's create two fake categorical variables, one agegrp representing eight age–group categories, and a 0/1 variable gender.

```
. set seed 12345
. generate agegrp = runiformint(1, 8)
. generate gender = runiformint(0, 1)
```

We want to split the data into four samples, where the first three samples are the same size, and the fourth sample is twice the size of each of the others. We specify split(1 1 1 2) using integer ratios. We specify the option balance(agegrp gender) to ensure that the distribution of agegrp × gender is roughly balanced across the four samples. The option show is useful for seeing the actual splits of the numbers of observations within each agegrp × gender group.

```
. splitsample, generate(svar, replace) split(1 1 1 2)
> balance(agegrp gender) show
note: some groups defined by balance() do not contain every sample value.
```

agegrp	gender	svar 1	svar 2	svar 3	svar 4	Total
1	0	2	1	2	3	8
1	1	1	2	1	3	7
2	0	2	2	1	4	9
2	1	1	1	1	2	5
3	0	1	1	1	2	5
3	1	1	1	0	2	4
4	0	2	2	2	4	10
4	1	2	2	1	4	9
5	0	1	0	1	1	3
5	1	1	0	1	1	3
6	0	1	0	1	1	3
6	1	2	2	1	4	9
7	0	0	1	0	1	2
7	1	1	1	1	2	5
8	0	2	1	2	3	8
8	1	2	2	3	4	11

We get a message "some groups defined by `balance()` do not contain every sample value". Indeed, all the groups of size three have no observations in sample 2. Because we are splitting the data into four samples, obviously we need at least four observations in a group for every sample to contain at least one observation.

Second, we notice that all groups of the same size are split into the four samples with exactly the same number of observations in each sample. For example, the two groups of size eight (`agegrp` $=1$, `gender` $=0$ and `agegrp` $=8$, `gender` $=0$) both have two observations in each of samples 1 and 3, one observation in sample 2, and three observations in sample 4.

Groups of the same size have exactly the same sample-size splits because, by default, the sample sizes for the splits are calculated using a deterministic formula. If the sizes of the groups vary, this typically would not be an issue. Overall, one would expect the actual split proportions to be close to the specified split proportions. But imagine if all, or almost all, the group sizes were the same. What if the size of each group were eight observations in this example? Every group would be split $2:1:2:3$ by observations, yielding actual split proportions of $25\%:12.5\%:25\%:37.5\%$, which are rather different from the specified split proportions of $20\%:20\%:20\%:40\%$.

The option rround provides a solution for this problem. It randomly rounds the split sample sizes when the split cannot be made exactly.

```
. splitsample, generate(svar, replace) split(1 1 1 2)
> balance(agegrp gender) rround rseed(54321) show
note: some groups defined by balance() do not contain every sample value.
```

agegrp	gender	svar 1	svar 2	svar 3	svar 4	Total
1	0	2	1	2	3	8
1	1	2	1	1	3	7
2	0	2	2	1	4	9
2	1	1	1	1	2	5
3	0	1	1	1	2	5
3	1	1	1	1	1	4
4	0	2	2	2	4	10
4	1	2	2	2	3	9
5	0	1	1	0	1	3
5	1	1	1	1	0	3
6	0	0	1	1	1	3
6	1	1	2	2	4	9
7	0	0	0	0	2	2
7	1	1	1	1	2	5
8	0	1	2	2	3	8
8	1	2	2	2	5	11

We see that the groups of sizes three, eight, and nine now have different splits by numbers of observations. The groups of size five have exactly the same splits by size because they could be divided exactly based on the specified split ratios of $1 : 1 : 1 : 2$.

The option rround with balance() thus does a "more random" assignment of observations (or clusters), which is important when the sizes of the balance groups are small. When the sizes of the balance groups are large, and the sizes of the groups vary, splits made with or without rround will be similar.

Note that rround with balance() is suitable for random treatment assignment with matching defined by values of the balance variables.

The computational procedure for option rround first randomly assigns as many observations to the split samples as it can to match the specified split proportions exactly. Leftover observations are assigned to samples by dividing them randomly based on the specified split ratios. Splitting ratios must be specified as integers to facilitate this method of splitting the leftovers. See *Methods and formulas*.

◁

▷ Example 4: Missing values

varlist can be specified with splitsample to handle missing values. Let's say we want to divide our data into training and validation samples for a lasso or other procedure. Imagine that the variables in the lasso have more than a few missing values. Specifying these variables as *varlist* for splitsample means that the sample ID variable created will have missing values whenever any of the variables in *varlist* are missing.

Here's an illustration. We create a couple of variables with missing values.

```
. set seed 1234
. generate y = runiform()
. replace  y = . if runiform() < 0.1
(11 real changes made, 11 to missing)
. generate x = runiform()
. replace  x = . if runiform() < 0.1
(15 real changes made, 15 to missing)
```

Then split the data specifying these variables to be checked for missing:

```
. splitsample y x, generate(svar, replace)
. tabulate svar, miss
```

svar	Freq.	Percent	Cum.
1	38	37.62	37.62
2	38	37.62	75.25
.	25	24.75	100.00
Total	101	100.00	

The split was done exactly for the observations without missing values.

◁

Stored results

`splitsample` stores the following in `r()`:

Scalars

`r(N)`	total number of observations
`r(N_clust)`	total number of clusters
`r(n_samples)`	number of split samples

Macros

`r(clustvar)`	name of cluster variable
`r(balancevars)`	names of balance variables
`r(rngstate)`	random-number state used

Methods and formulas

Let $r_1, r_2, \ldots, r_K$ be the arguments to `split(`*numlist*`)`. If the split is specified using `nsplit(#)`, then we set each $r_k = 1$, and the number of split samples is $K = \#$. The split sample proportions are

$$p_k = \frac{r_k}{R} \quad \text{where } R = \sum_{i=1}^{K} r_i$$

The cumulative proportions are

$$s_k = \sum_{i=1}^{k} p_i$$

For the default deterministic rounding, we calculate cumulative sample sizes:

$$M_k = \text{round}(N s_k)$$

where N is the total number of observations or the number of clusters, and round$(\cdot)$ is Stata's `round()` function. When the option `balance()` is specified, N is the number of observations or clusters in a single balance group. The sample sizes $N_1, N_2, \ldots, N_K$ are given by

$$N_1 = M_1$$
$$N_k = M_k - M_{k-1} \quad \text{for } k = 2, \ldots, K$$

When the option `rround` is specified for random rounding, we first divide N, the number of observations or clusters, as follows:

$$N = cR + d$$

where R is the sum of $r_1, r_2, \ldots, r_K$; c is a nonnegative integer; and $0 \leq d < R$. In other words, cR observations can be split into K samples matching the specified split proportions exactly. We randomly pick cR observations and assign them to the samples. The leftover d observations are randomly placed in R bins without replacement, where the first r_1 bins represent sample 1, the next r_2 bins represent sample 2, and so on.

The computational procedure for random rounding thus requires $r_1, r_2, \ldots, r_K$ to be integers and also requires $R \leq N$. To reduce the variance of the random rounding, the integers $r_1, r_2, \ldots, r_K$ should have no common factors.

Also see

[D] **sample** — Draw random sample

Stata, Stata Press, Mata, NetCourse, and NetCourseNow are registered trademarks of StataCorp LLC. Stata and Stata Press are registered trademarks with the World Intellectual Property Organization of the United Nations. StataNow is a trademark of StataCorp LLC. Other brand and product names are registered trademarks or trademarks of their respective companies. Copyright © 1985–2025 StataCorp LLC, College Station, TX, USA. All rights reserved.

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