Title

stata.com

cluster programming utilities — Cluster-analysis programming utilities

Description

The `cluster query`, `cluster set`, `cluster delete`, `cluster parsedistance`, and `cluster measures` commands provide tools for programmers to add their own cluster-analysis subroutines to Stata's `cluster` command; see [MV] **cluster** and [MV] **cluster programming subroutines**. These commands make it possible for the new command to take advantage of Stata's cluster-management facilities.

`cluster query` provides a way to obtain the various attributes of a cluster analysis in Stata. If *clname* is omitted, `cluster query` returns in `r(names)` a list of the names of all currently defined cluster analyses. If *clname* is provided, the various attributes of the specified cluster analysis are returned in `r()`. These attributes include the type, method, (dis)similarity used, created variable names, notes, and any other information attached to the cluster analysis.

`cluster set` allows you to set the various attributes that define a cluster analysis in Stata, including naming your cluster results and adding the name to the master list of currently defined cluster results. With `cluster set`, you can provide information on the type, method, and (dis)similarity measure of your cluster-analysis results. You can associate variables and Stata characteristics (see [P] **char**) with your cluster analysis. `cluster set` also allows you to add notes and other specified fields to your cluster-analysis result. These items become part of the dataset and are saved with the data.

`cluster delete` allows you to delete attributes from a cluster analysis in Stata. This command is the inverse of `cluster set`.

`cluster parsedistance` takes the similarity or dissimilarity *measure* name and checks it against the list of those provided by Stata, taking account of allowed minimal abbreviations and aliases. Aliases are resolved (for instance, `Euclidean` is changed into the equivalent `L2`).

`cluster measures` computes the similarity or dissimilarity *measure* between the observations listed in the `compare()` option and the observations included based on the `if` and `in` conditions and places the results in the variables specified by the `generate()` option. See [MV] **matrix dissimilarity** for the `matrix dissimilarity` command that places (dis)similarities in a matrix.

Stata also provides a method for programmers to extend the `cluster` command by providing subcommands; see [MV] **cluster programming subroutines**.

Syntax

Obtain various attributes of a cluster analysis

```
cluster query [clname]
```

Set various attributes of a cluster analysis

```
cluster set [clname] [, set_options]
```

Delete attributes from a cluster analysis

```
cluster delete clname [, delete_options]
```

Check similarity and dissimilarity measure name

```
cluster parsedistance measure
```

Compute similarity and dissimilarity measure

```
cluster measures varlist [if] [in], compare(numlist) generate(newvarlist)
    [measures_options]
```

set_options	Description
addname	add *clname* to the master list of cluster analyses
type(*type*)	set the cluster type for *clname*
method(*method*)	set the name of the clustering method for the cluster analysis
similarity(*measure*)	set the name of the similarity measure used for the cluster analysis
dissimilarity(*measure*)	set the name of the dissimilarity measure used for the cluster analysis
var(*tag varname*)	set *tag* that points to *varname*
char(*tag charname*)	set *tag* that points to *charname*
other(*tag text*)	set *tag* with *text* attached to the tag marker
note(*text*)	add a note to the *clname*

delete_options	Description
zap	delete all possible settings for *clname*
delname	remove *clname* from the master list of current cluster analyses
type	delete the cluster type entry from *clname*
method	delete the cluster method entry from *clname*
similarity	delete the similarity entries from *clname*
dissimilarity	delete the dissimilarity entries from *clname*
notes(*numlist*)	delete the specified numbered notes from *clname*
allnotes	remove all notes from *clname*
var(*tag*)	remove *tag* from *clname*
allvars	remove all the entries pointing to variables for *clname*
varzap(*tag*)	same as var(), but also delete the referenced variable
allvarzap	same as allvars, but also delete the variables
char(*tag*)	remove *tag* that points to a Stata characteristic from *clname*
allchars	remove all entries pointing to Stata characteristics for *clname*
charzap(*tag*)	same as char(), but also delete the characteristic
allcharzap	same as allchars, but also delete the characteristics
other(*tag*)	delete *tag* and its associated text from *clname*
allothers	delete all entries from *clname* that have been set using other()

measures_options	Description
* compare(*numlist*)	use *numlist* as the comparison observations
* generate(*newvarlist*)	create *newvarlist* variables
measure	(dis)similarity measure; see *Options for cluster measures* for available measures; default is L2
propvars	interpret observations implied by if and in as proportions of binary observations
propcompares	interpret comparison observations as proportions of binary observations

* compare(*numlist*) and generate(*newvarlist*) are required.

Options for cluster set

addname adds *clname* to the master list of currently defined cluster analyses. When *clname* is not specified, the addname option is mandatory, and here, cluster set automatically finds a cluster name that is not currently in use and uses this as the cluster name. cluster set returns the name of the cluster in r(name). If addname is not specified, the *clname* must have been added to the master list previously (for instance, through a previous call to cluster set).

type(*type*) sets the cluster type for *clname*. type(hierarchical) indicates that the cluster analysis is hierarchical-style clustering, and type(partition) indicates that it is a partition-style clustering. You are not restricted to these types. For instance, you might program some kind of fuzzy partition-clustering analysis, so you then use type(fuzzy).

method(*method*) sets the name of the clustering method for the cluster analysis. For instance, Stata uses method(kmeans) to indicate a kmeans cluster analysis and uses method(single) to indicate single-linkage cluster analysis. You are not restricted to the names currently used within Stata.

`similarity(`*measure*`)` and `dissimilarity(`*measure*`)` set the name of the similarity or dissimilarity measure used for the cluster analysis. For example, Stata uses `dissimilarity(L2)` to indicate the L2 or Euclidean distance. You are not restricted to the names currently used within Stata. See [MV] ***measure_option*** and [MV] **cluster** for a listing and discussion of (dis)similarity measures.

`var(`*tag varname*`)` sets a marker called *tag* in the cluster analysis that points to the variable *varname*. For instance, Stata uses `var(group` *varname*`)` to set a grouping variable from a kmeans cluster analysis. With single-linkage clustering, Stata uses `var(id` *idvarname*`)`, `var(order` *ordervarname*`)`, and `var(height` *hgtvarname*`)` to set the `id`, `order`, and `height` variables that define the cluster-analysis result. You are not restricted to the names currently used within Stata. Up to 10 `var()` options may be specified with a `cluster set` command.

`char(`*tag charname*`)` sets a marker called *tag* in the cluster analysis that points to the Stata characteristic named *charname*; see [P] **char**. This characteristic can be either an `_dta[]` dataset characteristic or a variable characteristic. Up to 10 `char()` options may be specified with a `cluster set` command.

`other(`*tag text*`)` sets a marker called *tag* in the cluster analysis with *text* attached to the *tag* marker. Stata uses `other(k #)` to indicate that `k` (the number of groups) was # in a kmeans cluster analysis. You are not restricted to the names currently used within Stata. Up to 10 `other()` options may be specified with a `cluster set` command.

`note(`*text*`)` adds a note to the *clname* cluster analysis. The `cluster notes` command (see [MV] **cluster notes**) is the command to add, delete, or view cluster notes. The `cluster notes` command uses the `note()` option of `cluster set` to add a note to a cluster analysis. Up to 10 `note()` options may be specified with a `cluster set` command.

Options for cluster delete

`zap` deletes all possible settings for cluster analysis *clname*. It is the same as specifying the `delname`, `type`, `method`, `similarity`, `dissimilarity`, `allnotes`, `allcharzap`, `allothers`, and `allvarzap` options.

`delname` removes *clname* from the master list of current cluster analyses. This option does not affect the various settings that make up the cluster analysis. To remove them, use the other options of `cluster delete`.

`type` deletes the cluster type entry from *clname*.

`method` deletes the cluster method entry from *clname*.

`similarity` and `dissimilarity` delete the similarity and dissimilarity entries, respectively, from *clname*.

`notes(`*numlist*`)` deletes the specified numbered notes from *clname*. The numbering corresponds to the returned results from the `cluster query` *clname* command. The `cluster notes drop` command (see [MV] **cluster notes**) drops a cluster note. It, in turn, calls `cluster delete`, using the `notes()` option to drop the notes.

`allnotes` removes all notes from the *clname* cluster analysis.

`var(`*tag*`)` removes from *clname* the entry labeled *tag* that points to a variable. This option does not delete the variable.

`allvars` removes all the entries pointing to variables for *clname*. This option does not delete the corresponding variables.

`varzap(`*tag*`)` is the same as `var()` and actually deletes the variable in question.

`allvarzap` is the same as `allvars` and actually deletes the variables.

`char(`*tag*`)` removes from *clname* the entry labeled *tag* that points to a Stata characteristic (see [P] **char**). This option does not delete the characteristic.

`allchars` removes all the entries pointing to Stata characteristics for *clname*. This option does not delete the characteristics.

`charzap(`*tag*`)` is the same as `char()` and actually deletes the characteristics.

`allcharzap` is the same as `allchars` and actually deletes the characteristics.

`other(`*tag*`)` deletes from *clname* the *tag* entry and its associated text, which were set by using the `other()` option of the `cluster set` command.

`allothers` deletes all entries from *clname* that have been set using the `other()` option of the `cluster set` command.

Options for cluster measures

`compare(`*numlist*`)` is required and specifies the observations to use as the comparison observations. Each of these observations will be compared with the observations implied by the `if` and `in` conditions, using the specified (dis)similarity *measure*. The results are stored in the corresponding new variable from the `generate()` option. There must be the same number of elements in *numlist* as there are variable names in the `generate()` option.

`generate(`*newvarlist*`)` is required and specifies the names of the variables to be created. There must be as many elements in *newvarlist* as there are numbers specified in the `compare()` option.

measure specifies the similarity or dissimilarity measure. The default is `L2` (synonym `Euclidean`). This option is not case sensitive. See [MV] ***measure_option*** for detailed descriptions of the supported measures.

`propvars` is for use with binary measures and specifies that the observations implied by the `if` and `in` conditions be interpreted as proportions of binary observations. The default action with binary measures treats all nonzero values as one (excluding missing values). With `propvars`, the values are confirmed to be between zero and one, inclusive. See [MV] ***measure_option*** for a discussion of the use of proportions with binary measures.

`propcompares` is for use with binary measures. It indicates that the comparison observations (those specified in the `compare()` option) are to be interpreted as proportions of binary observations. The default action with binary measures treats all nonzero values as one (excluding missing values). With `propcompares`, the values are confirmed to be between zero and one, inclusive. See [MV] ***measure_option*** for a discussion of the use of proportions with binary measures.

Remarks and examples

stata.com

▷ Example 1

Programmers can determine which cluster solutions currently exist by using the `cluster query` command without specifying a cluster name to return the names of all currently defined clusters.

```
. use http://www.stata-press.com/data/r14/auto
(1978 Automobile Data)
. cluster k gear turn trunk mpg displ, k(6) name(grpk6L2) measure(L2) gen(g6l2)
. cluster k gear turn trunk mpg displ, k(7) name(grpk7L2) measure(L2) gen(g7l2)
```

```
. cluster kmed gear turn trunk mpg displ, k(6) name(grpk6L1) measure(L1) gen(g6l1)
. cluster kmed gear turn trunk mpg displ, k(7) name(grpk7L1) measure(L1) gen(g7l1)
. cluster dir
grpk7L1
grpk6L1
grpk7L2
grpk6L2
. cluster query
. return list
macros:
              r(names) : "grpk7L1 grpk6L1 grpk7L2 grpk6L2"
```

Here there are four cluster solutions. A programmer can further process the `r(names)` returned macro. For example, to determine which current cluster solutions used kmeans clustering, we would loop through these four cluster solution names and, for each one, call `cluster query` to determine its properties.

```
. local clusnames `r(names)'
. foreach cname of local clusnames {
  2.         cluster query `cname'
  3.         if "`r(method)'" == "kmeans" {
  4.                 local kmeancls `kmeancls' `cname'
  5.         }
  6. }
. di "{tab}Cluster analyses using kmeans: `kmeancls'"
        Cluster analyses using kmeans: grpk7L2 grpk6L2
```

Here we examined `r(method)`, which records the name of the cluster-analysis method. Two of the four cluster solutions used kmeans.

◁

▷ Example 2

We interactively demonstrate `cluster set`, `cluster delete`, and `cluster query`, though in practice these would be used within a program.

First, we add the name `myclus` to the master list of cluster analyses and, at the same time, set the type, method, and similarity.

```
. cluster set myclus, addname type(madeup) method(fake) similarity(who knows)
. cluster query
. return list
macros:
              r(names) : "myclus grpk7L1 grpk6L1 grpk7L2 grpk6L2"
. cluster query myclus
. return list
macros:
               r(name) : "myclus"
         r(similarity) : "who knows"
             r(method) : "fake"
               r(type) : "madeup"
```

`cluster query` shows that `myclus` was successfully added to the master list of cluster analyses and that the attributes that were `cluster set` can also be obtained.

Now we add a reference to a variable. We will use the word `group` as the *tag* for a variable `mygrpvar`. We also add another item called `xyz` and associate some text with the `xyz` item.

```
. cluster set myclus, var(group mygrpvar) other(xyz some important info)
. cluster query myclus
. return list
macros:
               r(name) : "myclus"
             r(o1_val) : "some important info"
             r(o1_tag) : "xyz"
           r(groupvar) : "mygrpvar"
            r(v1_name) : "mygrpvar"
             r(v1_tag) : "group"
         r(similarity) : "who knows"
             r(method) : "fake"
               r(type) : "madeup"
```

The `cluster query` command returned the `mygrpvar` information in two ways. The first way is with `r(v#_tag)` and `r(v#_name)`. Here there is only one variable associated with `myclus`, so we have `r(v1_tag)` and `r(v1_name)`. This information allows the programmer to loop over all the stored variable names without knowing beforehand what the *tag*s might be or how many there are. You could loop as follows:

```
local i 1
while "`r(v`i'_tag)'" != "" {
        ...
        local ++i
}
```

The second way the variable information is returned is in an `r()` result with the *tag* name appended by `var`, `r(`*tag*`var)`. In our example, this is `r(groupvar)`. This second method is convenient when, as the programmer, you know exactly which *varname* information you are seeking.

The same logic applies to characteristic attributes that are `cluster set`.

Now we continue with our interactive example:

```
. cluster delete myclus, method var(group)
. cluster set myclus, note(a note) note(another note) note(a third note)
. cluster query myclus
. return list
macros:
              r(name) : "myclus"
             r(note3) : "a third note"
             r(note2) : "another note"
             r(note1) : "a note"
             r(o1_val) : "some important info"
             r(o1_tag) : "xyz"
         r(similarity) : "who knows"
               r(type) : "madeup"
```

We used `cluster delete` to remove the method and the `group` variable we had associated with `myclus`. Three notes were then added simultaneously by using the `note()` option of `cluster set`. In practice, users will use the `cluster notes` command (see [MV] **cluster notes**) to add and delete cluster notes. The `cluster notes` command is implemented with the `cluster set` and `cluster delete` programming commands.

We finish our interactive demonstration of these commands by deleting more attributes from `myclus` and then eliminating `myclus`. In practice, users would remove a cluster analysis with the `cluster drop` command (see [MV] **cluster utility**), which is implemented with the `zap` option of the `cluster delete` command.

```
. cluster delete myclus, allnotes similarity
. cluster query myclus
. return list
macros:
               r(name) : "myclus"
            r(o1_val) : "some important info"
            r(o1_tag) : "xyz"
              r(type) : "madeup"
. cluster delete myclus, zap
. cluster query
. return list
macros:
             r(names) : "grpk7L1 grpk6L1 grpk7L2 grpk6L2"
```

The cluster attributes that are `cluster set` become a part of the dataset. They are saved with the dataset when it is `saved` and are available again when the dataset is `used`; see [D] **save**.

◁

❑ Technical note

You may wonder how Stata's cluster-analysis data structures are implemented. Stata data characteristics (see [P] **char**) hold the information. The details of the implementation are not important, and in fact, we encourage you to use the `set`, `delete`, and `query` subcommands to access the cluster attributes. This way, if we ever decide to change the underlying implementation, you will be protected through Stata's version-control feature.

❑

▷ Example 3

The `cluster parsedistance` programming command takes as an argument the name of a similarity or dissimilarity measure. Stata then checks this name against those that are implemented within Stata (and available to you through the `cluster measures` command). Uppercase or lowercase letters are allowed, and minimal abbreviations are checked. Some of the measures have aliases, which are resolved so that a standard measure name is returned. We demonstrate the `cluster parsedistance` command interactively:

```
. cluster parsedistance max
. sreturn list
macros:
            s(drange) : "0 ."
             s(dtype) : "dissimilarity"
              s(unab) : "maximum"
              s(dist) : "Linfinity"
. cluster parsedistance Eucl
. sreturn list
macros:
            s(drange) : "0 ."
             s(dtype) : "dissimilarity"
              s(unab) : "Euclidean"
              s(dist) : "L2"
. cluster parsedistance correl
```

```
. sreturn list
macros:
             s(drange) : "1 -1"
              s(dtype) : "similarity"
               s(unab) : "correlation"
               s(dist) : "correlation"
. cluster parsedistance jacc
. sreturn list
macros:
             s(drange) : "1 0"
             s(binary) : "binary"
              s(dtype) : "similarity"
               s(unab) : "Jaccard"
               s(dist) : "Jaccard"
```

`cluster parsedistance` returns `s(dtype)` as either `similarity` or `dissimilarity`. It returns `s(dist)` as the standard Stata name for the (dis)similarity and returns `s(unab)` as the unabbreviated standard Stata name. `s(drange)` gives the range of the measure (most similar to most dissimilar). If the measure is designed for binary variables, `s(binary)` is returned with the word `binary`, as seen above.

See [MV] ***measure_option*** for a listing of the similarity and dissimilarity measures and their properties.

◁

▷ Example 4

`cluster measures` computes the similarity or dissimilarity measure between each comparison observation and the observations implied by the `if` and `in` conditions (or all the data if no `if` or `in` conditions are specified).

We demonstrate with the auto dataset:

```
. use http://www.stata-press.com/data/r14/auto, clear
(1978 Automobile Data)
. cluster measures turn trunk gear in 1/10, compare(3 11) gen(z3 z11) L1
. format z* %8.2f
. list turn trunk gear z3 z11 in 1/11
```

	turn	trunk	gear_r~o	z3	z11
1.	40	11	3.58	6.50	14.30
2.	40	11	2.53	6.55	13.25
3.	35	12	3.08	0.00	17.80
4.	40	16	2.93	9.15	8.65
5.	43	20	2.41	16.67	1.13
6.	43	21	2.73	17.35	2.45
7.	34	10	2.87	3.21	20.59
8.	42	16	2.93	11.15	6.65
9.	43	17	2.93	13.15	4.65
10.	42	13	3.08	8.00	9.80
11.	44	20	2.28	.	.

Using the three variables `turn`, `trunk`, and `gear_ratio`, we computed the L1 (or absolute value) distance between the third observation and the first 10 observations and placed the results in the variable `z3`. The distance between the 11th observation and the first 10 was placed in variable `z11`.

There are many measures designed for binary data. Below we illustrate `cluster measures` with the matching coefficient binary similarity measure. We have 8 observations on 10 binary variables, and we will compute the matching similarity measure between the last 3 observations and all 8 observations.

```
. use http://www.stata-press.com/data/r14/clprogxmpl1, clear
. cluster measures x1-x10, compare(6/8) gen(z6 z7 z8) matching
. format z* %4.2f
. list
```

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	z6	z7	z8
1.	1	0	0	0	1	1	0	0	1	1	0.60	0.80	0.40
2.	1	1	1	0	0	1	0	1	1	0	0.70	0.30	0.70
3.	0	0	1	0	0	0	1	0	0	1	0.60	0.40	0.20
4.	1	1	1	1	0	0	0	1	1	1	0.40	0.40	0.60
5.	0	1	0	1	1	0	1	0	0	1	0.20	0.60	0.40
6.	1	0	1	0	0	1	0	0	0	0	1.00	0.40	0.60
7.	0	0	0	1	1	1	0	0	1	1	0.40	1.00	0.40
8.	1	1	0	1	0	1	0	1	0	0	0.60	0.40	1.00

Stata treats all nonzero observations as one (except missing values, which are treated as missing values) when computing these binary measures.

When the similarity measure between binary observations and the means of groups of binary observations is needed, the `propvars` and `propcompares` options of `cluster measures` provide the solution. The mean of binary observations is a proportion. The value 0.2 would indicate that 20% of the values were one and 80% were zero for the group. See [MV] ***measure_option*** for a discussion of binary measures. The `propvars` option indicates that the main body of observations should be interpreted as proportions. The `propcompares` option specifies that the comparison observations be treated as proportions.

We compare 10 binary observations on five variables to 2 observations holding proportions by using the `propcompares` option:

```
. use http://www.stata-press.com/data/r14/clprogxmpl2, clear
. cluster measures a* in 1/10, compare(11 12) gen(c1 c2) matching propcompare
. list
```

	a1	a2	a3	a4	a5	c1	c2
1.	1	1	1	0	1	.6	.56
2.	0	0	1	1	1	.36	.8
3.	1	0	1	0	0	.76	.56
4.	1	1	0	1	1	.36	.44
5.	1	0	0	0	0	.68	.4
6.	0	0	1	1	1	.36	.8
7.	1	0	1	0	1	.64	.76
8.	1	0	0	0	1	.56	.6
9.	0	1	1	1	1	.32	.6
10.	1	1	1	1	1	.44	.6
11.	.8	.4	.7	.1	.2	.	.
12.	.5	0	.9	.6	1	.	.

Stored results

`cluster query` with no arguments stores the following in `r()`:

Macros

`r(names)`	cluster solution names

`cluster query` with an argument stores the following in `r()`:

Macros

`r(name)`	cluster name
`r(type)`	type of cluster analysis
`r(method)`	cluster-analysis method
`r(similarity)`	similarity measure name
`r(dissimilarity)`	dissimilarity measure name
`r(note#)`	cluster note number #
`r(v#_tag)`	variable tag number #
`r(v#_name)`	varname associated with `r(v#_tag)`
`r(`*tag*`var)`	varname associated with *tag*
`r(c#_tag)`	characteristic tag number #
`r(c#_name)`	characteristic name associated with `r(c#_tag)`
`r(c#_val)`	characteristic value associated with `r(c#_tag)`
`r(`*tag*`char)`	characteristic name associated with *tag*
`r(o#_tag)`	other tag number #
`r(o#_val)`	other value associated with `r(o#_tag)`

`cluster set` stores the following in `r()`:

Macros

`r(name)`	cluster name

`cluster parsedistance` stores the following in `s()`:

Macros

`s(dist)`	(dis)similarity measure name
`s(unab)`	unabbreviated (dis)similarity measure name (before resolving alias)
`s(darg)`	argument of (dis)similarities that take them, such as `L(#)`
`s(dtype)`	`similarity` or `dissimilarity`
`s(drange)`	range of measure (most similar to most dissimilar)
`s(binary)`	`binary` if the measure is for binary observations

`cluster measures` stores the following in `r()`:

Macros

`r(generate)`	variable names from the `generate()` option
`r(compare)`	observation numbers from the `compare()` option
`r(dtype)`	`similarity` or `dissimilarity`
`r(distance)`	the name of the (dis)similarity measure
`r(binary)`	`binary` if the measure is for binary observations

Also see

[MV] **cluster** — Introduction to cluster-analysis commands

[MV] **clustermat** — Introduction to clustermat commands

[MV] **cluster programming subroutines** — Add cluster-analysis routines