Description Remarks and examples Also see

Description

Programmers of estimation commands can customize how `estat` works after their commands. If you want to use only the standard `estat` subcommands, `ic`, `summarize`, and `vce`, you do not need to do anything; see [R] **estat**. Stata will automatically handle those cases.

Remarks and examples

Remarks are presented under the following headings:

Standard subcommands

For `estat` to work, your estimation command must be implemented as an e-class program, and it must store its name in `e(cmd)`.

`estat vce` requires that the covariance matrix be stored in `e(V)`, and `estat summarize` requires that the estimation sample be marked by the function `e(sample)`. Both requirements can be met by using `ereturn post` with the `esample()` option in your program; see [P] **ereturn**.

Finally, `estat ic` requires that your program store the final log likelihood in `e(ll)` and the sample size in `e(N)`. If your program also stores the log likelihood of the null (constant only) model in `e(ll_0)`, it will appear in the output of `estat ic`, as well.

Adding subcommands to estat

To add new features (subcommands) to `estat` for use after a particular estimation command, you write a handler, which is nothing more than an ado-file command. The standard is to name the new command *cmd*`_estat`, where *cmd* is the name of the corresponding estimation command. For instance, the handler that provides the special `estat` features after `regress` is named `regress_estat`, and the handler that provides the special features after `pca` is named `pca_estat`.

Next you must let `estat` know about your new handler, which you do by filling in `e(estat_cmd)` in the corresponding estimation command. For example, in the code that implements `pca` is the line

```
ereturn local estat_cmd "pca_estat"
```

Finally, you must write *cmd*`_estat`. The syntax of `estat` is

```
estat subcmd ...
```

When the `estat` command is invoked, the first and only thing it does is call `‘e(estat_cmd)’` if `‘e(estat_cmd)’` exists. This way, your handler can even do something special in the standard cases, if that is necessary. We will get to that, but in the meantime, understand that the handler receives just what `estat` received, which is exactly what the user typed. The outline for a handler is

begin *cmd*_estat.ado

```
program cmd_estat, rclass
                version 19.5      // (or version 19 if you do not have StataNow)
                if "`e(cmd)'" != "cmd" {
                        error 301
                }
                gettoken subcmd rest : 0, parse(" ,")
                if "`subcmd'"=="first_special_subcmd" {
                        First_special_subcmd `rest'
                }
                else if "`subcmd'"=="second_special_subcmd" {
                        Second_special_subcmd `rest'
                }
                ...
                else {
                        estat_default `0'
                }
                return add
        end
        program First_special_subcmd, rclass
                syntax ...
                ...
        end
        program Second_special_subcmd, rclass
                syntax ...
                ...
end
```

end *cmd*_estat.ado

The ideas underlying the above outline are simple:

1. You check that `e(cmd)` matches *cmd*.
2. You isolate the *subcmd* that the user typed and then see if it is one of the special cases that you wish to handle.
3. If *subcmd* is a special case, you call the code you wrote to handle it.
4. If *subcmd* is not a special case, you let Stata's `estat_default` handle it.

When you check for the special cases, those special cases can be new *subcmd*s that you wish to add, or they can be standard *subcmd*s whose default behavior you wish to override.

▷ Example 1

Suppose that we have written the estimation command `myreg` and want the `estat` subcommands `fit` and `sens` to work after it, in addition to the standard subcommands. Moreover, we want to be able to abbreviate `sens` as `se` or `sen`. The following code fragment illustrates the structure of our `myreg_estat` handler program:

begin myreg_estat.ado

```
program myreg_estat, rclass
        version 19.5      // (or version 19 if you do not have StataNow)
        gettoken subcmd rest : 0 , parse(", ")
        local lsubcmd= length("`subcmd'")
        if "`subcmd'" == "fit" {
                Fit `rest'
        }
        else if "`subcmd'" == substr("sens",1,max(2,`lsubcmd')) {
                Sens `rest'
        }
        else {
                estat_default `0'
        }
        return add
end
program Fit, rclass
        syntax ...
        ...
end
program Sens, rclass
        syntax ...
        ...
end
```

end myreg_estat.ado

Say that we issue the command

```
estat sen, myoption("Circus peanuts")
```

The only way that the above differs from the standard outline is the complication we added to handle the abbreviation of *subcmd* `sens`. Rather than asking if `"`subcmd'"=="sens"`, we asked if `"`subcmd'"==substr("sens",1,max(2,`lsubcmd'))`, where `lsubcmd` was previously filled in with `length("`subcmd'")`.

◁

Overriding standard behavior of a subcommand

Occasionally, you may want to override the behavior of a subcommand normally handled by `estat_default`. This is accomplished by providing a local handler. Consider, for example, `summarize` after `pca`. The standard way of invoking `estat summarize` is not appropriate here—`estat summarize` extracts the list of variables to be summarized from `e(b)`. This does not work after `pca`. Here the varlist has to be extracted from the column names of the correlation or covariance matrix `e(C)`. This varlist is transferred to `estat summarize` (or more directly to `estat_summ`) as the argument of the standard `estat_summ` program.

```
program Summarize
        syntax [, *]
        tempname C
        matrix `C' = e(C)
        estat_summ `:colnames `C'', `options'
end
```

You add the local handler by inserting an additional switch in *cmd*`_estat` to ensure that the `summarize` subcommand is not handled by the default handler `estat_default`. As a detail, we have to make sure that the minimal abbreviation is `summarize`.

begin pca_estat.ado

```
program pca_estat, rclass
        version 19.5      // (or version 19 if you do not have StataNow)
        gettoken subcmd rest : 0 , parse(", ")
        local lsubcmd= length("`subcmd'")
        if `"`subcmd'"' == substr("summarize", 1, max(2, `lsubcmd')) {
                Summarize `rest'
        }
        else {
                estat_default `0'
        }
        return add
end
program Summarize
        syntax ...
        ...
end
```

end pca_estat.ado

Also see

[R] **estat** — Postestimation statistics