The Stata Journal (2011)
11, Number 2, pp. 321–322

Stata tip 99: Taking extra care with encode

Clyde Schechter
Albert Einstein College of Medicine
Yeshiva University
New York, NY
clyde.schechter@einstein.yu.edu

`encode` (see [D] **encode**) has long been one of Stata's basic data-management commands. `encode` maps the distinct strings of a string variable to an integer-valued numeric variable for which the strings become value labels. Unless you specify a preexisting set of value labels through its `label()` option, `encode` uses the alphanumeric order of distinct string values present in the dataset to determine numeric values 1, 2, 3, and so on. Thus if `"a"`, `"b"`, and `"d"` were the distinct values of a variable, `svar`, in one dataset, then typing

```
. encode svar, generate(nvar)
```

would produce `nvar`, in which 1, 2, and 3 correspond to `"a"`, `"b"`, and `"d"`, respectively. However, the same command applied to a dataset in which `"a"`, `"b"`, `"c"`, and `"d"` were the distinct values of `svar` would produce an encoding that was overlapping but also different: 3 would correspond to `"c"` and 4 to `"d"`. Because value labels are what the user sees in text and graphic output, it could be easy to miss the difference on casual inspection. Moreover, this difference could easily prove problematic if two or more datasets were to be combined, say, by using `append` or `merge`. Indeed, using `encode` on the same variable in multiple datasets that will later be combined can only be called dangerous.

Having been bitten by this many times, I have developed some precautionary data-management practices that I commend to others.

1. There are certain types of variables that recur frequently in my work. For many of these variables, I have developed a standard encoding that I always use. The code to create standard value labels is explicit in some do-files that I routinely `do`, `run`, or `include` in my dataset creation do-files. These value labels cover all the possible values these variables can take. Whenever I `encode` one of these variables, I always explicitly use the `label()` option with these labels.

2. In large projects that will involve multiple datasets with overlapping variables not part of my "standard" list, whenever I use `encode`, I routinely follow up with a `label save` as an audit of that particular encoding. In later work with the same variable in other datasets, before I `encode`, I again `do`, `run`, or `include` the corresponding labeling do-file and then use the explicit `label()` option in the `encode` command. If `encode` finds new levels of the variable not already in the label, it adds them to the label. I follow up using `label save, replace` again so that my labeler do-file remains up to date.

dm0057

3. So that I do not rely on my memory to know whether I have previously developed a labeling for a variable, my practice for nonroutine variables is to give the value label the same name as the variable and to name the labeler do-file using the form *varname*`_label.do`. Then, when I want to `encode` such a variable, I precede the `encode` with

   ```
   . capture run varname_label.do
   ```

   In fact, I have an ado-file that is a wrapper for `encode`—it handles all this for me.

Although these practices may seem cumbersome and can lead to a project directory being a bit cluttered with do-files that just generate labels, adherence to these practices has saved me from some nasty analysis errors that are hard to root out otherwise.