Title

stata.com

encode — Encode string into numeric and vice versa

Description

`encode` creates a new variable named *newvar* based on the string variable *varname*, creating, adding to, or just using (as necessary) the value label *newvar* or, if specified, *name*. *Do not* use `encode` if *varname* contains numbers that merely happen to be stored as strings; instead, use `generate` *newvar* `=real(`*varname*`)` or `destring`; see [U] **24.2 Categorical string variables**, [FN] **String functions**, and [D] **destring**.

`decode` creates a new string variable named *newvar* based on the "encoded" numeric variable *varname* and its value label.

Quick start

Generate numeric `newv1` from string `v1`, using the values of `v1` to create a value label that is applied to `newv1`

```
encode v1, generate(newv1)
```

Same as above, but name the value label `mylabel1`

```
encode v1, generate(newv1) label(mylabel1)
```

Same as above, but refuse to encode `v1` if values exist in `v1` that are not present in preexisting value label `mylabel1`

```
encode v1, generate(newv1) label(mylabel1) noextend
```

Convert numeric `v2` to string `newv2` using the value label applied to `v2` to generate values of `newv2`

```
decode v2, generate(newv2)
```

Menu

encode

Data > Create or change data > Other variable-transformation commands > Encode value labels from string variable

decode

Data > Create or change data > Other variable-transformation commands > Decode strings from labeled numeric variable

Syntax

String variable to numeric variable

```
encode varname [if] [in], generate(newvar) [label(name) noextend]
```

Numeric variable to string variable

```
decode varname [if] [in], generate(newvar) [maxlength(#)]
```

Options for encode

`generate(`*newvar*`)` is required and specifies the name of the variable to be created.

`label(`*name*`)` specifies the name of the value label to be created or used and added to if the named value label already exists. If `label()` is not specified, `encode` uses the same name for the label as it does for the new variable.

`noextend` specifies that *varname* not be encoded if there are values contained in *varname* that are not present in `label(`*name*`)`. By default, any values not present in `label(`*name*`)` will be added to that label.

Options for decode

`generate(`*newvar*`)` is required and specifies the name of the variable to be created.

`maxlength(#)` specifies how many bytes of the value label to retain; # must be between 1 and 32,000. The default is `maxlength(32000)`.

Remarks and examples

stata.com

Remarks are presented under the following headings:

encode

`encode` is most useful in making string variables accessible to Stata's statistical routines, most of which can work only with numeric variables. `encode` is also useful in reducing the size of a dataset. If you are not familiar with value labels, read **[U] 12.6.3 Value labels**.

The maximum number of associations within each value label is 65,536. Each association in a value label maps a string of up to 32,000 bytes to a number. For plain ASCII text, the number of bytes is equal to the number of characters. If your string has other Unicode characters, the number of bytes is greater than the number of characters. See **[U] 12.4.2 Handling Unicode strings**. If your variable contains string values longer than 32,000 bytes, then only the first 32,000 bytes are retained and assigned as a value label to a number.

Example 1

We have a dataset on high blood pressure, and among the variables is sex, a string variable containing either "male" or "female". We wish to run a regression of high blood pressure on race, sex, and age group. We type regress hbp race sex age_grp and get the message "no observations".

```
. use https://www.stata-press.com/data/r18/hbp2
. regress hbp sex race age_grp
no observations
r(2000);
```

Stata's statistical procedures cannot directly deal with string variables; as far as they are concerned, all observations on sex are missing. encode provides the solution:

```
. encode sex, gen(gender)
. regress hbp gender race age_grp
```

Source	SS	df	MS	Number of obs	=	1,121
				F(3, 1117)	=	15.15
Model	2.01013476	3	.67004492	Prob > F	=	0.0000
Residual	49.3886164	1,117	.044215413	R-squared	=	0.0391
				Adj R-squared	=	0.0365
Total	51.3987511	1,120	.045891742	Root MSE	=	.21027

hbp	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
gender	.0394747	.0130022	3.04	0.002	.0139633	.0649861
race	-.0409453	.0113721	-3.60	0.000	-.0632584	-.0186322
age_grp	.0241484	.00624	3.87	0.000	.0119049	.0363919
_cons	-.016815	.0389167	-0.43	0.666	-.093173	.059543

encode looks at a string variable and makes an internal table of all the values it takes on, here "male" and "female". It then alphabetizes that list and assigns numeric codes to each entry. Thus 1 becomes "female" and 2 becomes "male". It creates a new int variable (gender) and substitutes a 1 where sex is "female", a 2 where sex is "male", and a *missing* (.) where sex is *null* (""). It creates a value label (also named gender) that records the mapping 1 ↔ female and 2 ↔ male. Finally, encode labels the values of the new variable with the value label.

◁

Example 2

It is difficult to distinguish the result of encode from the original string variable. For instance, in our last two examples, we typed encode sex, gen(gender). Let's compare the two variables:

```
. list sex gender in 1/4
```

	sex	gender
1.	female	female
2.		.
3.	male	male
4.	male	male

They look almost identical, although you should notice the missing value for gender in the second observation.

The difference does show, however, if we tell `list` to ignore the value labels and show how the data really appear:

```
. list sex gender in 1/4, nolabel
```

	sex	gender
1.	female	1
2.		.
3.	male	2
4.	male	2

We could also ask to see the underlying value label:

```
. label list gender
gender:
           1 female
           2 male
```

`gender` really is a numeric variable, but because *all* Stata commands understand value labels, the variable displays as "male" and "female", just as the underlying string variable `sex` would.

◁

▷ Example 3

We can drastically reduce the size of our dataset by encoding strings and then discarding the underlying string variable. We have a string variable, `sex`, that records each person's sex as "male" and "female". Because female has six characters, the variable is stored as a `str6`.

We can `encode` the `sex` variable and use `compress` to store the variable as a `byte`, which takes only 1 byte. Because our dataset contains 1,130 people, the string variable takes 6,780 bytes, but the encoded variable will take only 1,130 bytes.

```
. use https://www.stata-press.com/data/r18/hbp2, clear
. describe
Contains data from https://www.stata-press.com/data/r18/hbp2.dta
 Observations:         1,130
    Variables:             7                  3 Mar 2022 06:47
```

Variable name	Storage type	Display format	Value label	Variable label
id	str10	%10s		Record identification number
city	byte	%8.0g		City
year	int	%8.0g		Year
age_grp	byte	%8.0g	agefmt	Age group
race	byte	%8.0g	racefmt	Race
hbp	byte	%8.0g	yn	High blood pressure
sex	str6	%9s		Sex

```
Sorted by:
. encode sex, generate(gender)
```

```
. list sex gender in 1/5

     +-----------------+
     |    sex   gender |
     |-----------------|
  1. | female   female |
  2. |              .  |
  3. |   male     male |
  4. |   male     male |
  5. | female   female |
     +-----------------+

. drop sex
. rename gender sex
. compress
  variable sex was long now byte
  (3,390 bytes saved)
. describe
Contains data from https://www.stata-press.com/data/r18/hbp2.dta
 Observations:         1,130
    Variables:             7                  3 Mar 2022 06:47
-------------------------------------------------------------------------------
Variable      Storage   Display    Value
    name         type    format    label      Variable label
-------------------------------------------------------------------------------
id              str10   %10s                  Record identification number
city            byte    %8.0g                 City
year            int     %8.0g                 Year
age_grp         byte    %8.0g      agefmt     Age group
race            byte    %8.0g      racefmt    Race
hbp             byte    %8.0g      yn         High blood pressure
sex             byte    %8.0g      gender     Sex
-------------------------------------------------------------------------------
Sorted by:
     Note: Dataset has changed since last saved.
```

The size of our dataset has fallen from 24,860 bytes to 19,210 bytes.

◁

❑ Technical note

In the examples above, the value label did not exist before `encode` created it, because that is not required. If the value label does exist, `encode` uses your encoding as far as it can and adds new mappings for anything not found in your value label. For instance, if you wanted "`female`" to be encoded as 0 rather than 1 (possibly for use in linear regression), you could type

```
. label define gender 0 "female"
. encode sex, gen(gender)
```

You can also specify the name of the value label. If you do not, the value label is assumed to have the same name as the newly created variable. For instance,

```
. label define sexlbl 0 "female"
. encode sex, gen(gender) label(sexlbl)
```

❑

decode

`decode` is used to convert numeric variables with associated value labels into true string variables.

▷ Example 4

We have a numeric variable named `female` that records the values 0 and 1. `female` is associated with a value label named `sexlbl` that says that 0 means male and 1 means female:

```
. use https://www.stata-press.com/data/r18/hbp3, clear
. describe female
Variable      Storage   Display    Value
    name         type    format    label      Variable label
------------------------------------------------------------------------------
female          byte    %8.0g      sexlbl     Female
. label list sexlbl
sexlbl:
           0 Male
           1 Female
```

We see that `female` is stored as a `byte`. It is a numeric variable. Nevertheless, it has an associated value label describing what the numeric codes mean, so if we `tabulate` the variable, for instance, it appears to contain the strings "male" and "female":

```
. tabulate female
     Female |      Freq.     Percent        Cum.
------------+-----------------------------------
       Male |        695       61.61       61.61
     Female |        433       38.39      100.00
------------+-----------------------------------
      Total |      1,128      100.00
```

We can create a real string variable from this numerically encoded variable by using `decode`:

```
. decode female, gen(sex)
. describe sex
Variable      Storage   Display    Value
    name         type    format    label      Variable label
------------------------------------------------------------------------------
sex             str6    %9s                   Female
```

We have a new variable called `sex`. It is a string, and Stata automatically created the shortest possible string. The word "female" has six characters, so our new variable is a `str6`. `female` and `sex` appear indistinguishable:

```
. list female sex in 1/4
     +---------------+
     | female      sex |
     |---------------|
  1. | Female   Female |
  2. |      .          |
  3. |   Male     Male |
  4. |   Male     Male |
     +---------------+
```

But when we add `nolabel`, the difference is apparent:

```
. list female sex in 1/4, nolabel
```

	female	sex
1.	1	Female
2.	.	
3.	0	Male
4.	0	Male

◁

▷ Example 5

`decode` is most useful in instances when we wish to match-merge two datasets on a variable that has been encoded inconsistently.

For instance, we have two datasets on individual states in which one of the variables (`state`) takes on values such as "CA" and "NY". The state variable was originally a string, but along the way the variable was encoded into an integer with a corresponding value label in one or both datasets.

We wish to merge these two datasets, but either 1) one of the datasets has a string variable for state and the other an encoded variable or 2) although both are numeric, we are not certain that the codings are consistent. Perhaps "CA" has been coded 5 in one dataset and 6 in another.

Because `decode` will take an encoded variable and turn it back into a string, `decode` provides the solution:

```
use first                       (load the first dataset)
decode state, gen(st)           (make a string state variable)
drop state                      (discard the encoded variable)
sort st                         (sort on string)
save first, replace             (save the dataset)
use second                      (load the second dataset)
decode state, gen(st)           (make a string variable)
drop state                      (discard the encoded variable)
sort st                         (sort on string)
merge 1:1 st using first        (merge the data)
```

◁

Video example

How to convert categorical string variables to labeled numeric variables

References

Cox, N. J., and C. B. Schechter. 2018. Speaking Stata: Seven steps for vexatious string variables. *Stata Journal* 18: 981–994.

Schechter, C. B. 2011. Stata tip 99: Taking extra care with encode. *Stata Journal* 11: 321–322.

Also see

[D] **compress** — Compress data in memory

[D] **destring** — Convert string variables to numeric variables and vice versa

[D] **generate** — Create or change contents of variable

[U] 12.6.3 Value labels

[U] 24.2 Categorical string variables