Tracy Garnar

2/12/19

Data Cleaning and Basic Data Manipulation

This Community Resource builds upon previous community resources prepared by Karina Salazar. This will cover the steps one should take to appropriately clean and verify their data, as well as creating several kinds of variables that one often needs for their analysis and discussing some common mistakes people make when creating new variables.

Data Cleaning

Even if we download the GSS or another commonly available dataset from the internet, or receive it from another researcher, we should take steps to verify that the dataset is not corrupt and contains all of the information we need. Furthermore, there will almost always be a need to create new variables in order to produce the analyses we need for our work. I previously covered importing and verifying imported datasets, so will not do so here.

The first principle to keep in mind is that you never want to work on the original dataset. ALWAYS make a copy of your original dataset and keep it in a safe place. Also, right away in your do-file, once you open it, save it with a new name. You will also want to add metadata to the dataset as discussed earlier.

Four Principles for Safe Variable Creation

Long (2009:242) outlined four principles for creating new variables that you should always follow to ensure maximum accuracy:

1. New variables always get new names.
2. Always double-check that you constructed your new variables properly.
3. Always document your new variables with notes and labels.
4. Don't delete the source variables once you have created new ones.

Ensuring that new variables always receive new names is the easiest way to ensure that you are a) not using the incorrectly configured variable, and b) not overwriting the incorrectly configured variable (which could present a problem if you later decide you need the source variable configured differently or want to create another variable out of it).

Once you create a new variable, you should ALWAYS double-check that it was constructed correctly. In particular, missing values can produce unexpected results in analyses using newly created variables. Therefore, if there are missing data, you need to be extra careful to account for it properly in the newly created variable. There are several ways to do this; we will cover some of the most common ways below.

Once you create a new variable, you should always leave copious documentation regarding how you created the variable and for what reason. This information is a lifesaver if you need to go back after the fact and reconfigure the source variable differently or if someone later has questions about it. You should also make sure you label the variable and variable values for the newly created variable as well.

Once you have created your new variables, you might be tempted to drop the source variables from your dataset. However, what happens if you discover six months from now that you need to reconfigure the source variable differently? MOST of the time, it is much safer and will allow you to work more efficiently if you keep the source variables. There are a few rare exceptions, though:

1. If the dataset is extremely large and size is a concern
2. If the source variable had errors that you corrected in the newly created variables. However, you should still consider keeping the source variable, and changing the variable label to something like "source variable – DO NOT USE" with a note affixed explaining the issue.

Variable Manipulation

Very rarely will you find that data are configured just the way you need it for the analyses you want to do. For instance, the dataset I'm working with here needs to have the circuits combined into groups reflecting their general political orientation, and ages and years in office need to be calculated for judges. I would also like to create dummy variables to use in regressions.

Stata offers several core commands to help you create new variables. These commands are `generate` (or `gen`), `egen` (and extensions provided through the ado file `egenmore`), `clonevar`, `replace`, and `recode`. These commands each have a variety of options and features, and these options and features can be combined in various ways to allow a great deal of flexibility in variable creation. I'll concentrate most here on egen, since Karina has already covered the other four in great detail.

Generally speaking, you'll go through a three-step process every time you create a variable:

1. Get to know the original variable and how it's structured – is it a categorical or continuous variable? How is it stored (string or numeric? What precision?) What are possible values? How is it labeled, and will you want to keep the labels for your new variable or relabel it?
2. Create your new variable
3. Double-check to make sure you constructed your new variable correctly

Generate

The core command to create a new variable is `generate` (or `gen`). This command uses the following syntax:

`generate` *newvar* = *exp* [*if*] [*in*]

where *newvar* is the name you desire for your new variable, and *exp* references the Stata expression you would like to have performed to create the new variable. *If* and *in* are optional conditions that allow you to specify if you would like only a certain subset of observations from the original variable included in the new variable. If you include the *if* or *in* option, anything that doesn't fit the specified conditions will be coded as missing in the new variable. You can also abbreviate `generate` to `gen` (or even `g`) though this will make your code harder to read.

Unlike `clonevar` (discussed below), creating a new command using `generate` will not copy variable or value labels to the new variable.

Clonevar

`Clonevar` creates a duplicate of an existing variable, copying over not only the data from the original variable, but also the storage type, variable label, and value labels. This is helpful if you want to create a temporary variable (a variable that you'll only use as an intermediate step towards creating a new variable that you'll use in your analyses). The syntax is as follows:

`clonevar` *newvar* = *exp* [*if*] [*in*]

<u>Egen (and egenmore)</u>

`Egen` stands for "extended generate," and allows you much more flexibility in creating variables than is available using `generate`. Even more options are available in the ado file `egenmore`. There are too many options to give all of them due justice, but I'll cover just a few of the most commonly used options. To get information on everything `egen` (and `egenmore`) can do for you, type `help egen` and `help egenmore` into the Command window in Stata.

`Egen` uses the following syntax:

`egen` [*type*] *newvar* = *function*(*arguments*) [*if*] [*in*] [, *options*]

Type allows you to (optionally) specify how Stata is to store the new variable. The function (along with arguments) tells Stata how to calculate the values of the new variable.

Cut

The *cut* function allows you to generate a new ordinal variable based on cutpoints you specify with respect to the original variable. You could do this using `generate` and a series of `replace` commands, but this is a cleaner, more efficient, and less error-prone method. The command will take the following form:

`egen` *newvar* = *cut(oldvar), {at(#,#,...,#)* or *group(#)} [icodes* or *label]*

The numbers in parentheses within the *at* option specifies the lowest bound for the cutpoints. You will need to specify either 0 or your minimum value as the first cutpoint; otherwise you'll find that your lowest values may not make it in to your new variable as a valid value! Similarly, if you don't set up a cutpoint for your highest value higher than the maximum in your original variable, anything higher than the highest cutpoint will be classified as missing in your new variable.

You can use either the *icodes* option or the label option to specify how you want the values of the new variable reported. *Icodes* will set up integer value labels (0, 1, 2,….). *Label* will set up labels with your specified cutpoints.

Example:

I would like to use my previously created variable, judge_age_dec, to create a categorical variable denoting whether the judge making the decision was young, middle-aged, or old when they made the decision. First, let's review how judge_age_dec is structured.

```
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
judge_age_~c |      1,245    63.41205     9.93563         41         93

.
. codebook judge_age_dec

-----------------------------------------------------------------------------
judge_age_dec                                  judge's age as of date of decision
-----------------------------------------------------------------------------

                  type:  numeric (float)

                 range:  [41,93]                      units:  1
         unique values:  52                       missing .:  0/1,245

                  mean:    63.412
             std. dev:   9.93563

           percentiles:        10%       25%       50%       75%       90%
                                51        56        62        70        78
```

We can generate a variable using cutpoints as follows:

```
.
. egen judge_age_cat = cut(judge_age_dec), at(40, 65, 75) icodes
(207 missing values generated)
```

There shouldn't be any missing values. Let's see what this did….

```
  judge's |
 age as of|
   date of|          judge_age_cat
  decision|         0          1          . |     Total
----------+---------------------------------+----------
        41|         1          0          0 |         1
        42|         1          0          0 |         1
        43|         3          0          0 |         3
        44|         5          0          0 |         5
```

(some output omitted)

```
        73|         0         19          0 |        19
        74|         0         23          0 |        23
        75|         0          0         25 |        25
        76|         0          0         34 |        34
        77|         0          0         23 |        23
        78|         0          0         18 |        18
        79|         0          0         31 |        31
        80|         0          0         21 |        21
        81|         0          0         12 |        12
        82|         0          0         11 |        11
        83|         0          0          4 |         4
        84|         0          0          4 |         4
        85|         0          0          2 |         2
        86|         0          0          3 |         3
        87|         0          0          2 |         2
        88|         0          0          4 |         4
        89|         0          0          8 |         8
        90|         0          0          2 |         2
        91|         0          0          2 |         2
        93|         0          0          1 |         1
----------+---------------------------------+----------
     Total|       733        305        207 |     1,245

.
```

It looks like anyone 75 and up was classified as missing. Looking back at the code, I didn't specify a higher bound cutpoint, so I need to do so.

```
. egen judge_age_cat2 = cut(judge_age_dec), at(40, 65, 75, 95) icodes

.
```

Looks like there might not be any missing data. Let's doublecheck ourselves again to make sure.

```
   judge's │
 age as of │ categorical - judge's age as of
   date of │        date of decision
  decision │     young  middle-ag        old │     Total
───────────┼─────────────────────────────────┼──────────
        41 │         1          0          0 │         1 
        42 │         1          0          0 │         1 
        43 │         3          0          0 │         3 
        44 │         5          0          0 │         5 
```

(some output omitted)

```
        74 │         0         23          0 │        23 
        75 │         0          0         25 │        25 
        76 │         0          0         34 │        34 
        77 │         0          0         23 │        23 
        78 │         0          0         18 │        18 
        79 │         0          0         31 │        31 
        80 │         0          0         21 │        21 
        81 │         0          0         12 │        12 
        82 │         0          0         11 │        11 
        83 │         0          0          4 │         4 
        84 │         0          0          4 │         4 
        85 │         0          0          2 │         2 
        86 │         0          0          3 │         3 
        87 │         0          0          2 │         2 
        88 │         0          0          4 │         4 
        89 │         0          0          8 │         8 
        90 │         0          0          2 │         2 
        91 │         0          0          2 │         2 
        93 │         0          0          1 │         1 
───────────┼─────────────────────────────────┼──────────
     Total │       733        305        207 │     1,245 
```

Looking at the top row, the ones that were previously classified as missing are now classified as 2, which is what we want. Now let's go ahead and label the categories of the new variable with descriptive labels.

```
──────────────────────────────────────────────────────────────────────────────
judge_age_cat2             categorical - judge's age as of date of decision
──────────────────────────────────────────────────────────────────────────────

                  type:  numeric (float)
                 label:  judge_age_cat2_value

                 range:  [0,2]                        units:  1
         unique values:  3                        missing .:  0/1,245

            tabulation:  Freq.   Numeric  Label
                           733         0  young
                           305         1  middle-aged
                           207         2  old
```

Using the *group(#)* option allows you to set up some specified number of equally-sized (in terms of number of observations, based on a frequency distribution) groups.

```
. egen judge_age_cat3 = cut(judge_age_dec), group(4) icodes

.
```

Let’s confirm our work…

```
   judge's |
 age as of |   categorical based on freq dist - judge's
   date of |          age as of date of decision
  decision |  55 or you      56-61      62-69  70 and ol |     Total
-----------+--------------------------------------------+----------
        41 |         1          0          0          0 |         1 
        42 |         1          0          0          0 |         1 
        43 |         3          0          0          0 |         3 
        44 |         5          0          0          0 |         5 
        45 |        10          0          0          0 |        10 
        46 |         8          0          0          0 |         8 
        47 |        13          0          0          0 |        13 
        48 |        24          0          0          0 |        24 
        49 |        21          0          0          0 |        21 
        50 |        31          0          0          0 |        31 
        51 |        35          0          0          0 |        35 
        52 |        34          0          0          0 |        34 
        53 |        29          0          0          0 |        29 
        54 |        35          0          0          0 |        35 
        55 |        43          0          0          0 |        43 
        56 |         0         36          0          0 |        36 
        57 |         0         49          0          0 |        49 
        58 |         0         37          0          0 |        37 
        59 |         0         44          0          0 |        44 
        60 |         0         42          0          0 |        42 
        61 |         0         53          0          0 |        53 
        62 |         0          0         74          0 |        74 
        63 |         0          0         55          0 |        55 
        64 |         0          0         50          0 |        50 
        65 |         0          0         42          0 |        42 
        66 |         0          0         40          0 |        40 
        67 |         0          0         33          0 |        33 
        68 |         0          0         31          0 |        31 
        69 |         0          0         29          0 |        29 
        70 |         0          0          0         29 |        29 
        71 |         0          0          0         29 |        29 
        72 |         0          0          0         30 |        30 
        73 |         0          0          0         19 |        19 
        74 |         0          0          0         23 |        23 
        75 |         0          0          0         25 |        25 
        76 |         0          0          0         34 |        34 
        77 |         0          0          0         23 |        23 
        78 |         0          0          0         18 |        18 
        79 |         0          0          0         31 |        31 
        80 |         0          0          0         21 |        21 
        81 |         0          0          0         12 |        12 
        82 |         0          0          0         11 |        11 
        83 |         0          0          0          4 |         4 
        84 |         0          0          0          4 |         4 
        85 |         0          0          0          2 |         2 
        86 |         0          0          0          3 |         3 
        87 |         0          0          0          2 |         2 
        88 |         0          0          0          4 |         4 
        89 |         0          0          0          8 |         8 
        90 |         0          0          0          2 |         2 
        91 |         0          0          0          2 |         2 
        93 |         0          0          0          1 |         1 
-----------+--------------------------------------------+----------
     Total |       293        261        354        337 |     1,245 
```

Now we can label our new variable with the cutpoints Stata chose for us based on the frequency distribution.

```
-------------------------------------------------------------------------------
judge_age_cat3                        categorical based on freq dist -
                                      judge's age as of date of decision
-------------------------------------------------------------------------------

                  type:  numeric (float)
                 label:  judge_age_cat3_value

                 range:  [0,3]                        units:  1
         unique values:  4                        missing .:  0/1,245

            tabulation:  Freq.   Numeric  Label
                           293         0  55 or younger
                           261         1  56-61
                           354         2  62-69
                           337         3  70 and older
```

One advantage of using the *group(#)* option is that because Stata will base its cutpoints on the frequency distribution, you can ensure that all of your categories will have roughly equal numbers of observations, which is important for certain statistical procedures. Also, you can ensure that each group is both exhaustive and mutually exclusive, meaning that you’re not leaving any values out or having any observations qualify for two overlapping categories.

On the other hand, you might want to have each category cover an equal amount of “ground” regardless of how many observations fall into each category, which is important for some statistical procedures. For instance, I might want to specify age ranges of up to 50, 51-60, 61-70, and so forth. I can do so using the *a(b)c* syntax within the *at* option. Example:

```
. egen judge_age_cat4 = cut(judge_age_dec), at(40(10)100) icodes

.
```

Let’s see what it did.

```
  judge's |
age as of |
  date of |                      judge_age_cat4
 decision |         0          1          2          3          4          5 |     Total
----------+------------------------------------------------------------------+----------
       41 |         1          0          0          0          0          0 |         1
       42 |         1          0          0          0          0          0 |         1
       43 |         3          0          0          0          0          0 |         3
       44 |         5          0          0          0          0          0 |         5
       45 |        10          0          0          0          0          0 |        10
       46 |         8          0          0          0          0          0 |         8
       47 |        13          0          0          0          0          0 |        13
       48 |        24          0          0          0          0          0 |        24
       49 |        21          0          0          0          0          0 |        21
       50 |         0         31          0          0          0          0 |        31
       51 |         0         35          0          0          0          0 |        35
       52 |         0         34          0          0          0          0 |        34
       53 |         0         29          0          0          0          0 |        29
       54 |         0         35          0          0          0          0 |        35
       55 |         0         43          0          0          0          0 |        43
       56 |         0         36          0          0          0          0 |        36
       57 |         0         49          0          0          0          0 |        49
       58 |         0         37          0          0          0          0 |        37
       59 |         0         44          0          0          0          0 |        44
       60 |         0          0         42          0          0          0 |        42
       61 |         0          0         53          0          0          0 |        53
       62 |         0          0         74          0          0          0 |        74
       63 |         0          0         55          0          0          0 |        55
       64 |         0          0         50          0          0          0 |        50
       65 |         0          0         42          0          0          0 |        42
       66 |         0          0         40          0          0          0 |        40
       67 |         0          0         33          0          0          0 |        33
       68 |         0          0         31          0          0          0 |        31
       69 |         0          0         29          0          0          0 |        29
       70 |         0          0          0         29          0          0 |        29
       71 |         0          0          0         29          0          0 |        29
       72 |         0          0          0         30          0          0 |        30
       73 |         0          0          0         19          0          0 |        19
       74 |         0          0          0         23          0          0 |        23
       75 |         0          0          0         25          0          0 |        25
       76 |         0          0          0         34          0          0 |        34
       77 |         0          0          0         23          0          0 |        23
       78 |         0          0          0         18          0          0 |        18
       79 |         0          0          0         31          0          0 |        31
       80 |         0          0          0          0         21          0 |        21
       81 |         0          0          0          0         12          0 |        12
       82 |         0          0          0          0         11          0 |        11
       83 |         0          0          0          0          4          0 |         4
       84 |         0          0          0          0          4          0 |         4
       85 |         0          0          0          0          2          0 |         2
       86 |         0          0          0          0          3          0 |         3
       87 |         0          0          0          0          2          0 |         2
       88 |         0          0          0          0          4          0 |         4
       89 |         0          0          0          0          8          0 |         8
       90 |         0          0          0          0          0          2 |         2
       91 |         0          0          0          0          0          2 |         2
       93 |         0          0          0          0          0          1 |         1
----------+------------------------------------------------------------------+----------
    Total |        86        373        449        261         71          5 |     1,245
```

As you can see, Stata created our new variable with 6 categories – one for up to 50 years old, one for 51-60, one for 61-70, one for 71-80, one for 81-90, and one for over 90. Note that the frequency distribution is not equal across all values of the variable, which could cause problems in some statistical procedures. We'll go ahead and label our new variable now.

```
judge_age_cat4                          categorical based on age - judge's age as of date of decision

                  type:  numeric (float)
                 label:  judge_age_cat4_value

                 range:  [0,5]                        units:  1
         unique values:  6                        missing .:  0/1,245

            tabulation:  Freq.   Numeric  Label
                            86         0  50 or younger
                           373         1  51-60
                           449         2  61-70
                           261         3  71-80
                            71         4  81-90
                             5         5  over 90
```

Group

One of the most powerful things `egen` can do for you is make it much easier to create a composite categorical variable using the *group* function. We can use this to create interaction variables out of categorical variables.

This takes the following syntax:

> `egen` *newvar = group(varlist) [, missing label lname(name) truncate(num)]*

Let's walk through a basic example. We'd like to create interaction variables for various combinations of judge's race and judge's sex. First, we'll check how the variables are structured:

```
-> tabulation of gender_judge

Judge's Sex |      Freq.     Percent        Cum.
------------+-----------------------------------
       Male |      1,075       86.35       86.35
     Female |        170       13.65      100.00
------------+-----------------------------------
      Total |      1,245      100.00

-> tabulation of race_judge

    Race of |
      judge |      Freq.     Percent        Cum.
------------+-----------------------------------
      White |      1,123       90.20       90.20
      Black |         75        6.02       96.22
   Hispanic |         41        3.29       99.52
      Asian |          6        0.48      100.00
------------+-----------------------------------
      Total |      1,245      100.00
```

```
────────────────────────────────────────────────────────────────────────────────────────────
gender_judge                                                                      Judge's Sex
────────────────────────────────────────────────────────────────────────────────────────────

                  type:  numeric (byte)
                 label:  gender_judge

                 range:  [0,1]                        units:  1
         unique values:  2                        missing .:  0/1,245

            tabulation:  Freq.   Numeric  Label
                         1,075         0  Male
                           170         1  Female

────────────────────────────────────────────────────────────────────────────────────────────
race_judge                                                                      Race of judge
────────────────────────────────────────────────────────────────────────────────────────────

                  type:  numeric (byte)
                 label:  race_judge

                 range:  [1,4]                        units:  1
         unique values:  4                        missing .:  0/1,245

            tabulation:  Freq.   Numeric  Label
                         1,123         1  White
                            75         2  Black
                            41         3  Hispanic
                             6         4  Asian

.
```

Now let's create our new variable:

```
. egen racesex = group(gender_judge race_judge)

.
```

And let's see what it did:

```
-> tabulation of racesex

group(gende |
    r_judge |
race_judge) |      Freq.     Percent        Cum.
────────────┼───────────────────────────────────
          1 |        974       78.23       78.23
          2 |         58        4.66       82.89
          3 |         37        2.97       85.86
          4 |          6        0.48       86.35
          5 |        149       11.97       98.31
          6 |         17        1.37       99.68
          7 |          4        0.32      100.00
────────────┼───────────────────────────────────
      Total |      1,245      100.00
```

Before we can label our new variable, we need to figure out how it determined which categories to assign to which values. To do this, we can do the following:

```
. table racesex gender_judge race_judge, missing

─────────────────────────────────────────────────────────────────────────────
group(gen │
der_judge │                  Race of judge and Judge's Sex
race_judg │ ─── White ───      ─── Black ───     ── Hispanic ──      ─── Asian ───
e)        │   Male  Female       Male  Female       Male  Female       Male  Female
──────────┼──────────────────────────────────────────────────────────────────
        1 │    974       .          .       .          .       .          .       .
        2 │      .       .         58       .          .       .          .       .
        3 │      .       .          .       .         37       .          .       .
        4 │      .       .          .       .          .       .          6       .
        5 │      .     149          .       .          .       .          .       .
        6 │      .       .          .      17          .       .          .       .
        7 │      .       .          .       .          .       4          .       .
─────────────────────────────────────────────────────────────────────────────
```

Now we've labeled the variable. Note that this code did not produce a group for Asian female, presumably because there aren't any Asian female judges in this dataset. If this is of concern, you may want to go ahead and create the interaction terms manually using generate and replace – be careful here because more lines of code create more opportunities for error.

```
────────────────────────────────────────────────────────────────────────────────────────────────
racesex                                                       interaction - race and sex of judge
────────────────────────────────────────────────────────────────────────────────────────────────

                  type:  numeric (float)
                 label:  racesex_value

                 range:  [1,7]                        units:  1
         unique values:  7                        missing .:  0/1,245

            tabulation:  Freq.   Numeric  Label
                           974         1  white male
                            58         2  Black male
                            37         3  Hispanic male
                             6         4  Asian male
                           149         5  white female
                            17         6  black female
                             4         7  Hispanic female
```

`Egenmore` adds yet more options to the already extensive list of options `egen` gives you for creating variables. `Egenmore` comes in an ado file, so it needs to be installed first. Type `help egenmore` to get an idea of what it can do for you.

<u>rall</u>

We can use *rall* to check to see whether any of a set of variables have a certain value on any of those variables before including them in a composite variable. It looks like this: `egen` *newvar = rall(varlist) , cond(condition) [symbol(symbol)]*

For instance, we can create an indicator variable that checks for missing data on a series of variables for different causes of action before we include them in a composite cause-of-action variable. We could alternatively do this by using tab or codebook. There are over 10 different cause of action variables in our dataset, but for simplicity's sake, we'll just include three in our indicator variable: retaliation, not_hired and fired. Your newly created indicator variable will return 1 if the specified conditions are true for each and every variable in the list, and 0 otherwise.

Once we've seen how the original variables are constructed, we can generate our indicator variable:

```
. egen cause_act_ind = rall(retaliation not_hired fired), cond(@ > 0 & @ < .)

.
```

The *cond* option at the end specifies the conditions under which the new variable will be considered to have a valid value. For instance, *cond(@ > 0 & @ < .)* checks whether each of the variables has a positive, valid (non-missing) value. The "@" sign is the symbol indicating that x should, in this case, be positive and non-missing. If for some strange reason you had "@" as a valid value in your dataset, you'll need to specify another symbol to use in place of "@", which you would do using the *symbol* option.

Let's see how well it worked.

```
     +--------------------------------------+
     | cause_~d   retali~n   not_hi~d   fired |
     |--------------------------------------|
  1. |        0         No         No      No |
  2. |        0        Yes         No      No |
  3. |        0         No         No     Yes |
  4. |        0         No         No     Yes |
  5. |        0        Yes         No     Yes |
     |--------------------------------------|
  6. |        0         No         No      No |
  7. |        0         No         No      No |
  8. |        0         No         No      No |
  9. |        0        Yes         No      No |
 10. |        0         No         No      No |
     |--------------------------------------|
 11. |        0         No         No      No |
 12. |        0        Yes         No      No |
 13. |        0         No         No      No |
 14. |        0         No         No     Yes |
 15. |        0        Yes         No      No |
```

(portion of output omitted)

```
. tab cause_act_ind retaliation

           |   P claims employer
cause_act_ |  acted in retaliation
       ind |        No        Yes |     Total
-----------+----------------------+----------
         0 |       807        438 |     1,245
-----------+----------------------+----------
     Total |       807        438 |     1,245
```

```
. tab cause_act_ind not_hired

           |   P claims being
cause_act_ | illegally not hired
       ind |        No        Yes |     Total
-----------+----------------------+----------
         0 |     1,146         99 |     1,245
-----------+----------------------+----------
     Total |     1,146         99 |     1,245

. tab cause_act_ind fired

           |   P claims being
cause_act_ |   illegally fired
       ind |        No        Yes |     Total
-----------+----------------------+----------
         0 |       747        498 |     1,245
-----------+----------------------+----------
     Total |       747        498 |     1,245

.
```

Since 0 is not considered positive and none of the cases had 1 (yes) for each of the variables included in the dataset (because it's impossible to file a case against the same employer claiming you were both not hired and fired by them!) it will return 0 for every case. We see here that that is indeed the case, and can go ahead and label our variable.

Replace

`Replace` allows you to use an existing variable and replace its values. In keeping with Long's (2009) recommendation, you will almost always want to do this to a newly created variable you created using `generate` or `egen`, rather than replacing the values of the original variable. You can use this procedure to recode a dichotomous variable coded 1-2 in traditional dummy variable format (0-1), for instance.

The syntax is as follows:

```
replace var=#1 if var==#2
```
or
```
replace var=#1 if var2==2 | var2==3
```

where #1 represents the desired value of the variable, and #2 indicates the original value of the variable.

Karina provides detailed examples of the replace command.

Recode

`Recode` allows you to combine values of a continuous variable into a categorical variable. You can also use `recode` as you would use `replace`, to recode dichotomous variables coded 1-2 into a traditional dummy variable format. Again, you'll usually want to do this on a newly created variable, not your source variable. The syntax is as follows:

```
        recode var (rule 1) (rule 2) (more rules as needed...),
gen(newvar)
```

You'll specify how you want the categories combined in the *rule* options. You can specify a one-on-one mapping (e.g., mapping male=1 in the original variable to male=0 in the new variable) by typing "(1=0)". A many-to-one mapping (for example, combining from 13-15 years of education into a single "college" value) by typing "(13/15=1)". The `gen(`*`newvar`*`)` option allows you to combine the new-variable-creation step with the recode step, instead of first generating a new variable and then recoding it.

Again, see Karina's community resource for examples.

Common Problems

I will briefly describe the most common problems people encounter when creating new variables. Karina provides a very detailed explanation along with excellent examples that I recommend you check out, and I won't recreate her work here.

According to Long (2009), the most common problems that people encounter when creating new variables are:

1. Failure to account for missing values
2. Failure to ensure categorical variables are mutually exclusive and exhaustive
3. Overly complicated programming
4. Using "or" when you meant "and", and vice versa

Missing values

Probably the most common mistake people make when creating new variables is failing to account for missing data properly. Recall that Stata treats "." as positive infinity, which can lead to misleading results. Failure to account for missing data also might mean that your estimates are inaccurate. Therefore, it is crucial to make sure you're not only aware of how the original variable records missing data and how much there is in the original variable, but be mindful of how you treat it in the new variable.

To be on the safe side, run `codebook` for each variable you want to reconfigure.

```
--------------------------------------------------------------------------------------------
rep78                                                                    Repair Record 1978
--------------------------------------------------------------------------------------------

                 type:  numeric (int)

                range:  [1,5]                       units:  1
        unique values:  5                       missing .:  5/74

           tabulation:  Freq.  Value
                            2  1
                            8  2
                           30  3
                           18  4
                           11  5
                            5  .
```

If we are using an evaluation involving the greater than sign, we need to exclude any missing data from that calculation. For instance, if we did a calculation in which we wanted anything with a repair history higher than 3, just typing ">3" would produce inaccurate results. Therefore, we would need to type ">3

& !=." to make sure we were truly capturing only cars with repair histories of 4 or 5, and not capturing cars with a missing repair history.

The notation "!=." will only work with standard missing values, though. If you have a complex missing value, you need to use the notation "!missing(varname)" instead.

If running a tab or summarize, always include the ",missing" option at the end.

You may be tempted to drop all cases from the dataset with missing data on a variable or set of variables. However, using the command `drop if missing(var)` will actually drop those observations from your dataset completely, which isn't going to help you if you later want to include those observations in procedures for which there isn't missing data on other variables. It's much safer to use "!=." or "!missing(varname)" instead.

<u>Ensuring categories are mutually exclusive and exhaustive</u>

It is vital to make sure our categories are mutually exclusive and exhaustive. Mutually exclusive categories ensure that each observation can fall into only one category, and exhaustive categories ensure that each observation has a valid category it can fall into, instead of being relegated to missing data. Refer to the example I outlined under "egen cut" to see this in action. Karina also provides detailed examples of coding that is not mutually exclusive and exhaustive.

To fix this, the best way is to use the *group* option (or "a(b)c" notation in the *at* option) in `egen cut` to make sure that you're accounting for all possible categories. To ensure there is no overlap and no missing data in your newly created variable, `tab` is your friend.

<u>Overly complicated programming</u>

Stata chokes if you try to include "&" (the *and* operator) and "|" (the *or* operator) in the same line of code. Karina provides detailed examples of how including both the *and* and the *or* operators in the same line of code can lead to unexpected results.

To fix this, split up your lines of code and create a temporary variable so that you are only using the *and* operator in one line, and only the *or* operator in the other line. You can then use the temporary variable to combine the previous operations in a final variable.

<u>Using "and" when you meant "or", and vice versa</u>

This is related to the previous common error. If you want to create a dummy variable that returns 1 only when an observation = yes for one or two values of a source variable, humans would think that you could use the "or" operator to specify that the dummy variable should return one if either of those conditions were met. However, Stata doesn't think like we do – you actually have to use the "and" operator in this case. See Karina's community resource for a detailed explanation.

You can only really see if this common error is playing a role by doublechecking your coding after generating the new variable.

Works cited
Long, J. Scott. 2009. *The Workflow of Data Analysis Using Stata*. College Station, TX: Stata Press.