11 Creating new variables

generate and replace

This chapter shows the basics of creating and modifying variables in Stata. We saw how to work with the Data Editor in [GSM] **6 Using the Data Editor**—this chapter shows how we would do this from the Command window. The two primary commands used for this are

- `generate` for creating new variables. It has a minimum abbreviation of `g`.
- `replace` for replacing the values of an existing variable. It may not be abbreviated because it alters existing data and hence can be considered dangerous.

The most basic form for creating new variables is `generate` *newvar* `=` *exp*, where *exp* is any kind of *expression*. Of course, both `generate` and `replace` can be used with `if` and `in` qualifiers. An expression is a formula made up of constants, existing variables, operators, and functions. Some examples of expressions (using variables from the `auto` dataset) would be `2 + price`, `weight^2` or `sqrt(gear_ratio)`.

The operators defined in Stata are given in the table below:

Arithmetic		Logical		Relational (numeric and string)	
`+`	addition	`!`	not	`>`	greater than
`-`	subtraction	`|`	or	`<`	less than
`*`	multiplication	`&`	and	`>=`	$>$ or equal
`/`	division			`<=`	$<$ or equal
`^`	power			`==`	equal
				`!=`	not equal
`+`	string concatenation				

Stata has many mathematical, statistical, string, date, time-series, and programming functions. See `help functions` for the basics, and see the *Stata Functions Reference Manual* for a complete list and full details of all the built-in functions.

You can use menus and dialogs to create new variables and modify existing variables by selecting menu items from the **Data > Create or change data** menu. This feature can be handy for finding functions quickly. However, we will use the Command window for the examples in this chapter because we would like to illustrate simple usage and some pitfalls.

Stata has some utility commands for creating new variables:

- The `egen` command is useful for working across groups of variables or within groups of observations. See [D] **egen** for more information.
- The `encode` command turns categorical string variables into encoded numeric variables, while its counterpart `decode` reverses this operation. See [D] **encode** for more information.
- The `destring` command turns string variables that should be numeric, such as numbers with currency symbols, into numbers. To go from numbers to strings, the `tostring` command is useful. See [D] **destring** for more information.

We will focus our efforts on `generate` and `replace`.

generate

There are some details you should know about the `generate` command:

- The basic form of the `generate` command is `generate` *newvar* `=` *exp*, where *newvar* is a new variable name and *exp* is any valid expression. You will get an error message if you try to `generate` a variable that already exists.
- An algebraic calculation using a missing value yields a missing value, as does division by zero, the square root of a negative number, or any other computation which is impossible.
- If missing values are generated, the number of missing values in *newvar* is always reported. If Stata says nothing about missing values, then no missing values were generated.
- You can use `generate` to set the storage type of the new variable as it is generated. You might want to create an indicator (0/1) variable as a `byte`, for example, because it saves 3 bytes per observation over using the default storage type of `float`.

Below are some examples of creating new variables from the `afewcarslab` dataset, which we created in *Labeling values of variables* in [GSM] **9 Labeling data**. (To work along, start by opening the `auto` dataset with `sysuse auto`. We are using a smaller dataset to make shorter listings.) The last example shows a way to generate an indicator variable for cars weighing more than 3,000 pounds. Logical expressions in Stata result in `1` for "true" and `0` for "false". The `if` qualifier is used to ensure that the computations are done only for observations where `weight` is not missing.

```
. use afewcarslab
(A few 1978 cars)
. list make mpg weight
```

	make	mpg	weight
1.	VW Rabbit	25	1930
2.	Olds 98	21	4060
3.	Chev. Monza	.	2750
4.		22	2930
5.	Datsun 510	24	2280
6.	Buick Regal	20	3280
7.	Datsun 810	.	2750

```
. * changing MPG to liters per 100km
. generate lphk = 3.7854 * (100 / 1.6093) / mpg
(2 missing values generated)
. label var lphk "Liters per 100km"
. * getting logarithms of price
. g lnprice = ln(price)
. * making an indicator of hugeness
. gen byte huge = weight >= 3000 if !missing(weight)
. l make mpg weight lphk lnprice huge
```

	make	mpg	weight	lphk	lnprice	huge
1.	VW Rabbit	25	1930	9.408812	8.454679	0
2.	Olds 98	21	4060	11.20097	9.084097	1
3.	Chev. Monza	.	2750	.	8.207129	0
4.		22	2930	10.69183	8.318499	0
5.	Datsun 510	24	2280	9.800845	8.532869	0
6.	Buick Regal	20	3280	11.76101	8.554296	1
7.	Datsun 810	.	2750	.	9.003193	0

replace

Whereas `generate` is used to create new variables, `replace` is the command used for existing variables. Stata uses two different commands to prevent you from accidentally modifying your data. The `replace` command cannot be abbreviated. Stata generally requires you to spell out completely any command that can alter your existing data.

```
. list make weight

     +-------------------+
     |        make   weight |
     |-------------------|
  1. |   VW Rabbit     1930 |
  2. |     Olds 98     4060 |
  3. | Chev. Monza     2750 |
  4. |                 2930 |
  5. |  Datsun 510     2280 |
     |-------------------|
  6. | Buick Regal     3280 |
  7. |  Datsun 810     2750 |
     +-------------------+

. * will give an error because weight already exists
. gen weight = weight/1000
variable weight already defined
r(110);

. * will replace weight in lbs by weight in 1000s of lbs
. replace weight = weight/1000
(7 real changes made)

. list make weight

     +-------------------+
     |        make   weight |
     |-------------------|
  1. |   VW Rabbit     1.93 |
  2. |     Olds 98     4.06 |
  3. | Chev. Monza     2.75 |
  4. |                 2.93 |
  5. |  Datsun 510     2.28 |
     |-------------------|
  6. | Buick Regal     3.28 |
  7. |  Datsun 810     2.75 |
     +-------------------+
```

Suppose that you want to create a new variable, `predprice`, which will be the predicted price of the cars in the following year. You estimate that domestic cars will increase in price by 5% and foreign cars, by 10%.

One way to create the variable would be to first use `generate` to compute the predicted domestic car prices. Then use `replace` to change the missing values for the foreign cars to their proper values.

```
. gen predprice = 1.05*price if foreign==0
(3 missing values generated)
. replace predprice = 1.10*price if foreign==1
(3 real changes made)
. list make foreign price predprice, nolabel
```

	make	foreign	price	predpr~e
1.	VW Rabbit	1	4697	5166.7
2.	Olds 98	0	8814	9254.7
3.	Chev. Monza	0	3667	3850.35
4.		0	4099	4303.95
5.	Datsun 510	1	5079	5586.9
6.	Buick Regal	0	5189	5448.45
7.	Datsun 810	1	8129	8941.9

Of course, because `foreign` is an indicator variable, we could generate the predicted variable with one command:

```
. gen predprice2 = (1.05 + 0.05*foreign)*price
. list make foreign price predprice predprice2, nolabel
```

	make	foreign	price	predpr~e	predpr~2
1.	VW Rabbit	1	4697	5166.7	5166.7
2.	Olds 98	0	8814	9254.7	9254.7
3.	Chev. Monza	0	3667	3850.35	3850.35
4.		0	4099	4303.95	4303.95
5.	Datsun 510	1	5079	5586.9	5586.9
6.	Buick Regal	0	5189	5448.45	5448.45
7.	Datsun 810	1	8129	8941.9	8941.9

generate with string variables

Stata is smart. When you generate a variable and the expression evaluates to a string, Stata creates a string variable with a storage type as long as necessary, and no longer than that. `where` is a `str1` in the following example:

```
. list make foreign

     +-------------------+
     |        make  foreign |
     |-------------------|
  1. |   VW Rabbit  foreign |
  2. |     Olds 98 domestic |
  3. | Chev. Monza domestic |
  4. |             domestic |
  5. |  Datsun 510  foreign |
     |-------------------|
  6. | Buick Regal domestic |
  7. |  Datsun 810  foreign |
     +-------------------+

. gen where = "D" if foreign=="domestic":origin
(3 missing values generated)

. replace where = "F" if foreign=="foreign":origin
(3 real changes made)

. list make foreign where

     +-------------------------+
     |        make  foreign  where |
     |-------------------------|
  1. |   VW Rabbit  foreign      F |
  2. |     Olds 98 domestic      D |
  3. | Chev. Monza domestic      D |
  4. |             domestic      D |
  5. |  Datsun 510  foreign      F |
     |-------------------------|
  6. | Buick Regal domestic      D |
  7. |  Datsun 810  foreign      F |
     +-------------------------+

. describe where
              storage   display    value
variable name   type    format     label      variable label
------------------------------------------------------------------------------
where           str1    %9s
```

Stata has some useful tools for working with string variables. Here we split the `make` variable into make and model and then create a variable that has the model together with where the model was manufactured:

```
. gen model = usubstr(make, ustrpos(make," ")+1,.)
(1 missing value generated)
. gen modelwhere = model + " " + where
. list make where model modelwhere
```

	make	where	model	modelw~e
1.	VW Rabbit	F	Rabbit	Rabbit F
2.	Olds 98	D	98	98 D
3.	Chev. Monza	D	Monza	Monza D
4.		D		D
5.	Datsun 510	F	510	510 F
6.	Buick Regal	D	Regal	Regal D
7.	Datsun 810	F	810	810 F

There are a few things to note about how these commands work:

1. `ustrpos(`s_1`,`s_2`)` produces an integer equal to the first character in the string s_1 at which the string s_2 is found or 0 if it is not found. In this example, `ustrpos(make," ")` finds the position of the first space in each observation of `make`.
2. `usubstr(`s`,`$start$`,`len`)` produces a string of length len characters, beginning at character $start$ of string s. If $c_1 = .$, the result is the string from character $start$ to the end of string s.
3. Putting 1 and 2 together: `usubstr(`s`,ustrpos(`s`," ")+1,.)` will always give the string s with its first word removed. Because `make` contains both the make and the model of each car, and `make` never contains a space in this dataset, we have found each car's model.
4. The operator "`+`", when applied to string variables, will concatenate the strings (that is, join them together). The expression `"this" + "that"` results in the string `"thisthat"`. When the variable `modelwhere` was generated, a space (`" "`) was added between the two strings.
5. The missing value for a string is nothing special—it is simply the empty string `""`. Thus the value of `modelwhere` for the car with no make or model is `" D"` (note the leading space).
6. If your strings might contain Unicode characters, use the Unicode versions of the string functions, as shown above. See [U] **12.4.2 Handling Unicode strings**.