Variables: Types, Generation, Replacing, Labeling, Recoding, Notes

Variables

- The word “variable” can mean many things in programming and statistics, but it has a specific meaning in Stata:
  - A variable in Stata reflects the social scientist’s common definition of a variable, not the programmer’s
  - In databases this might be called a *field,* while in other languages it might be simply a *column*
  - There are “variables” held in memory and available for Stata users, but these remain hidden to the beginning user

Basic Commands

- In order to learn some basic things about variables, we will be using three commands:

`generate`	— generates a new variable
`replace`	— replaces an already existing variable
`describe`	— describes the details of a variable

Generate

```
generate [type] variable_name = expression
```

- This command generates a new variable with type *type* based on a given expression
- The expression can be a constant (e.g. a number 4, or a string “Hello!”) or can be dependent on other variables (e.g. weight/height or age + 5).
- This command will only generate a new variable if that variable name is available

Replace

```
replace variable_name = expression
```

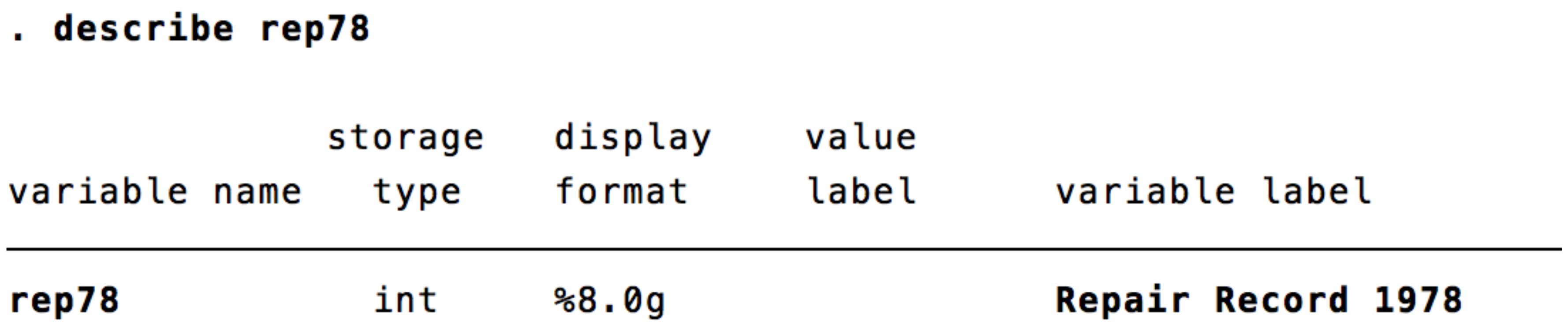
- This command replaces the values of a pre-existing variable based on a given expression
- Just like `generate` this can be a constant or dependent on other variables (e.g. weight/height or age + 5).
- This command will only generate a new variable if that variable name is available

Describe

```
describe

describe variable_name
```

- This command will output a description of the variable, including the variable name, storage type, value label, and variable label



```
. describe rep78

              storage   display    value
variable name   type    format     label      variable label
---------------------------------------------------------------
rep78           int     %8.0g                 Repair Record 1978
```

Variable Types

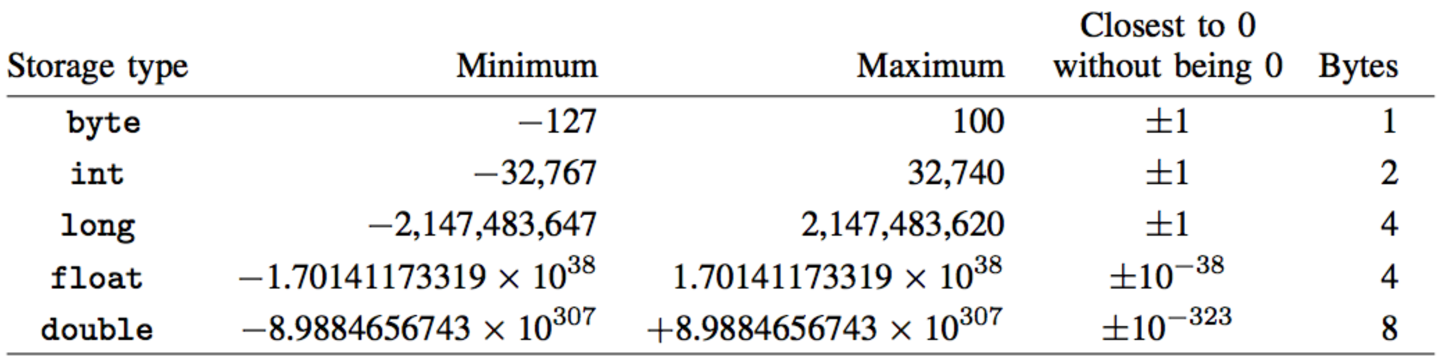
- Every Stata variable will be defined as a particular *type*
  - The *type* of a variable defines what kind of data is expected for each variable (a number or a string of characters)
  - The *type* of a variable also defines what kinds of commands can be performed on a variable, and what kind of results we can expect from these commands
  - Finally, the *storage type* of a variable determines how the variable is stored in memory (more important for advanced users)

Types and Missing

- Number: Height measured in inches
    - A missing number will be represented with a `.`
    - To see entries with missing values, you can use the command `list if missing(variable_name)`
- String: A participant’s name or address
    - Missing strings are represented by the empty string `“”`

Example: Numbers

- Numbers come in 5 storage types:
  - byte
  - int
  - long
  - float
  - double



Storage type	Minimum	Maximum	Closest to 0 without being 0	Bytes
`byte`	-127	100	± 1	1
`int`	$-32{,}767$	$32{,}740$	± 1	2
`long`	$-2{,}147{,}483{,}647$	$2{,}147{,}483{,}620$	± 1	4
`float`	$-1.70141173319 \times 10^{38}$	$1.70141173319 \times 10^{38}$	$\pm 10^{-38}$	4
`double`	$-8.9884656743 \times 10^{307}$	$+8.9884656743 \times 10^{307}$	$\pm 10^{-323}$	8

`if rules and .`

- There is one important Stata idiosyncrasy regarding if qualifiers and missing numbers
  - Behind the scenes, Stata records a missing `.` as a *very large number*
  - As a result, any if rule using a `>` or `>=` will always evaluate to True (1) when a missing is present

```
generate m = .

generate test = 1 if m > 23
```

- Whenever you are using these operators, it makes sense to check beforehand for missing values, or include an additional clause to your rule:

```
generate test = 1 if m > 23 & !missing(m)
```

Number Storage Types

- Storage types are not a big concern for beginning users
  - Stata is very intelligent about default variable type creation, and expanding types when using the `replace` command
  - However, incorrect assignment of types using the `generate` command can result in missing values **without** Stata reporting an error

Keep It Simple!

- For the beginning user, messing around with types can lead to errors
- But an understanding of what types are can sometimes help you figure out odd behaviors and missing data in variables
- It only makes sense to call certain commands on certain types of variables (calling `mean` on a string variable does not make sense)

Labels

- There are several labeling scopes in Stata (dataset, variable, variable levels [i.e. values])
- Labels are a very useful way of conveying metadata about your variables and dataset to others and to yourself
- Good practice suggests that variables and levels of categorical variables should always be labeled
  - Without proper labeling and coding, the interpretation and manipulation of unfamiliar datasets can be needlessly difficult

Dataset and Variable Labels

```
label data label
```

- This command will label your entire dataset with a particular *label* that can be seen when using the command `describe dataset_name`

```
label variable variable_name label
```

- This command will assign a *label* to a variable in your dataset that can be seen when using the command `describe variable_name` or by looking in the variables portion of your Stata window

Variable Levels

- Categorical variables are very common in the social sciences, but must be labeled well to avoid misinterpretation
- Like many statistical programming languages, Stata analyzes categorical variables by assigning numerical values to categories (e.g. “Living” = 0, “Deceased” = 1).
- The actual values assigned to dichotomous variables are arbitrary and only relevant for interpretation (but it makes sense for 0/1 to match the variable name)
- Depending on the type of modeling, dummy variables or ordinal variables will require certain values

Generate a Categorical Variable with Labels

- Generate the numerical values based on another variable

```
generate eff_car = 1 if mpg > 33

replace eff_car = 0 if mpg <= 33

replace eff_car = -1 if mpg < 20
```

- Define a label for each value

```
label define label_name level1 “Label1” level2
“Label2”

label define eff_car_label 1 “Great” 0 “Fair” -1
“Poor”
```

More Label Details

- Assign label to variable

```
label values variable_name label_name

label values eff_car eff_car_label
```

- Modify existing label

```
label define label_name level “newLabel”, modify

label define eff_car_label 0 “Good”, modify
```

`label` and `codebook`

```
label list
```

- This command will output a neat list of all the labels in the current dataset

```
label dir
```

- This command will output the variables with labels

```
codebook
```

- This command will output a detailed codebook with information

```
codebook, problems
```

- This command highlights potential problems with current dataset (variables with more than 9 values are assumed to be continuous)

encode

```
encode variable_name, generate(new_variable)
```

- This command creates a new **number** variable from a string variable, with level labels that correspond to the original string variable
- This is a quick and easy way to generate correctly labeled numerical coding of string variables
- We can give this a try for the countries in our movie_metadata file.

```
encode country, generate(country_code)

codebook country_code
```

`recode`

- The `recode` command is an easy way to recode existing **numerical variables** base on a simple rule

```
recode variable_name rule
```

- For example, we could recode the -1 in the `eff_car` variable (saving first):

```
save autotemp

recode eff_car -1 = 2
```

recode

- Often you want to recode more than one number, to do so you can separate your rules using parentheses:

```
use autotemp, clear

recode eff_car (-1 = 0) (0 = 1) (1 = 2)
```

- Additionally, you can recode and generate a new variable rather than replacing your current one:

```
use autotemp, clear

recode eff_car (-1 = 0) (0 = 1) (1 = 2), generate(new_eff_car)
```

- Finally, you can combine this with the immediate creation of new level labels:

```
recode eff_car (-1 = 0 “Decent”) (0 = 1 “Improved”) (1 = 2
“Excellent”), generate(new_eff_car) label(new_eff_car)
```

Notes

- The `notes` command allows users to add notes to a dataset or a variable that will be permanently linked when the data is saved
- You might want to include details on data provenance, reminders to collaborators, or questions about a particular value

`notes` - displays all notes for a dataset

`notes:`*`note`* - adds a note to the dataset as a whole

`notes variable_name:`*`note`* - adds a note to a particular variable

Exercises (1)

1. Auto Data

   A. Create a do file called auto.do

   B. Call **sysuse** auto (to load auto data)

   C. Create a new price category variable (price_cat). If a car is less than 4000, assign a 0; between 4000 and 6000, assign a 1; greater than 6000 assign a 2.

   D. Give your new variable a thoughtful label. Then, create custom labels for these three values of price_cat and assign them.

   E. Modify the most expensive label to be “Fancy”

   F. Add a note to this dataset with your name and today’s date

Exercises (2)

1. Movie Metadata

    A. Edit your movies.do file so that your dataset will include movies from all countries – then run it.

    B. Create a new categorical variable (country_code) for countries. Give this variable a meaningful label

    C. Use the codebook to check how many observations have missing country_codes, then drop these observations from the dataset.

    D. Create a new categorical variable (cheap). Movies with a budget over 100,000,000 should have a 0, others have 1. Watch out for missing values!

    E. Rename the cheap variable to expensive. Recode it so that observations that used to be 0 are 1, and vice versa

    F. Add brief notes to these two new variables with your name and date