Data Management in Stata

Harvard MIT Data Center

The Institute
for Quantitative Social Science
at Harvard University

Outline

Topic

Materials and Setup

- Lab computer log in:
    - USERNAME: dataclass
    - PASSWORD: dataclass
- Workshop materials:
    - Download class materials from `http://j.mp/stata-datman`
    - Open a file browser, right-click on `StataDatMan.zip`, select the `WinZip` menu and select `Extrat to Here`.

Workshop Description

- This is an Introduction to data management in Stata
- Assumes basic knowledge of Stata
- Not appropriate for people already well familiar with Stata
- If you are catching on before the rest of the class, experiment with command features described in help files

Organization

- Please feel free to ask questions at any point if they are relevant to the current topic (or if you are lost!)
- There will be a Q&A after class for more specific, personalized questions
- Collaboration with your neighbors is encouraged
- If you are using a laptop, you will need to adjust paths accordingly

Opening Files in Stata

- Look at bottom left hand corner of Stata screen
  - This is the directory Stata is currently reading from
- Files are located in the StataDatMan folder in your home directory
- Start by telling Stata where to look for these

```
// change directory
cd "C:/Users/dataclass/Desktop/StataDatMan"

// Use dir to see what is in the directory:
dir
dir dataSets

// use the gss data set
use dataSets/gss.dta
```

Topic

Basic Data Manipulation Commands

Basic commands you'll use for generating new variables or recoding existing variables:

- gen
- egen
- replace
- recode

Many different means of accomplishing the same thing in Stata – find what is comfortable (and easy) for you!

Generate and Replace

The `replace` command is often used with logic statements. Available logical operators include the following:

==	equal to (status quo)
!=	not equal to
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
&	and
|	or

For example:

```
// create "hapnew" variable
gen hapnew = . //set to missing
//set to 0 if happy equals 1
replace hapnew=0 if happy==1
//set to 1 if happy both and hapmar are greater than 3
replace hapnew=1 if happy>3 & hapmar>3
tab hapnew // tabulate the new variable
```

Recode

The recode command is basically generate and replace combined. You can recode an existing variable OR use recode to create a new variable (via the gen option).

```
// recode the wrkstat variable
recode wrkstat (1=8) (2=7) (3=6) (4=5) (5=4) (6=3) (7=2) (8=1)
// recode wrkstat into a new variable named wrkstat2
recode wrkstat (1=8), gen(wrkstat2)
// tabulate workstat
tab wrkstat
```

- The table below illustrates common forms of recoding

Rule	Example	Meaning
#=#	3=1	3 recoded to 1
##=#	2. =9	2 and . recoded to 9
#/# = #	1/5=4	1 through 5 recoded to 4
nonmissing=#	nonmiss=8	nonmissing recoded to 8
missing=#	miss=9	missing recoded to 9

egen

The egen command ("extensions" to the gen command) provides convenient methods for performing many common data manipulation tasks.
For example, we can use egen to create a new variable that counts the number of "yes" responses on computer, email and internet use:

```
// count number of yes on use comp email and net
egen compuser= anycount(usecomp usemail usenet), values(1)
tab compuser
```

Here are some additional examples of egen in action:

```
// assess how much missing data each participant has:
egen countmiss = rowmiss(age-wifeft)
codebook countmiss
// compare values on multiple variables
egen ftdiff=diff(wkftwife wkfthusb)
codebook ftdiff
```

You will need to refer to the documentation to discover what else egen can do: type "help egen" in Stata to get a complete list of functions.

Exercise 1: Generate, Replace, Recode & Egen

Open the gss.dta data.

1. Generate a new variable that represents the squared value of age.
2. Generate a new variable equal to "1" if income is greater than "19".
3. Create a new variable that counts the number of missing responses for each respondent.

Topic

The "bysort" Command

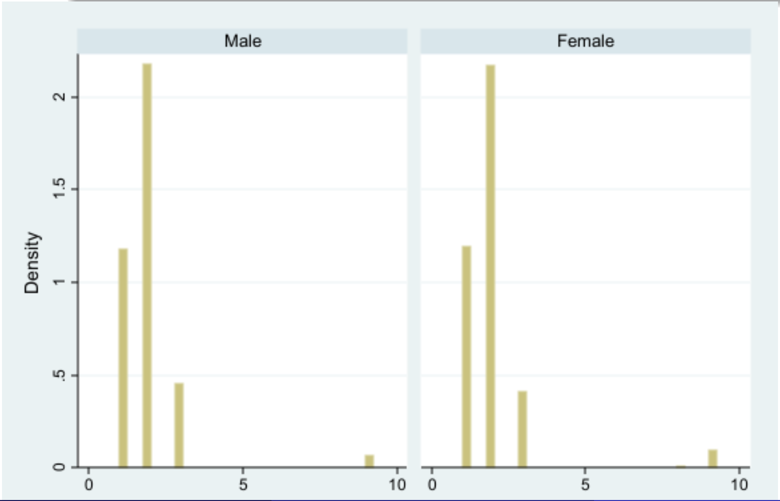
Sometimes, you'd like to create variables based on different categories of a single variable. For example, say you want to look at happiness based on whether an individual is male or female. The "bysort" prefix does just this:

```
// tabulate happy separately for male and female
bysort sex: tab happy
// generate summary statistics using bysort
bysort state: egen stateincome = mean(income)
bysort degree: egen degreeincome = mean(income)
bysort marital: egen marincomesd = sd(income)
```

By prefix vs. by options

Some commands won't work with by prefix, but instead have a by option:

```
// generate separate histograms for female and male
hist nethrs, by(sex)
```



Topic

Missing Values

You always need to consider how missing values are coded when recoding variables.

- Stata's symbol for a missing value is "."
- Stata interprets "." as a large value
- Easy to make mistakes!

To identify highly educated women, we might use the command:

```
// generate and replace without considering missing values
gen hi_ed=0
replace hi_ed=1 if wifeduc>15
// What happens to our missing values?
tab hi_ed, mi nola
```

It looks like around 66% have higher education, but look closer:

```
// gen hi_ed2, but don't set a value if wifeduc is missing
gen hi_ed2 = 0 if wifeduc != .
// only replace non-missing
replace hi_ed2=1 if wifeduc >15 & wifeduc !=.
tab hi_ed2, mi //check to see that missingness is preserved
```

The correct value is 10%. Moral of the story? Be careful with missing values and remember that Stata considers missing values to be large!

Bulk Conversion to Missing Values

Often the data collection/generating procedure will have used some other value besides "." to represent missing values. The `mvdecode` command will convert all these values to missing. For example:

```
mvdecode _all, mv(999)
```

- The "_all" command tells Stata to do this to all variables
- Use this command carefully!
    - If you have any variables where "999" is a legitimate value, Stata is going to recode it to missing
    - As an alternative, you could list var names separately rather than using "_all"

Topic

Variable Types

Stata uses two main types of variables: String and Numeric. To be able to perform any mathematical operations, your variables need to be in a numeric format. Stata can store numbers with differing levels of precision, as described in the table below.

type	Minimum	Maximum	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	$-1.70141173319*10^{38}$	$1.70141173319*10^{38}$	$+/-10^{-38}$	4
double	$-8.9884656743*10^{307}$	$8.9884656743*10^{307}$	$+/-10^{-323}$	8

- Precision for float is $3.795\text{x}10^{-8}$.
- Precision for double is $1.414\text{x}10^{-16}$.

Converting to and from Strings

Stata provides several ways to convert to and from strings. You can use `tostring` and `destring` to convert from one type to the other:

```
// convert degree to a string
tostring degree, gen(degree_s)
// and back to a number
destring degree_s, gen(degree_n)
```

Use `decode` and `encode` to convert to/from variable labels:

```
// convert degree to a descriptive string
decode degree, gen(degree_s2)
// and back to a number with labels
encode degree_s2, gen(degree_n2)
```

Converting Strings to Date/Time

Often date/time variables start out as strings – You'll need to convert them to numbers using one of the conversion functions listed below.

Format	Meaning	String-to-numeric conversion function
%tc	milliseconds	clock(string, mask)
%td	days	date(string, mask)
%tw	weeks	weekly(string, mask)
%tm	months	monthly(string, mask)
%tq	quarters	quarterly(string, mask)
%ty	years	yearly(string, mask)

Date/time variables are stored as the number of units elapsed since 01jan1960 00:00:00.000. For example, the `date` function returns the number of days since that time, and the `clock` function returns the number of milliseconds since that time.

```
// create string variable and convert to date
gen date = "November 9 2020"
gen date1 = date(date, "MDY")
list date1 in 1/5
```

Formatting Numbers as Dates

Once you have converted the string to a number you can format it for display. You can simply accept the defaults used by your formatting string or provide details to customize it.

```
// format so humans can read the date
format date1 %d
list date1 in 1/5
// format with detail
format date1 %tdMonth_dd,_CCYY
list date1 in 1/5
```

Exercise 2: Missing Values, String Conversion, and by Processing

1. Recode values "99" and "98" on the variable, "hrs1" as "missing."
2. Recode the marital variable into a "string" variable and then back into a numeric variable.
3. Create a new variable that associates each individual with the average number of hours worked among individuals with matching educational degrees (see the last "by" example for inspiration).

Topic

Merging Datasets

You can `merge` variables from a second dataset to the dataset you're currently working with.

- Current active dataset = master dataset
- Dataset you'd like to merge with master = using dataset

There are different ways that you might be interested in merging data:

- Two datasets with same participant pool, one row per participant (1:1)
- A dataset with one participant per row with a dataset with multiple rows per participant (1:many or many:1)

Merging Datasets

Before you begin:

- Identify the "ID" that you will use to merge your two datasets
- Determine which variables you'd like to merge
- In Stata >= 11, data does NOT have to be sorted
- Variable types must match across datasets (there is a "force" option to get around this, but not recommended)

Example: Let's say that we had one dataset with individual students (master) and another dataset with information about the students' schools called "school.dta". We would merge these as follows:

```
// Not run: conceptual example only. Merge school and student data
merge m:1 schoolID using school.dta
```

Merge Options

There are several options that provide more fine-grain control over how the merge is carried out:

- In standard merge, the master dataset is the authority and WON'T CHANGE
- If your master dataset has missing data and some of those values are not missing in your using dataset, specify "update" – this will fill in missing data in master
- If you want data from your using dataset to overwrite that in your master, specify "replace update" – this will replace master data with using data UNLESS the value is missing in the using dataset

Appending Datasets

Sometimes you have observations in two different datasets, or you'd like to add observations to an existing dataset. In this case you can use the `append` command to add observations to the end of the observations in the master dataset. For example:

```
// Not run: conceptual example. add rows of data from dataset2
append using dataset2
```

To keep track of where observations came from, use the `generate` option as shown below:

```
// Not run: conceptual example.
append using dataset1, generate(observesource)
```

There is a "force" option will allow for data type mismatches, but again this is not recommended.

Topic

Collapse

Collapse will take master data and create a new dataset of summary statistics

- Useful in hierarchical linear modeling if you'd like to create aggregate, summary statistics
- Can generate group summary data for many descriptive stats
- Can also attach weights

Before you collapse:

- Save your master dataset and then save it again under a new name (this will prevent collapse from writing over your original data_
- Consider issues of missing data. Do you want Stata to use all possible observations? If not, the `cw` (casewise) option will make casewise deletions

Collapse Example

Suppose you have a dataset with patient information from multiple hospitals and you want to generate mean levels of patient satisfaction for hospital:

```
// Not run: conceptual example. calculate average ptsatisfaction by hospital
save originaldata
collapse (mean) ptsatisfaction, by(hospital)
save hospitalcollapse
```

You could also generate different statistics for multiple variables

```
// create mean ptsatisfaction, median ptincome, sd ptsatisfaction for each h
collapse (mean) ptsatisfaction (median) ptincome (sd) ptsatisfaction, by(hos
```

- What if you want to rename your new variables in this process?

```
// Same as previous example, but rename variables
collapse (mean) ptsatmean=ptsatisfaction (median) ptincmed=ptincome
 (sd) sdptsat=ptsatisfaction, by(hospital)
```

Exercise 3: Merge, Append, and Collapse

Open the gss2.dta dataset. This dataset contains only half of the variables that are in the complete gss dataset.

1. Merge dataset gss1.dta with dataset gss2.dta. The identification variable is "id."
2. Open the gss.dta dataset and merge in data from the "marital.dta" dataset, which includes income information grouped by individuals' marital status. The marital dataset contains collapsed data regarding average statistics of individuals based on their marital status.
3. Open the gssAppend.dta dataset and Create a new dataset that combines the observations in gssAppend.dta with those in gssAddObserve.dta.
4. Open the gss.dta dataset. Create a new dataset that summarizes mean and standard deviation of income based on individuals' degree status ("degree"). In the process of creating this new dataset, rename your three new variables.

Topic

Help Us Make This Workshop Better

- Please take a moment to fill out a very short feedback form
- These workshops exist for you–tell us what you need!
- `http://tinyurl.com/StataDatManFeedback`

Additional resources

- training and consulting
  - IQSS workshops: `http://projects.iq.harvard.edu/rtc/filter_by/workshops`
  - IQSS statistical consulting: `http://rtc.iq.harvard.edu`
- Stata resources
  - UCLA website: `http://www.ats.ucla.edu/stat/Stata/`
  - Great for self-study
  - Links to resources
- Stata website: `http://www.stata.com/help.cgi?contents`
- Email list: `http://www.stata.com/statalist/`