14.32 Recitation 2

Paul Schrimpf

February 12, 2009

This recitation will go over the Stata code from lecture 3 in greater detail. This code did two things.

1 Random Numbers and Histograms

The first part generated some random number and drew a histogram. Here is the code:

```
capture log close
log using ln3.log, text replace
clear
// draw 5000 random normals and uniforms, then make histograms
set obs 5000
gen rnor = rnormal()
hist rnor, bin(20) percent ///
  subtitle("Histogram of Random Normal Draws")
graph export ln3_normHist.eps, replace
!epstopdf ln3_normHist.eps

histogram runi, bin(20) percent ///
  subtitle("Histogram of Random Uniform Draws")
graph export ln3_unifHist.eps, replace
!epstopdf ln3_unifHist.eps
log close
```

Some explanations

- Preceding any command with capture hides its output including any error messages. A do-file will stop running if it encounters an error. log close closes a log file if there's one open, and returns an error otherwise. log using filename.log opens a new log file, unless there's already one open, in which case it returns an error. Beginning the do-file with capture log close is convenient when you're debugging. If there's an error in your do-file, it will stop running before it gets to the log close command at the end. Then after you fix the error and re-run the do file, the first thing you need to do is close the open log.
- // makes the rest of a line into a comment. You can also enclose comments in /* A comment */
- /// extends a command onto the next line. By default commands are separated by new lines

```
#delimit; // means commands end with ; instead
gen runi = uniform();
hist runi, bin(20) percent
  subtitle("Histogram of Random Uniform Draws");
#delimit cr // change back to commands ending with new lines aka
            // carraige returns
```

- ! passes the command to the operating system shell is an alias for !. In general, it's a good idea to make your programs require as little user intervention as possible, but sometimes you will need to use external programs, especially when creating nice looking output. This is when shell can be handy. In the above code, we use the linux utility epstopdf utility to convert an eps graph to a pdf. For some reason, Stata can only create pdfs directly on Macs.
- Most Stata commands can be abbreviated – hist and histogram are the same, as are: su, sum, and summarize ; and gen and generate

2 Histogram of CPS Earnings

The second part of the code drew a histogram of weekly earnings. It uses data from the Current Population Survey, or CPS. The CPS is a monthly survey conducted by the Bureau of Labor Statistics. The survey was created to measure unemployment, and each month it asks questions about labor force status. Additionally, in March each year, the survey asks questions about income. This is the data we will mostly be interested in. The Integrated Public Use Microdata Series (IPUMS) website provides access to the March CPS. You can read a brief description of the CPS data hosted at IPUMS at http://cps.ipums.org/cps/intro.shtml.

Before downloading any data, you must register as a user (and agree to "use it for good – never for evil," which seems awfully restrictive to me). After registering, there's a fairly easy to use web interface for selecting which variables you want. Once you submit your selection, it will create a fixed format txt file containing the data, and Stata, SAS, and SPSS code for loading the data. The Stata code looks something like the following:

```
/* Important: you need to put the .dat and .do files in one folder/
   directory and then set the working folder to that folder. */

set more off

clear
infix ///
 int     year                                  1-4 ///
 long    serial                                5-9 ///
 byte    mish                                 10 ///
 float   hhwt                                11-17 ///
 using paul_s_mit_edu_002.dat

replace hhwt=hhwt/100

label var year "Survey year"
label var serial "Household serial number"
label var mish "Month in sample, household level"
label var hhwt "Household weight"

label values year yearlbl

label values serial seriallbl

label define mishlbl 1 "One"
label define mishlbl 2 "Two", add
label define mishlbl 3 "Three", add
label define mishlbl 4 "Four", add
label define mishlbl 5 "Five", add
```

```
label define mishlbl 6 "Six", add
label define mishlbl 7 "Seven", add
label define mishlbl 8 "Eight", add
label values mish mishlbl

label values hhwt hhwtlbl
```

- infix is used for reading fixed format data files. A fixed format data file looks like a big block of numbers, eg:

  ```
  2008000014005283801552520531060480000000000000000000
  2008000014005283802581520890560840000000000000002885
  2008000034002939001342520366510331000000000000000800
  ```

  infix int year 1−4 means that a variable named year will be created by taking the digits in columns 1-4 of the file and storing them as an int.

- label var year "Survey year" attaches a label to year. This label will be displayed when you use the describe command. It will also appear in appropriate places on graphs and tables.

- label define mishlbl 1 "One" creates a value label. When you have a categorical variable coded as an integer, you can create value labels to keep track of what the numbers mean. The labels will be used to display results when appropriate, such as with the tabulate command. label values mish mishlbl attaches the value label mishlbl to the variable mish

To use the IPUMS data, you just run their do-file by typing do email_mit_edu_001.do (or whatever it's called).

The code that actually draws the histograms is:

```
clear
set mem 500m
/*
   Data was extracted from the IPUMS-CPS  http://cps.ipums.org/cps/
   The following variables are used: age, sex, wkswork1, incwage
*/
// unzip cps data (only need to do once)
//!gunzip paul_s_mit_edu_002.dat.gz
/* The IPUMS-CPS provides a do-file for loading the data
   and giving variables nice names. */
do paul_s_mit_edu_002.do

gen earnwke = incwage / wkswork1
gen earnwkeUncen = earnwke
label var earnwkeUncen "Weekly wage and salary earnings (incwage/wksworks1)"
// censor to make histogram look nicer
replace earnwke = . if earnwke > 4000
label var earnwke "Weekly wage and salary earnings"
// keep only men 25-50
keep if sex==1 & age >=25 & age<51
// some summary statistics
sum earnwke*, detail

hist earnwke, bin(20) percent addlabels ///
```

```
  subtitle("Histogram of usual weekly wages -- population distribution") ///
  note("CPS 2008, males age 25-50")
graph export ln3_cpsEarn.eps, replace
!epstopdf ln3_cpsEarn.eps
```

By now, most of this code should be pretty clear. I think every command has either already been covered in these notes, or in Yuqiao's.

Since there were some questions about it in lecture, let's see what happens when we censor weekly earnings at a different value and when we use a different measure of weekly earnings. The above code graphs usual weekly wages, which are defined as wage income last year divided by weeks worked last year. The CPS also includes a variable named earnweek. The description of this variable from the IPUMS website is:

> EARNWEEK reports how much the respondent usually earned per week at their current job, before deductions. Interviewers asked directly about total weekly earnings and also collected information about the usual number of hours worked per week and the hourly rate of pay at the current job. The figure given in EARNWEEK is the higher of the values derived from these two sources: 1) the respondent's answer to the question, "How much do you usually earn per week at this job before deductions?"; or 2) for workers paid by the hour (and coded as "2" in PAIDHOUR), the reported number of hours the respondent usually worked at the job, multiplied by the hourly wage rate given in HOURWAGE.

We should not expect this variable to be exactly the same as our usual weekly wage measure, but it should be similar. The file exploreEarn.do draws a bunch of histograms to compare the two variables. Some new commands in this file include:

- pwcorr to compute pairwise correlations
- reg to estimate a regression, we'll see a lot more of this.
- Using the option name() for graphs to make more than one graph window stay open at a time.
- addplot() to draw plots on top of one another
  - This command appears to have a bug (which I spent a couple of hours trying to figure out). Look at the overlaid histograms with the percent option. It doesn't look like the sum of both sets of bars is 100%. Overlaid density histograms look like they might be correct.

3 Sampling from a data set

To draw a sample with replacement from the data in memory, you can use bsample. In lecture, we used it as a follows:

```
// draw sample of 1500, with replacement
bsample 1500 if earnwke!=.
```

Being simple to use and fast, bsample is the best way to sample with replacement from data. However, there's always more than one way to do something. The following commands accomplish the same thing as bsample 1500

```
// these commands are equivalent to bsample, but slower and more verbose
gen id = _n // goes from 1 to n
sort id
tempfile tmp
save `tmp'
clear
```

```
set obs 1500
gen id = ceil(runiform()*_N) // sample 1500 ids with replacement
sort id
merge id using `tmp'
keep if _merge==3
drop _merge id
```