Stata Tutorial for DEAL

Datasets and Basics Analysis

Ricardo E. Miranda [1] and Xiaoqi (Jade) Peng[1]

Duke University[1]

Nov 8-9, 2022

Outline

Introduction

Plan for the day

Learn basics of variable construction and regression analysis.

We will do this using the datasets from the previous lecture

- Data on expenditures and characteristics of households.
- Data on living place characteristics

Questions to answer?

1. Do households that live in the same place have similar incomes?
2. Does expenditures in education vary by gender of the household head?
    - Is this **because** of the gender?

You don't need to know linear regression

For most of the analysis we will use a tool that is called **linear regression**.

You will have more than a course on all the technical details of linear regression, but the idea is very simple. **Wat is the best linear relation that explains the data?** Note: It might still do a very poor job.

Formally, if Y and X are two variables, we are trying to find a relation of the form

$$Y = a + bX$$

. We do this by trying to find a and b that minimize the sum of (squared) prediction errors.

Note: Economists don't believe the world is linear, but it is a practical simplification.

Managing Data Sets

Preserve / Restore, Drop / Keep

preserve and restore deal with the programming problem where the user's data must be changed to achieve the desired result but, when the program concludes, the programmer wishes to undo the damage done to the data.

- **preserve:**The data in memory remain unchanged. When the program or do-file concludes, the user's data are automatically restored
- **restore:**useful when the programmer needs the original data back and knows that no more damage will be done to the data.
  - **restore, preserve:** when the programmer needs the data back but plans further damage
  - **restore, not:** when the programmer wishes to cancel the previous preserve and to have the data currently in memory returned to the user.

Drop and Keep (if $\cdots$):

- **For variables,** "drop" deletes the variables listed after it. "keep" deletes all variables not listed.
- **For cases,** "drop" deletes cases satisfying the if condition. "keep" deletes all cases that do not satisfy.

Merge

merge is for adding new variables from a second dataset to existing observations. *(Use merge, for instance, when combining hospital patient and discharge datasets. If you wish to add new observations to existing variables, then use "append".)*

- **match merges**
  - one-to-one *merge 1:1 varlist using filename [, options]*
  - one-to-many *merge 1:m varlist using filename [, options]*
  - many-to-one *merge m:1 varlist using filename [, options]*
  - many-to-many *merge m:m varlist using filename [, options]*
- **sequential merges** *merge 1:1 options n using filename [, options]*

Options are optional and offers more choices for merging. See next slide or pg 3-4 in the official Stata documentation for detailed information.

Merge - Options

`keepusing(`*varlist*`)`	variables to keep from using data; default is all
`generate(`*newvar*`)`	name of new variable to mark merge results; default is `_merge`
`nogenerate`	do not create `_merge` variable
`nolabel`	do not copy value-label definitions from using
`nonotes`	do not copy notes from using
`update`	update missing values of same-named variables in master with values from using
`replace`	replace all values of same-named variables in master with nonmissing values from using (requires `update`)
`noreport`	do not display match result summary table
`force`	allow string/numeric variable type mismatch without error

Figure: Merge Options

Append

append appends Stata-format datasets stored on disk to the end of the dataset in memory.

Syntax

`append using` *filename* [*filename* ...] [, *options*]

You may enclose *filename* in double quotes and must do so if *filename* contains blanks or other special characters.

options	Description
`generate(`*newvar*`)`	*newvar* marks source of resulting observations
`keep(`*varlist*`)`	keep specified variables from appending dataset(s)
`nolabel`	do not copy value-label definitions from dataset(s) on disk
`nonotes`	do not copy notes from dataset(s) on disk
`force`	append string to numeric or numeric to string without error

Figure: Append Syntax and Options

Joinby

Form pairwise combinations of observations from dataset in memory with those from the "using" dataset using all common variables or as specified.

Syntax

`joinby` [*varlist*] `using` *filename* [`,` *options*]

options	Description
Options	
When observations match:	
`update`	replace missing data in memory with values from *filename*
`replace`	replace all data in memory with values from *filename*
When observations do not match:	
`unmatched(none)`	ignore all; the default
`unmatched(both)`	include from both datasets
`unmatched(master)`	include from data in memory
`unmatched(using)`	include from data in *filename*
`_merge(`*varname*`)`	*varname* marks source of resulting observation; default is `_merge`
`nolabel`	do not copy value-label definitions from *filename*

Figure: Joinby Syntax and Options

Aggregation

sort & bysort

$$\underline{\texttt{sort}}\ \textit{varlist}\ \left[\textit{in}\right]\ \left[\texttt{, stable}\right]$$

sort arranges the observations of the current data into ascending order based on the values of the variables in varlist.

- **stable** specifies that observations with the same values of the variables in varlist keep the same relative order in the sorted data that they had previously.

The **by** prefix repeats the command for each group of observations for which the values of the variables in varlist are the same.

$$\underline{\texttt{bysort}}\ \textit{varlist}_1\ \left[\texttt{(}\textit{varlist}_2\texttt{)}\right]\ \left[\texttt{, rc0}\right]\texttt{:}\quad \textit{stata\_cmd}$$

- **rc0** specifies that even if the stata cmd produces an error in one of the by-groups, then by is still to run the stata cmd on the remaining by-groups. This is especially useful when stata cmd is an estimation command and some by-groups have insufficient observations.

Collapse

collapse converts the master dataset into means, sums, medians, etc.

`collapse` *clist* [*if*] [*in*] [*weight*] [, *options*]

where *clist* is either

[(*stat*)] *varlist* [[(*stat*)] ...]

[(*stat*)] *target_var*=*varname* [*target_var*=*varname* ...] [[(*stat*)] ...]

stat is assumed to be *mean* if unspecified, but it could be set to *sum*, *count*, *count*, etc.

Options

`by(`*varlist*`)`	groups over which *stat* is to be calculated
`cw`	casewise deletion instead of all possible observations
`fast`	do not restore the original dataset should the user press *Break*; programmer's command

Variable Construction

gen/egen

generate is for simple transformations. **egen** is typically used when working across observations. For example, these two commands are equivalent:

```
. sort oldid
. by oldid: gen newid = 1 if _n==1
. replace newid = sum(newid)
. replace newid = . if missing(oldid)
```

```
. egen newid = group(oldid)
```

Figure: gen & egen comparison

Both commands solves the problem *"I already have an id variable, and I have multiple observations per id, but I want a new id variable containing 1 for the first id, 2 for the second, and so on."* whether the old id is string or numeric.

group(*varlist*) [, missing autotype label[(*lblname*[, replace truncate(#)])]]

- **group()** creates one variable taking on values 1, 2, ... for the groups formed by *varlist*. The order of the groups is that of the sort order of *varlist*.

Encode / For-loop

String variable to numeric variable

`encode` *varname* [*if*] [*in*] , `generate(`*newvar*`)` [`label(`*name*`)` `noextend`]

encode is most useful in making string variables accessible to Stata's statistical routines, most of which can work only with numeric variables.

For-loop syntax (the idea of for-loop should be very familiar):

```
forvalues lname = range {
        Stata commands referring to `lname'
}
```

where *range* is

$\#_1(\#_d)\#_2$	meaning $\#_1$ to $\#_2$ in steps of $\#_d$
$\#_1/\#_2$	meaning $\#_1$ to $\#_2$ in steps of 1
$\#_1\ \#_t$ `to` $\#_2$	meaning $\#_1$ to $\#_2$ in steps of $\#_t - \#_1$
$\#_1\ \#_t$ `:` $\#_2$	meaning $\#_1$ to $\#_2$ in steps of $\#_t - \#_1$

Data Analysis

About Outliers

1. **Trimming**: a percentage of the lowest and (normally an equal percentage of) the highest values of a variable are removed from the data when computing the mean. For example,
   - *trimmean income, percent(0(5)50)*

   remove $0, 5, 10, \cdots, 50$ percent of the cases on each tail of the distribution and show the means computed on each of the trimmed samples. The commands *trimmean* does not change or create any data; it computes means under different conditions of trimming.
2. **Winsordizing**: values at the tails of the distribution are not removed, but are recoded to less extreme values. For example,
   - *winsor income, p(.1) gen(inc_w10)*

   recodes the bottom and the top 10 per cent of the cases in variable 'income' to the values corresponding to the 10th and the 90th percentile, respectively, and write the result to variable inc_w10.

Hypothesis Testing (T-Test)

1. **Single Sample** *ttest write==50*
2. **Paired** *ttest write==read*
3. **Independent Group**
   - assuming **equal variance** *ttest write, by(female)*
   - assuming **unequal variance** *ttest write, by(female) unequal*

Summary Statistics

Variable; Obs (The number of valid observations used); Mean; Std. Err (estimated standard deviation of the sample mean); Std. Dev. (standard deviation); 95% Confidence Interval

Test Statistics

mean or diff; t (Student t-statistic); H0 (the null hypothesis being tested); degrees of freedom; $Pr(T < t), Pr(T > t)$; $Pr(|T| > |t|)$

Regress

regress performs Ordinary Least-Squares linear regression. It can also perform weighted estimation, compute robust and cluster–robust standard errors, and adjust results for complex survey designs.

Some examples to demonstrate the syntax:

- Regression of y on $x1, x2$, and indicators for categorical variable a:
  regress y x1 x2 i.a
- Add the interaction between continuous variable $x2$ and a:
  regress y x1 c.x2##i.a
- With cluster–robust standard errors for clustering by levels of cvar:
  regress y x1 x2 i.a, vce(cluster cvar)

Aregress

areg — Linear regression with a large dummy-variable set.

areg fits a linear regression absorbing one categorical factor. areg is designed for datasets with many groups, but not a number of groups that increases with the sample size (otherwise, use *xtreg*).

Syntax:

- Linear regression of y on x, absorbing an indicator variable for each level of cvar:

areg y x, absorb(cvar)