

**BREVE CURSO DE INTRODUCCIÓN A LA
PROGRAMACIÓN EN STATA (6.0)**

Autor: *Sergi Jiménez-Martín*
Universidad Carlos III de Madrid

DOC. N.º 7/01

Enero de 2001

ÍNDICE

- I. PANORAMA DE STATA
- II. EMPEZANDO Y TERMINANDO SESIONES. USO DE MEMORIA
- III. ESTRUCTURA BÁSICA DE COMANDOS
- IV. LEER DATOS DESDE FICHEROS ASCII
- V. OPERACIONES BÁSICAS CON DATOS
 - V.1. Creando variables nuevas
 - V.2. Procedimientos estadísticos básicos
 - V.3. Ordenación de datos y combinación de ficheros
 - V.4. Manipulación avanzada de variables
- VI. CONTROL DE FLUJOS, BUCLES Y PROGRAMS
- VII. GRÁFICOS Y RESULTADOS
- VIII. COMANDOS DE ESTIMACIÓN
- IX. MÁXIMA VEROSIMILITUD
- X. ADO FILES

I. PANORAMA DE STATA

Ventajas de STATA:

1. Fácil de usar.
2. Rapidez y homogeneidad de comandos en plataformas.
3. Portabilidad de los datos entre plataformas.
4. Gran uso en investigación.

— Extensiones comunes de los ficheros:

.dta: ficheros de datos en formato Stata.

.raw: ficheros de datos ASCII/Text.

.log: output de STATA.

.do: fichero de comandos.

.gph: Stata Graphic File.

.dct: fichero de instrucciones para leer datos ASCII (util para simulaciones).

.ado: Stata "Macro" files.

— Windows vs. Unix

Los comandos son los mismos y la estructura de funcionamiento. Sin embargo el interface gráfico es diferente (mucho menos agradable en la versión unís, aunque la versión 7.0 promete solucionarlo).

II. EMPEZANDO Y TERMINANDO SESIONES. USO DE MEMORIA

– En Windows, STATA se activa a través del icono WSTATA.EXE.

Aparece una ventana con un conjunto de ventanas: REVIEW; RESULTS; VARIABLES; COMMAND.

En la ventana **command** tenemos dos opciones:

- a) Ejecutar interactivamente los comandos.
- b) Ejecutar un fichero de comandos.

La segunda opción tiene muchas ventajas ya que:

1. Sirve de registro de las transformaciones y pasos realizados.
2. Permite comentar los pasos realizados.

3. Permite la ejecución en modo batch en la ventana de comandos: **.do prog1.do**. Alternativamente, haciendo doble clic sobre el icono del programa en Windows.

– Sobre plataformas UNIX se puede ejecutar: **% stata -b do prog1.do**

– Salida del programa: **.e,clear** [en la ventana COMMAND].

— Uso de memoria

STATA funciona cargando en memoria los datos necesarios para el análisis. El límite de la memoria del programa es el propio límite de la memoria del ordenador. También puede usar memoria virtual, aunque esto último sólo se recomienda en ordenadores tipo WS o con disco SCSI.

Instrucciones [en la ventana de comando o al principio de un programa] para el cambio de memoria:

```
.clear
```

```
. set memory 25000
```

Estas instrucciones reservan 25000 K de memoria para el uso de STATA.

Consejo de uso: Mirar el tamaño del fichero de datos y reservar 1.25 veces su tamaño.

III. ESTRUCTURA BÁSICA DE COMANDOS

STATA es una aplicación basada en comandos. La estructura básica es la siguiente:

. [quietly] [by group:] command [varlist] [weight] [if exp] [in range] [using filename] [, options]

Todas las estructuras entre corchetes son opcionales.

Ejemplo: **. tabulate a b if c==2, row col chi2**

“tabulate” tabula las frecuencias de a y b si la variable c vale 2. Con los subcomandos row col chi2 pedimos frecuencias relativas por filas y columnas y el estadístico chi-cuadrado de independencia.

NOTA: “==” es un operador lógico que se usa en condiciones. Veamos que nos dice la utilidad de ayuda sobre los operadores lógicos:

. help operators

help for operators

(manual: [U] 16 Functions and expressions)

Operators in expressions

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

Note that a double equal sign (==) is used for equality testing.

– **Descripción de opciones:**

quietly: Evita que se imprima output innecesario por pantalla durante la ejecución.

by group: Ejecuta el comando para grupos ordenados. Por ejemplo **.by edad sexo: mean aoi**.

varlist: Lista de variables sobre las que aplica el comando.

weight: Pesos sobre los que se ajusta el estadístico descrito en el comando.

if exp: Se usa para restringir la aplicación del comando según un criterio o condición lógica preestablecida.

in range: Se usa para restringir las observaciones sobre las que se aplica.

using filename: Indica el fichero sobre el que se aplica el comando.

, options: Opciones describiendo métodos estadísticos o resultados que se generan en el comando.

— **Obteniendo ayuda [HELP]**

STATA tiene una estructura de ayuda muy completa: **. help tabulate**

Si no esta seguro de un comando, use: **. lookup [chi square]**

IV. LEER DATOS DESDE FICHEROS ASCII

NOTA: PARA ESTA SECCION SE UTILIZAN LOS SIGUIENTES FICHEROS EJEMPLO:

– epa-*-96.raw.

Cada fichero raw contiene datos de la EPA sobre:

TIPO	Variable	POS	ETIQUETA
byte	ciclo	1-2	"periodo de referencia (nueva)"
byte	prov	3-6	"provincia residencia"
byte	npers	7-10	"persona numero"
byte	edad	11-14	"edad en años"
byte	relpp	15-18	"relación con la persona principal"
byte	sexo	19-22	"sexo"
byte	eciv	23-26	"estado civil"
byte	estud	27-30	"estudios terminados"
byte	aoi	31-34	"clasificación de lo entrevistados"
long	facele	35-42	"factor de elevación"
byte	facrep	43-46	"factor de repetición"

– epa96.dct, epa96fix.dct

Los comandos que permiten leer datos en formato ASCII son:

a) **Infile (lectura sin formato o con formato de ficheros ascii).**

b) **Infix.**

c) **insheet.**

a1) Lectura sin formato.

infile varlist [_skip[(#)] [varlist [_skip[(#)] ...]]] using filename

[if exp] [in range] [, automatic byvariable(#) clear]

. infile ciclo prov npers edad relpp using epa-1-96.raw

. infile ciclo prov npers edad relpp using epa-1-96.raw if prov==1

. infile ciclo prov npers edad relpp using if uniform()<=.1

a2) Lectura con el formato en un diccionario.

infile using filename [if exp] [in range] [, automatic using(filename2) clear]

La sintaxis de un diccionario (un fichero creado por un editor es la siguiente):

----- top of dictionary file -----

```
[infile] dictionary [using filename] {  
    * comments may be included freely  
    _lrecl(#)  
    firstlineoffile(#)  
    _lines(#)  
    _line(#)  
    _newline[(#)]  
    [_column(#)]  
    [_skip[(#)]]  
    [type] varname[:lblname] [%infmt] ["var lbl"]  
}
```

(tus datos pueden aparecer aquí)

----- bottom of dictionary file -----

Para un ejemplo completo, véase el ejemplo de lectura recursiva en la página siguiente.

b) **Infix:** lee datos desde un fichero auxiliar de formato fijo. La lectura se puede hacer a través de un diccionario.

. infix using dfilename [if exp] [in range] [, using(filename2) clear]

EJ1: . infix ciclo 1-2 prov 3-6 edad 11-14 using epa-1-96.raw, clear

EJ2: . infix epa96fix.dct if prov==1, using(epa-1-96.raw) clear

c) **insheet:** véanse los ejemplos contenidos en los manuales.

LECTURA RECURSIVA:

El ejemplo que exponemos a continuación, en el fichero **l.do**, lee datos ascii de cuatro trimestres consecutivos de la EPA utilizando el "dictionary" epa96.dct y los guarda en ficheros de datos stata (*.dta):

UN EJEMPLO DE LECTURA RECURSIVA

```
-----progl.do-----
clear
set memory 2000
set more 1
capture log close
log using l,replace
log close
capture prog drop doit
prog def doit
!del epa.dat
local i=96
while `i'<=96 {
local j=1
while `j'<=4 {
qui infile using epa.dct, using(epa-`j'-`i'.raw) clear
compress
save epa-`j'-`i'.dta,replace
clear
local j=`j'+1
}
local i=`i'+1
}
end
doit
-----
```

En este ejemplo se utiliza un objeto "program" de STATA y dos parámetros "local" de control de flujos. Más adelante dedicaremos una sección completa a estos dos conceptos. Veamos ahora la descripción del diccionario.

```
dictionary using epa.dat {
byte    ciclo    %2f    "periodo de referencia (nueva)"
byte    prov     %4f    "provincia residencia"
byte    npers    %4f    "persona numero"
byte    edad     %4f    "edad en años"
byte    relpp    %4f    "relación con la persona principal"
byte    sexo     %4f    "sexo"
byte    eciv     %4f    "estado civil"
byte    estud    %4f    "estudios terminados"
byte    aoi      %4f    "clasificación de lo entrevistados"
long    facele   %8f    "factor de elevación"
byte    facrep   %4f    "factor de repetición"}
}
```

Nota: después de la lectura de archivos ASCII siempre es conveniente minimizar el uso de memoria mediante la instrucción: **compress**

Al aplicar la instrucción todas las variables quedarán guardadas en uno de los siguientes formatos.

. help datatypes

Data and storage types

Numeric Storage Type	Bytes	Minimum	Maximum	Closest to 0 without being 0
byte	1	-127	126	+/-1
int	2	-32,767	32,766	+/-1
long	4	-2,147,483,647	2,147,483,646	+/-1
float	4	-10 ³⁶	10 ³⁶	+/-10 ⁻³⁶
double	8	-10 ³⁰⁸	10 ³⁰⁸	+/-10 ⁻³²³

Precision for float is 6x10⁻⁸ and for double is 2x10⁻¹⁶

String Storage Type	Bytes	Maximum length
str1	1	1
str80	80	80

comando relacionado: **. recast** (*cambia el formato de una variable*).

V. OPERACIONES BÁSICAS CON DATOS

En esta sección describimos la carga de un fichero y algunas operaciones de inspección básicas.

. use \proglepa-1-96, clear

. describe

Contains data from epa-1-96.dta

obs:	5,000		
vars:	11		19 Jan 2001 22:21
size:	90,000 (95.5% of memory free)		
1. ciclo	byte	%8.0g	periodo de referencia (nueva)
2. prov	byte	%8.0g	provincia residencia
3. npers	byte	%8.0g	persona numero
4. edad	byte	%8.0g	edad en años
5. relpp	byte	%8.0g	relación con la persona principal
6. sexo	byte	%8.0g	sexo
7. eciv	byte	%8.0g	estado civil
8. estud	byte	%8.0g	estudios terminados
9. aoj	byte	%8.0g	clasificación de los entrevistados
10. facele	long	%12.0g	factor de elevación
11. facrep	byte	%8.0g	factor de repetición

Sorted by:

.summarize.

[summarize se usa para obtener estadística descriptiva del conjunto de datos en memoria.]

Variable	Obs	Mean	Std. Dev.	Min	Max
ciclo	5000	94	0	94	94
prov	5000	1.8958	.7974143	1	3
npers	5000	2.078	1.160425	1	9
edad	5000	44.2122	19.11156	16	99
relpp	5000	2.0778	1.228678	1	9
sexo	5000	3.546	2.499827	1	6
eciv	5000	1.791	.6543735	1	4
estud	5000	5.5314	5.240239	1	31
aoi	5000	6.6286	2.375161	1	9
facele	5000	17606.05	8401.502	9324	43756
facrep	5000	1	0	1	1

– Etiquetas y descripción de información

```
. label variable sexo "sexo del entrevistado"
. label def sexolab 1 hombre 6 mujer
. label values sexo sexolab
. codebook sexo

sexo ----- sexo del entrevistado
type:          numeric (byte)
label          sexolab
range:        [1,6]          units:          1
unique values: 2             coded missing: 0 / 5000
tabulation:   Freq.         Numeric  Label
              2454         1       hombre
              2546         6       mujer
```

– Guardando los resultados de estos análisis: **.save [filename], replace**

. save junk.dta, replace

V.1. Creando variables nuevas

Existen cuatro formas básicas de crear/modificar variables:

A) NUEVAS VARIABLES: GENERATE

EJ.: **. generate edad2=edad*edad**

. generate rootedad=sqrt(edad)

. generate x=uniform() /generando numeros aleatorios */

. generate x=invnorm(uniform()) /generando num aleat. N(0,1) */

véase: **.help functions** para una descripción de todas las funciones que pueden usarse.

.encode, **.decode**, **.recast** para cambios de formato.

B) CAMBIO DE VARIABLES: REPLACE

.replace var=newvalue if var==some_value

.replace edad=65 if edad>=65

C) CODIFICACIÓN: RECODE

. generate age=edad

. recode age 16/24=1 25/34=2 35/44=3 45/54=4 55/64=5 *=9

. label age "Categorical Age Variable"

. label define age 1 "Under 24" 2 "25-34 Years" 3 "35-44 Years" 4 "45-54 Years" 5 "55-64 Years"

. la val age "age"

D) GENERANDO POR GRUPOS U OTROS CRITERIOS: EGEN

El comando **egen** se usa para producir variables que resumen otras variables para observaciones o grupos de observaciones. Por ejemplo, **egen** se puede usar para crear variables estandarizadas.

. egen stdedad=std(edad)

Más adelante analizaremos ejemplos más elaborados de este comando.

E) OPERADORES DE SERIES TEMPORALES

STATA permite usar operadores de series temporales: lags (L), leads (F), Diferencias (D) y estacionales (S). El comando clave para poder usar operadores de series temporales es **.tsset**. Veamos una descripción de su sintaxis:

```
help tsset
```

```
Declare dataset to be time-series data
```

```
tsset [panelvar] timevar [, format(%fmt) [ daily | weekly | monthly | quarterly | halfyearly | yearly | generic ]]
```

```
tsset
```

```
tsset, clear
```

```
tsfill [, full]
```

comandos relacionados: **tdates**, **time**; **tsreport**

Veamos ahora un ejemplo:

```

clear
. gen year=_n+1980
. gen a=uniform()
. tsset year
time variable: year, 1981 to 2000
. regress a L.a F.a LD.a

```

Source	SS	df	Ms	Number of obs = 17		
Model	.158498862	3	.052832954	F (3, 13)	=	0.60
Residual	1.14785552	13	.088296579	Prob > F	=	0.6273
				R-squared	=	0.1213
				Adj R-squared	=	-0.0814
Total	1.30635439	16	.081647149	Root MSE	=	.29715

a	Coef	Std. Err.	t	P> t	[95% Conf. Interval]	
L1	.3468004	.3692473	0.94	0.365	-.4509099	1.144511
F1	.0369506	.2850634	0.13	0.899	-.5788914	.6527927
LD	-.3220199	.2415616	-1.33	0.205	-.843882	.1998422
_cons	.2307251	.1936188	1.19	0.255	-.1875629	.6490132

V.2. Procedimientos estadísticos básicos

. **tabulate sexo** o alternativamente **.tab1 sexo**. Si escribimos:

. **tab1 sexo, generate (sexo)**, el comando tabula la variable sexo y crea dos variables ficticias llamadas sexo1 y sexo2 para las categorías hombre (valor 1) y mujer (valor 6). **Tabulate** también puede producir tablas de contingencia.

. **tabulate eciv sexo, row col chi**

Estado civil	Sexo del entrevistado		Total
	hombre	mujer	
1	899	708	1607
	55.94	44.06	100.00
	36.63	27.81	32.14
2	1464	1462	2926
	50.03	49.97	100.00
	59.66	57.42	58.52
3	62	310	372
	16.67	83.33	100.00
	2.53	12.18	7.44
4	29	66	95
	30.53	69.47	100.00
	1.18	2.59	1.90
Total	2454	2546	5000
	49.08	50.92	100.00
	100.00	100.00	100.00

Pearson chi2(3) = 200.8217 Pr = 0.000

– Comandos relacionados: **Table, tab1 tab2 tabi tabstats**. Veamos un ejemplo de **table**

. table eciv sexo, c(mean edad)

Estado civil	Sexo del entrevistado	
	hombre	mujer
1	27.2303	27.9774
2	51.0902	48.4036
3	73.1936	72.4871
4	44.3103	44.197

. tabstats

tabstat sexo, by(eciv) sta(mean sd)

Summary for variables: sexo

by categories of: eciv (estado civil)

eciv	mean	sd
1	3.202862	2.483052
2	3.498291	2.500427
3	5.166667	1.8659
4	4.473684	2.314809
Total	3.546	2.499827

– Las matrices de correlación se puede crear usando **.correlate**. Por ejemplo:

. correlate eciv sexo
(obs=5000)

	sexo	eciv
sexo	1.0000	
eciv	0.1627	1.0000

La regresion y el contraste de hipótesis (Wald) son relativamente fáciles.

. reg eciv sexo1 edad

Source	SS	df	Ms	Number of obs = 5000		
Model	794.176864	2	397.088432	F (2, 4997)	1473.73	
Residual	1346.41814	4997	.269445294	Prob > F	= 0.0000	
Total	2140.595	4999	.428204641	R-squared	= 0.3710	
				Adj R-squared	= 0.3708	
				Root MSE	. = .51908	
eciv	Coef	Std. Err.	t	P> t	[95% Conf. Interval]	
Sexo 1	-.1581769	.0147216	-10.745	0.000	-.1870377	-.129316
edad	.0201487	.0003851	52.318	0.000	.0193936	.0209037
_cons	.9778168	.0203352	48.085	0.000	.9379509	1.017683

```
.test sexo1=0 /* Test that sexo1 coefficient is zero */
```

```
. test sexo1
```

```
( 1) sexo1 = 0.0
```

```
F( 1, 4997) = 115.45
```

```
Prob > F = 0.0000
```

V. 3. Ordenación de datos y combinación de ficheros

- **drop/ keep**. Operan de forma similar. Se usan para descartar/conservar variables y/o observaciones.

```
. drop sexo11 /* borra sexo1 de la memoria */
```

```
. keep edad sexo /* borra todas las variables excepto edad y sexo */
```

```
. drop if edad <= 20 /* borra todas las observaciones para las que edad <=20 */
```

```
. keep if edad <=65 /* retiene observaciones para las que edad es menor que 65 */
```

– Ordenación de observaciones: [Sorting a File]

Antes de poder ejecutar una instrucción “by group” o un proceso de “merging” con otro fichero, debemos ordenar adecuadamente los datos según las variables. Sort realiza ordenación ascendente:

```
.sort sexo /* Ordenación según variable */
```

```
.sort sexo eciv /* Ordenación según dos variables */
```

Para realizar ordenaciones ascendentes o mixtas recurrimos a gsort:

```
.gsort -sexo /* Ordenación descendente*/
```

```
.gsort sexo -eciv /* ascendente de sexo y descendente de eciv */
```

– Merging Data Files

“Merging” quiere decir añadiendo variables al conjunto de datos activo. Para poder llevarlo a cabo ambos conjunto de datos deben estar ordenados en sobre la base de las mismas variables y en el mismo orden. Ejemplo de sintaxis:

Ejercicio 1: Combinamos los datos ya conocidos en **proglepa-1-96** con los datos en **proglinc.dta** que contiene datos de ingresos en logaritmos por sexo y edad.

```
use proglepa-1-96,clear
```

```
. sortr sexo edad
```

```
. merge sexo edad using inc
```

Nótese que **. merge** crea una variable adicional, **_merge**, que puede tomar tres valores:

1→ Observation found only in master

2→ Observation found on in “using filename”.

3→ Observation found in both

Por tanto, resulta conveniente tabularla para comprobar si obtenemos el resultado correcto.

. tab1 _merge

-> tabulation of _merge

_merge	Freq.	Percent	Cum.
3	5000	100.00	100.00
Total	5000	100.00	

– Actualizando ficheros con merge:

Suponga que tiene una nueva versión de algunas variables. Puede combinarlas con las anteriores haciendo:

. merge using filename, update replace

In este caso los valores de _merge puede ser:

1→ Observation found only in master

2→ Observation found on in "using filename"

3→ Observation found in both, identical values in both

4→ Observation from both, missing in master is updated

5→ Observation from both, master disagrees with using replaced with using value(s)

– Concatenando ficheros

. use data

. append using filename

– Formando combinaciones entre ficheros:

(a). Joinby. Crea un fichero con todas las parejas entre ficheros. Ejemplo:

. use parents

. joinby famid using children

(b) .Cross. Crea todas las posibles combinaciones entre ambos ficheros. [comando peligroso!!!]

. Cross using filename.

(c). fillin. Rellena con observaciones todas las posibles combinaciones de un listado de variables. Los valores de otras variables aparte de las que definen el relleno se asignan a "missing". (Util para balancear conjuntos de datos, especialmente paneles). Ejemplo:

. fillin sexo edad

V.4. Manipulación avanzada de variables

Existen numerosas posibilidades de manipular variables en STATA. Aunque veremos algunos ejemplos, se recomienda ver la ayuda de los comandos `generate` y `egen`. Ejemplos

- . **by group: generate size=_N** /*_N indica el número de observaciones en STATA*/
- . **egen difer=dif(A-B), by(group)** /* Donde A y B son variables */

nota: antes del comando `. by group` se recomienda poner `. quietly` ya que `. by group` genera un output en pantalla considerable.

— RESHAPE Y COLLAPSE

. **collapse**: este comando colapsa el conjunto de datos activo según algún criterio pre-determinado. Por ejemplo:

```
*----ejemplo construccion de pesos muestrales y pesos poblacionales según grupos de sexo y edad ----*
. use prog\epa-1-96,clear
replace facele=facele*facrep/100
collapse (count) sw=facele (sum) pw=facele, by(sexo edad) fast
```

Veáse la ayuda de STATA para una descripción completa de las opciones del comando.

. **reshape**: permite cambiar la organización de un conjunto de datos entre ancho y largo. En los ejercicios que seguirán veremos algunos ejemplo. Tomemos ahora el ejemplo de la ayuda de STATA.

(wide form)				
-i-		----- x_ij -----		
id	sex	inc80	inc81	inc82
1	0	5000	5500	6000
2	1	2000	2200	3300
3	0	3000	2000	1000

(long form)			
-i-	-j-		-x_ij-
id	year	sex	inc
1	80	0	5000
1	81	0	5500
1	82	0	6000
2	80	1	2000
2	81	1	2200
2	82	1	3300
3	80	0	3000
3	81	0	2000
3	82	0	1000

- a) Cambio de ancho a largo: `. reshape long inc, i(id) j(year)`
- b) Cambio de largo a ancho: `. reshape wide inc, i(id) j(year)`

Hasta la versión 5.0 el comando `. reshape` era un poco más complejo. Sobre el ejemplo anterior tenemos:

- . **reshape groups year 80-82**
- . **reshape vars inc**
- . **reshape cons id sex**
- . **reshape long [or wide]**

Véase la ayuda de STATA para una descripción completa de los comandos.

VI. CONTROL DE FLUJOS, BUCLES Y PROGRAMS

La instrucción **. program** define y manipula subrutinas. Opciones basicas.

```
. program define pgmname [, { nclass | rclass | eclass | sclass } ] /*definición de un programa */  
. program drop { pgmname [pgmname ...] | _all } /*borrado de programas anteriores */  
. program list [ pgmname [pgmname ...] | _all ] /*listado de programas */
```

Ejemplo:

```
capture prog drop doit  
prog def doit  
!del epa.dat  
local i=96  
while `i'<=96 {  
  local j=1  
  while `j'<=4 {  
    qui infile using epa.dct, using(epa-`j'-`i'.raw) clear  
    compress  
    save epa-`j'-`i'.dta,replace  
    clear  
    local j=`j'+1  
    local i=`i'+1  
  }  
  end  
  doit  
}
```

Las instrucciones **. global/macro** y **.local** permite definir controles de flujos. **. local** sólo es válido en el programa que se está ejecutando corrientemente, **. global/macro** permanece activo en todos los programas que se ejecutan en una secuencia. Sintaxis:

```
. local j=1
```

```
macro def l=1
```

– **Bucles.** Los bucles se definen de forma muy sencilla. Por ejemplo:

con local	con global/macro
local j=1	macro def l=1
while `j'<=4 {	while \$l=1 {
display “hola”	display “hola”
local j=`j'+1	macro def a=\$a+1
}	}

– **IF/ELSE.** La cláusula condicional es muy sencilla:

```
. if `j'==1 { display "hola"}
.else { display "adios"}
```

– Pasando argumento al programa: **. do programfile arg1 arg2 argN**

-----program: proba.do-----

Ejecución: **.do proba hola adios.**

display ``1''

. display ``1''

display ``2''

hola

-----end of proba.do-----

. display ``2''

adios

– **COMANDOS .for ; .foreach ; forvalues**

. for tiene la siguiente sintaxis

for listtype list : stata_cmd_containing_X

Si listtype es	entonces	list es	entonces list es
varlist		varlist	
newlist		new varlist	
numlist		numlist (see help numlist)	
anylist		list of words	

Ejemplos básicos:

```
. for var m*: replace X = X/10
. for new u1-u10: gen X = uniform()
. for num 1/5: count if freq==X
. for any . -1: replace x = . if y==X
. for var ch1 ch2 ch3: ttest X=0
. for var ch1-ch3: ttest X=0
. for var ch*: ttest X=0
. for var pre* post* \ num 2.53 4.2 0.93 2.53: ttest X=Y
. for num 1/20: rename vX answerX
. for any male female: reg y x1 x2 if sex=="X" \ predict yX
```

for loops pueden estar anidados. Por ejemplo: **. for A in num 1/5 : for B in num 1/3 : gen aAbB = aA * bB**

genera variables **a1b1, a1b2, a1b3, a2b1, ..., a5b3** del producto de **a1, a2, ..., a5** con **b1, b2, and b3**.

Véase **.help for** para una descripción completa

. foreach tiene la siguiente sintaxis: **foreach lname {in|of listtype} list {**

```
foreach file in this.dta that.dta theother.dta {
    append using "`file" }

local grains "rice wheat corn rye barley oats"

foreach x of local grains {
    display "`x'"
}
}
```

Loop over the elements of a global macro.

```
global money "Franc Dollar Lira Pound"

foreach y of global money {
    display "`y'" }
}
```

. forvalues itera sobre series de valores más o menos complejas.

```
forvalues x = 1/1000 { ... }

. forvalues i = 1(1)100 { generate x`i' = uniform() }
```

For variables var5, var6, ..., var13 output the # of obs. greater than 10.

```
. forval num = 5/13 { count if var`num' > 10 }
```

Produce individual summarize commands for variables x5, x10, ..., x300.

```
. forvalues k = 5 10 to 300 {
    summarize x`k'
}
}
```

A Loop over non-integer values that includes more than one command.

```
. forval x = 31.3 31.6 : 38 {
. count if var1 < `x' & var2 < `x'

    summarize myvar if var1 < `x'
}
}
```

VII. GRÁFICOS Y RESULTADOS

Producir gráficos sencillo es relativamente fácil. Ejemplo:

```
. use epa-1-96,clear /* usamos epa-1-96 como conjunto datos */
. gen empl=aoi==3 | aoi==4 /* generamos una dummy de empleo */
. collapse (mean) emp, by(sexo edad) /* calculamos medias por sexo y edad */
. gr emp edad if sexo==1, c(l) save(emph,replace) /* grafico de tasa empleo para hombres */
. gr emp edad if sexo==6, c(l) save(empm,replace) /* grafico de tasa empleo para mujeres */
```

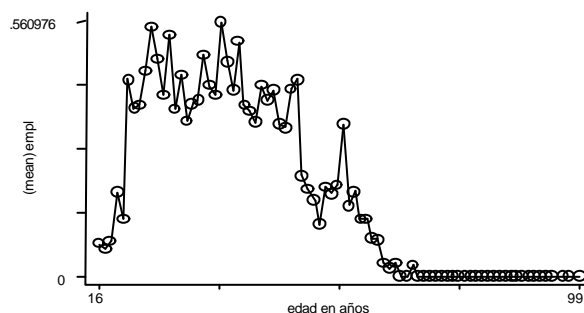
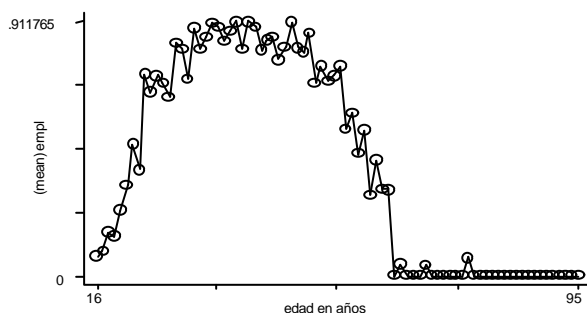
Si queremos combinarlos: `. gr using emph empm, save(comb,replace)`

Se pueden combinar tantos gráficos como se quieran aunque la combinaciones optimas son 1, 4, 9, 16, 25. Es decir, STATA divide el espacio gráfico en n^2 partes.

Para imprimirlo tenemos varias opciones (versión 6.0). Una a través del propio programa. Sin embargo, muchas veces resulta más interesante guardarlo como fichero eps o wmf, para importarlo en, por ejemplo un fichero Tex o MSword©.

-Fichero eps: `. gphprint, nologo sa(comb.eps,replace)` /* requiere tener configurada impresora PS */

-Fichero wmf: `. gphprint, nologo sa(comb.wmf,replace)`



Opciones típicas en gráficos:

- a) Títulos: `title("mygraph")`
- b) Escala de los ejes: `xlab() ylab()`
- c) Margenes: `gap(2)`
- d) Símbolos: `s(opd....)`

Hoy otras muchas opciones y por supuesto tipos de gráficos. En este caso se recomienda consultar tanto la ayuda de STATA como el manual de gráficos con STATA. [Además Véanse el ejemplo contenido en el programa wgt.do]

– GUARDANDO LOS RESULTADOS

En STATA los resultados del análisis se puede guardar en un archivo .log. Al principio del programa es conveniente abrir un fichero de output

. capture log close

. log using output.log, replace [o append]

Otras instrucciones útiles

. log on/off

. log close

. set loglinesize #

(A). EJEMPLO 2. RECAPITULACIÓN

En este ejemplo:

- (1) Ejecutamos un programa (master.do) que contiene varios subprogramas.
- (2) Leemos los archivos dta que hemos creado en el ejemplo 1 (**data.do**). En **data.do** se llama un programa (**recepta.do**) que transforma algunas variables. Creamos el archivo (**data.dta**).
- (3) Calculo de pesos poblacionales y muestrales (**wgt.do**)
- (4) Calculo de LFPR, empleo y desempleo (**rr.do**)
- (5) Gráficos de tendencias de tasas bajo pesos alternativos (**fwgt.do**)

```
/*MASTER.DO :Set of programs for predicting labor force participation rates,  
employment rates and unemployment rates using the Italian labor force survey (RTFL) */  
version 6.0  
set more 1  
* ----- Basic trends -----  
* Build the data set  
run data  
* Compute population and sample weights  
do wgt  
* Compute LFPR, employment and unemployment rates  
do rr  
* Trends in aggregate rates using alternative weighting schemes  
do fwgt
```

```

/* *****
data.DO
***** */
version 6.0
set more 1
* ----- Build-up the data -----
capture prog drop doit
prog def doit
/* */
local t=96
while `t'<=96 {
    local j=1
    while `j'<=4 {
        di "year=`t', trim=`j'"
        use `c_stata\prog\epa-`j'-`t', clear
        keep ciclo sexo edad estud aoi prov facele facrep eciv
        ren edad age
        ren sexo sex
        replace facele=facele*facrep
        ren facele peso
        drop facrep
        run recepa
        gen byte year=92 + int((ciclo-78)/4)

        gen byte qtr=ciclo-77-(year-92)*4
        if `t'==96 & `j'==1 { save data,replace}
        else {
            append using data
            save data,replace
        }
        local j=`j'+1
    }
    local t=`t'+1
}
use data, clear
sort ciclo
compress
save data, replace
end
doit

```

```

*-----RECEPA.DO-----
gen byte marstat=eciv
lab var marstat "estado civil"
drop eciv
gen byte educ=1 if estud<=7 /* analfabetos o sin estudios */
qui replace educ=2 if estud>=8 & estud<=13 /* BUP, COU, ESO, FP (Secund.) */
qui replace educ=3 if estud>=15 /* DIPLOMA 3 ANOS UNIV. */
lab var educ "nivel de estudios"
drop estud
rename prov reg
#delimit ;
recode reg 4 11 14 18 21 23 29 41 51 52=1 22 44 50=2 33=3 7=4 35 38=5 39=6
2 13 16 19 45=7 5 9 24 34 37 40 42 47 49=8 8 17 25 43=9
3 12 46=10 10 6=11 15 27 32 36=12 28=13 30=14 31=15
20 48=16 26=17;
#delimit cr
recode reg 1 5 18 11 14=1 7 8=2 13=3 2 4 9 10=4 *=5
recode aoi 3/4=3 5/6=5 1/2 7/9=7 /* relacion con la actividad */
gen byte empl=aoi==3|aoi==4 /* empleado */
gen byte unemp=aoi==5|aoi==6 /* parado */
drop aoi
ren reg regio
ren peso weight
ren unemp unempl
*-----

```

De los tres archivos que se ejecutan en master.do incluimos solo los dos primeros

```
/* RR.DO: Compute labor force participation rates, employment rates and unemployment rates by socio-demographic group. */

version 6.0
set more 1
use data, replace
replace sex=sex==6
local X="age sex educ marstat regio year qtr" /* covariates */
keep empl unempl `X' weight

replace age=24 if age<24
replace age=60 if age>=60 & age<=64
replace age=65 if age>=65
qui replace marstat=marstat==2
rename weight pop
replace empl=empl*pop
replace unempl=unempl*pop
collapse (sum) empl unempl pop, by(`X') fast
compress
qui replace year=year+1900
gen date=yq(year,qtr)
drop year qtr
format date %tq
compress
gen double AR=(empl+unempl)/pop /* LFPR */
gen double ER=empl/pop /* employment rate */
keep age sex educ marstat regio date AR ER
sort age sex educ marstat regio date
compress
lab data "Estimated rates"
save rr, replace
```

```

/* WGT.DO: Compute population and sample weights by socio-demographic group.
Need at least 105MB of RAM to run. */
version 6.0
set more 1
* ----- Build-up the data -----!del junk*.dta
-----
local X="age sex educ marstat regio year qtr" /* covariates */
local X1="age sex educ marstat regio date"
use data, clear
keep weight `X'
qui replace age=20 if age<20
replace sex=sex==6
replace age=24 if age<24
replace age=60 if age>=60 & age<=64
replace age=65 if age>=65
qui replace marstat=marstat==2
collapse (count) sw=weight (sum) pw=weight, by(`X') fast
qui replace year=year+1900
gen int date=yq(year,qtr)
drop year qtr
format date %tq
sort date
qui save wgt, replace
collapse (sum) sw pw, by(date) fast
rename sw S
rename pw P
sort date
merge date using wgt
assert _m==3
drop _m /* weights add up to 100 */

qui replace pw=100*pw/P
lab var pw "population weight (%)"
qui replace sw=100*sw/S
lab var sw "sample weight (%)"

drop P S
compress
sort `X1'

sort age sex educ marstat regio date
compress
lab data "Population and sample weights"
save wgt, replace
* ----- Graphs -----
use wgt, clear
collapse (sum) sw pw, by(age sex educ date) fast
collapse (mean) sw pw, by(educ age sex) fast
lab var sw "sample weights (%)"
lab var pw "population weights (%)"
local a1="by(educ) c(ll) sy(.p) xlab(20,30,40,50,60) ylab"
set textsize 80
gr sw pw age if sex==0 & 20<age & age<65, `a1' ti(Men) sa(wgt1,replace)
gr sw pw age if sex==1 & 20<age & age<65, `a1' ti(Women) sa(wgt2,replace)

use wgt, clear
ren age agegrp
recode agegrp 20/29=1 30/54=2 55/64=3 65/75=4
collapse (sum) sw pw, by(educ agegrp sex date) fast
lab var sw "sample weights (%)"
lab var pw "population weights (%)"
set textsize 80
local options="by(educ) c(ll) sy(.p) xlab ylab"
gr sw pw date if agegrp==1 & sex==0, `options' ti(Men aged 16-29) sa(wgt3,replace)
gr sw pw date if agegrp==2 & sex==0, `options' ti(Men aged 30-49) sa(wgt4,replace)
gr sw pw date if agegrp==3 & sex==0, `options' ti(Men aged 55-64) sa(wgt5,replace)
gr sw pw date if agegrp==4 & sex==0, `options' ti(Men aged 65+) sa(wgt6,replace)
gr sw pw date if agegrp==1 & sex==1, `options' ti(Women aged 16-29) sa(wgt7,replace)
gr sw pw date if agegrp==2 & sex==1, `options' ti(Women aged 30-54) sa(wgt8,replace)
gr sw pw date if agegrp==3 & sex==1, `options' ti(Women aged 55-64) sa(wgt9,replace)
gr sw pw date if agegrp==4 & sex==1, `options' ti(Women aged 65+) sa(wgt0,replace)

```

VIII. COMANDOS DE ESTIMACIÓN

STATA tiene un gran número de estimadores preprogramados (ficheros .ado) que permiten, por lo general, un gran número de opciones. La lista de comandos se puede agrupar como sigue:

-Estadísticas básicas

-Anova y Ancova:

-Regresión lineal y MV

-Modelos logísticos y probit

-Modelos de duración/supervivencia

-Series temporales

-Panel de datos

-Análisis de datos muestrales.

Casi todos los comandos de estimación de una ecuación se organizan como sigue:

command varlist [weight] [if exp] [in range] [, options]

Ejemplo:

```
. use epa-1-96, clear
. gen emp=aoi==3|aoi==4          /* empleo */
. gen par=aoi==5|aoi==6        /* paro */
. gen mujer=sexo==6
. gen head=relpp==1
. probit emp mujer head edad aw=facele in 1/2000
```

y los de ecuaciones múltiples:

command (varlist) ... (varlist) [weight] [if exp] [in range] [, options]

– Otros comandos de interés en la regresión:

Adjust: Crea tablas de predicción para variables categóricas.

. adjust, by(var1 var2) se generate(xb) [xb,p,exp]

test

lrtest: Contraste del RV después de comando que calculan la verosimilitud.

```
. clear
. use epa-1-96
. gen byte emp=aoi==4|aoi==5
. gen edad2=edad*edad
```

```

. qui logit emp edad edad2
. lrtest, saving(0)
. qui logit emp edad
. lrtest

```

Logit: likelihood-ratio test

chi2(1) = 1034.72
Prob > chi2 = 0.0000

EJEMPLO DE ESTIMACIÓN DE SISTEMAS DE ECUACIONES LINEALES

```

. sureg (price foreign weight length) (mpg foreign weight) (displ foreign weight)

o, usando global macros

. global price (price foreign weight length)
. global mpg (mpg foreign weight)
. global displ (displ foreign weight)

. sureg $price $mpg $displ

set more 1
clear
set memory 10000
set matsize 400
version 6.0
capture log close
log using air4,replace
log off
/*
Badi H. Baltagi, James M. Griffin, and Sharada R.Vadali.
"Excess Capacity: A Permanent Characteristic of U.S. Airlines, "
Journal of Applied Econometrics, Vol. 13, No. 5, 1998, pp. 645-657.
All data are in the file costfn.dat, a file in DOS format that is zipped
in bgv.zip.

*infile year vc y k pl pm pf pk ls ms rs stage str2 name using costfn.dat
use costfn,clear
la var y "OUTPUT (multilateral index of seat miles actually flown)"
la var vc "VARIABLE COST (sum of labor, fuel, material costs)"
la var k "CAPITAL STOCK"
la var pl "PRICE OF LABOR"
la var pm "PRICE OF MATERIALS"
la var pf "PRICE OF FUEL"
la var pk "PRICE OF CAPITAL"
la var ls "LABOR COSTS/ VARIABLE COSTS (labor share in variable costs)"
la var ms "MATERIAL COSTS/VARIABLE COSTS (material share in variable costs)"
la var rs "REVENUE SHARE VARIABLE=(REVENUE/VARIABLE COSTS)"
la var stage "STAGE LENGTH MEASURED AS AVERAGE LENGTH IN MILES OF A FIRM FLIGHT"
la var name "NAME OF THE AIRLINE"
la var year "year" */

```

```

use costfn,clear
* estimation
gen double lnVC=log(vc)
gen double lny=log(y)
gen double lnk=log(k)
gen double lnwf=log(pf)
gen double lnwl=log(pl/pf)
gen double lnwm=log(pm/pf)
gen double lns=log(stage)

gen double lnwlnwl=.5*lnwl*lnwl
gen double lnwlnwm=lnwl*lnwm
gen double lnwmlnwm=.5*lnwm*lnwm

gen double lny2=.5*lny*lny
gen double lnk2=.5*lnk*lnk
gen double lns2=.5*lns*lns
gen double lnslny=log(stage)*log(y)
gen double lnslnk=log(stage)*log(k)
gen double lnynk=log(y)*log(k)

gen double lnklnwl=lnk*lnwl
gen double lnklnwm=lnk*lnwm
gen double lnynlwl=lny*lnwl
gen double lnynlwm=lny*lnwm
gen double lnslnwl=lns*lnwl
gen double lnslnwm=lns*lnwm

gen double lnwlfw=lnwl
gen double lnwlfw=lnwm
replace lnVC=lnVC-lnwf
tab1 name, g(fi)
tab1 year, g(t)
drop fi1 t1
replace rs=-rs
*global lnVC (lnVC fi* t* lny-lnslnwm)
global lnVC (lnVC fi* t* lny lnk lnwl-lnwm lns-lnslnwm)
global ls (ls lny lnk lnwl-lnwm lns)
global ms (ms lny lnk lnwl-lnwm lns)
global rs (rs lny lnk lnwl-lnwm lns)
/* returns to scale restrictions */
cons de 1 [lnVC]lny + [lnVC]lnk = 1
cons de 2 [lnVC]lny2 + [lnVC]lnynk = 0
cons de 3 [lnVC]lnk2 = [lnVC]lny2
cons de 4 [lnVC]lnslny + [lnVC]lnslnk = 0
cons de 5 [lnVC]lnynlwl + [lnVC]lnklnwl = 0
cons de 6 [lnVC]lnynlwm + [lnVC]lnklnwm = 0
/* share equation restrictions */
cons de 7 [rs]_cons = [lnVC]lny
cons de 8 [rs]lny = [lnVC]lny2
cons de 9 [rs]lnk = [lnVC]lnynk
cons de 10 [rs]lnwl = [lnVC]lnynlwl
cons de 11 [rs]lnwm = [lnVC]lnynlwm
cons de 12 [rs]lns = [lnVC]lnslny
cons de 13 [ls]_cons = [lnVC]lnwl
cons de 14 [ls]lnwl = [lnVC]lnwlnwl
cons de 15 [ls]lnwm = [lnVC]lnwlnwm
cons de 16 [ls]lns = [lnVC]lnslnwl
cons de 17 [ls]lny = [lnVC]lnynlwl
cons de 18 [ls]lnk = [lnVC]lnklnwl
cons de 19 [ms]_cons = [lnVC]lnwm
cons de 20 [ms]lnwl = [lnVC]lnwlnwm
cons de 21 [ms]lnwm = [lnVC]lnwmlnwm
cons de 22 [ms]lns = [lnVC]lnslnwm
cons de 23 [ms]lny = [lnVC]lnynlwm
cons de 24 [ms]lnk = [lnVC]lnklnwm
/* homog, simmetry, adding up restrictions */
log on
*sureg $lnVC $ls $ms $rs, const(1-24) isure
reg3 $lnVC $ls $ms $rs, const(1-24) sure tol(0.001) ireg3
log close

```

– Ejemplo 2 de estimación:

Aparte de otros ejemplos que se presentarán a lo largo del curso, un ejemplo muy completo de procesos de estimación y contraste, incluyendo técnicas de panel se encuentra en el directorio EJEMPLO 5. Dicho conjunto de programas replica los resultados presentados en el artículo:

-Sergi Jiménez-Martín, J.M. Labeaga y A. López (1998) "Participation, heterogeneity and dynamics in tobacco consumption: evidence from cohort data" (con J.M Labeaga y Angel López), *Health Economics*, 7, 401-414.

Veamos un ejemplo muy sencillo del conjunto de programas en EJEMPLO 5:

```
*-----tsamp0.do-----programa usando poblacion mayor de 16a
set more 1
des
log using tabaco10, replace
use tabaco
keep if cpn>0
replace year=year-1900
#delimit ;
gen gtab=ctab*ptab; gen gtabn=ctabn*ptabn; gen gtabr=ctabr*ptabr;
gen share=100*gtab/cpn; la var share "share of tobacco consumption";
gen sharen=100*gtabn/cpn; la var sharen "share of black tobacco consumption";
gen sharer=100*gtabr/cpn; la var sharer "share of virginia tobacco consumption";
gen prtab=100*ptab/pc; la var prtab "real price of tobacco";
gen prtabn=100*ptabn/pc; la var prtabn "real price of black tobacco";
gen prtabr=100*ptabr/pc; la var prtabr "real price of virginia tobacco";
gen pcctab=1000000*(ctab/pobm16); la var pcctab "per capita>16a tobacco consumption"
set textsize 75;
gr share sharen sharer year, ylab(0.2,0.4,0.6,0.8,1,1.2,1.4,1.6,1.8,2.0)
c(III) xlab(64,70,75,80,85,90,94) saving(share,replace);
replace share=share/100;
gen lcpnrpc=log((10000*(cpn/pc))/pobm16);
gen lptab=log(ptab)-log(pc);
gen lpartf=log(partf);
gen lpcctab=log(pcctab); la var pcctab "log per capita>16a tob.cons.";
gen sharek=share/pcctab; la var sharek "share tob.cons. over pobm16";
gen share1=share[_n-1]; gen share2=share[_n-2];
gen lpcctab1=lpcctab[_n-1]; gen lpcctab2=lpcctab[_n-2];
sum share share1 share2 lcpnrpc lptab legissum lpartf lpcctab lpcctab1 lpcctab2;
egen sharem=mean(share);
reg share lptab lcpnrpc legissum;
gen ep=-1+_b[lptab]/sharem;gen er=1+_b[lcpnrpc]/sharem;sum ep er;
reg share lptab lcpnrpc legissum if year>=65;
replace ep=-1+_b[lptab]/sharem;replace er=1+_b[lcpnrpc]/sharem;sum ep er;
reg share lptab lcpnrpc legissum if year>=66;
replace ep=-1+_b[lptab]/sharem;replace er=1+_b[lcpnrpc]/sharem;sum ep er;
reg share share1 lptab lcpnrpc legissum;
reg share share1 lptab lcpnrpc legissum if year>=66;
reg share share1 share2 lptab lcpnrpc legissum;
reg share lptab lcpnrpc legissum lpartf;
replace ep=-1+_b[lptab]/sharem;replace er=1+_b[lcpnrpc]/sharem;sum ep er;
reg share lptab lcpnrpc legissum lpartf if year>=65;
replace ep=-1+_b[lptab]/sharem;replace er=1+_b[lcpnrpc]/sharem;sum ep er;
reg share lptab lcpnrpc legissum lpartf if year>=66;
replace ep=-1+_b[lptab]/sharem;replace er=1+_b[lcpnrpc]/sharem;sum ep er;
reg share share1 lptab lcpnrpc legissum lpartf;
reg share share1 lptab lcpnrpc legissum lpartf if year>=66;
reg share share1 share2 lptab lcpnrpc legissum lpartf;
reg lpcctab lptab lcpnrpc legissum;
reg lpcctab lptab lcpnrpc legissum if year>=65;
reg lpcctab lptab lcpnrpc legissum if year>=66;
reg lpcctab lpcctab1 lptab lcpnrpc legissum;
reg lpcctab lpcctab1 lptab lcpnrpc legissum if year>=66;
reg lpcctab lpcctab1 lpcctab2 lptab lcpnrpc legissum;
reg lpcctab lptab lcpnrpc legissum lpartf;
reg lpcctab lptab lcpnrpc legissum lpartf if year>=65;
reg lpcctab lptab lcpnrpc legissum lpartf if year>=66;
reg lpcctab lpcctab1 lptab lcpnrpc legissum lpartf;
reg lpcctab lpcctab1 lptab lcpnrpc legissum lpartf if year>=66;
reg lpcctab lpcctab1 lpcctab2 lptab lcpnrpc legissum lpartf;
```

– Variables instrumentales

```
. ivreg depvar [varlist1] (varlist2=varlist_iv) [if exp] [in range]
    [weight] [, level(#) beta hascons noconstant robust
    cluster(varname) first noheader eform(string)
    depname(varname) mse1 ]
```

depvar, varlist1, varlist2, and varlist_iv may contain time-series operators;

. **ivreg** estima una regresión lineal usando IV o 2SLS de **depvar** sobre varlist1 y varlist2 using varlist_iv (junto a varlist1) como instrumentos para varlist2. Es decir, varlist1 and varlist_iv son las variables exógenas y varlist2 las variables endógenas.

– Estimación de panel

Los estimadores de panel tiene la siguiente estructura:

```
. xtcmd ... [, i(varname) t(varname) ... ]
```

donde i indica el identificador individual y t() el tiempo. También se pueden definir en base a

```
. iis [identificador individuo]. Ejemplo: . iis pid
```

```
. tis [variable que identifica tiempo]. Ejemplo: . tis time
```

La versión 6.0 presenta importantes mejoras sobre las anteriores. Por lo tanto, esperamos a tener un conocimiento más profundo de dicha versión para realizar algún ejercicio de panel.

Aún así, en la siguiente lista resume los comandos de panel disponibles en la version 6.0:

```
help xtides Describe pattern of xt data
help xtsum Summarize xt data
help xttab Tabulate xt data
help xtdata Faster specification searches with xt data
help xtreg Fixed-, between- and random-effects, and population-averaged linear models
help xtregar Fixed- and random-effects linear models with an AR(1) disturbance
help xtgls Panel-data models using GLS
help xtpcse OLS or Prais-Winsten models with panel-corrected standard errors
help xtrchh Hildreth-Houck random coefficients models
help xtivreg Instrumental variables and two-stage least squares for panel-data models
help xtabond Arellano-Bond linear, dynamic panel data estimator
help xttobit Random-effects tobit models
help xtintreg Random-effects interval data regression models
help xtlogit Fixed-effects, random-effects, & population-averaged logit models
```

help xtprobit Random-effects and population-averaged probit models
help xtclog Random-effects and population-averaged cloglog models
help xtpois Fixed-effects, random-effects, & population-averaged Poisson models
help xtnbreg Fixed-effects, random-effects, & population-averaged negative binomial models
help xtgee Population-averaged panel-data models using GEE

Describimos a continuación el comando más popular: xtreg, que permite estimar modelos lineales:

```
xtreg depvar [varlist] [if exp] [, re level(#) i(varname) theta ]  
  
    xttest0          /* opcional presenta el test LM de Breusch-Pagan (1980)  
                    de detección de efectos aleatorios */  
  
    xthaus           /* opcional, presenta el test de Hausman sobre  
                    efectos y errores correlacionados */  
  
Between-effects model . xtreg depvar [varlist] [if exp] , be [ level(#) i() wls ]  
Fixed-effects model . xtreg depvar [varlist] [if exp] , fe [ level(#) i() ]  
ML Random-effects model . xtreg depvar [varlist] [weight] [if exp] , mle [ level(#) i() ]  
Population-averaged model . xtreg depvar [varlist] [weight] [if exp] , pa [level(#) i() off-  
set(varname) xtgee_options ]
```

Ejemplos: véanse los programas contenidos en el directorio EJEMPL 5.

Dicho conjunto de programas replica los resultados presentados en el artículo:

-Sergi Jiménez-Martín, J.M. Labeaga y A. López (1998) "Participation, heterogeneity and dynamics in tobacco consumption: evidence from cohort data" (con J.M Labeaga y Angel López), *Health Economics*, 7, 401-414.

IX. MÁXIMA VEROSIMILITUD

Como ejemplo de estimación de máxima verosimilitud, analizaremos el siguiente programa de estimación de un modelo de decisión de visita al doctor (probit) y número de visitas (binomial negativa).

```
set more 1  
set matsize 300  
version 6.0  
tempfile junk  
clear
```

```

capture log close
prog drop _all
program define hnbreg /* llvar xbeta1 xbeta2 */
    version 6.0
    args lnf theta1 theta2 theta3
    tempvar a1 a2 a3 a4 y m
    qui {
        gen double `a3' = normprob(`theta1')
        gen double `a4' = 1 - normprob(`theta1')
        gen double `y' = $ML_y2
        scalar `m' = exp(-`theta3')
        gen double `a1' = lngamma(`m'+`y') - lngamma(`y'+1) /*
        */ - lngamma(`m') - `m'*ln(1+exp(`theta2'+`theta3')) /*
        */ - `y'*ln(1+exp(-`theta2'-`theta3'))
        gen double `a2' = ln(1- (1/(1+exp(`theta2'+`theta3')))^`m')
        replace `lnf' = /*
        */ sign($ML_y2) * ln(`a3') + /*
        */ (1 - sign($ML_y2)) * (ln(`a4')) + /*
        */ sign($ML_y2) * `a1' - /*
        */ sign($ML_y2) * `a2'
    end
    prog def doit
    local s=1
    while `s'<=2 {
        clear
        set obs 45
        gen age=_n+19
        gen hinc=(_n+4)/100.
        gen hinc2=hinc*hinc
        save pred`s',replace
        /* variables */
        log using mv_gpv`s',replace
        use sample if sex==`s',clear
        sort country
        merge country using ppp
        replace hinc=hinc/ppp93 if wave==2
        replace hinc=hinc/ppp94 if wave==3
        replace hinc=hinc/100000.
        gen hinc2=hinc*hinc

```

```

local X1="age agesq head marr sdw size nonat edu e se u pt prof cler"
local X2="serv pnsup hinc hinc2 pub he1 he2 he3 he4 jrisk hpsmix"
keep country sex dhgp hgp `X1' `X2'
save junk,replace
local l=11
while `l'<=12 {
#delimit ;
use junk if country==`l' & dhgp~=.,clear;
display "country" `l';
probit dhgp `X1' `X2' if country==`l';
mat ini1=get(_b);
nbreg hgp `X1' `X2' if country==`l';
mat ini2=get(_b);
mat iniv= ini1, ini2;
ml model lf hnbreg (dhgp =`X1' `X2') (hgp =`X1' `X2') /lnalpha if country==`l';
ml init iniv, copy;
ml max, diff ;
summ hinc if country==`l'; /*prediccion*/
local h=_result(3);
summ age if country==`l';
local a=_result(3);
use pred`k',clear; /* prediccion de visitas */
gen algp`l'=normprob(_b[eq1:_cons]+age*_b[eq1:age] + age*age*_b[eq1:agesq]
+_b[eq1:head]+_b[eq1:size]*3+_b[eq1:e]+_b[eq1:he1]
+`h'*_b[eq1:hinc] + `h'*`h'*_b[eq1:hinc2]);
gen ilgp`l'=normprob(_b[eq1:_cons]+hinc*_b[eq1:hinc] + hinc2*_b[eq1:hinc2]
+_b[eq1:head]+_b[eq1:size]*3+_b[eq1:e]+_b[eq1:he1]
+`a'*_b[eq1:age] + `a'*`a'*_b[eq1:agesq]);
gen a2gp`l'=exp(_b[eq2:_cons]+age*_b[eq2:age] + age*age*_b[eq2:agesq]
+_b[eq2:head]+_b[eq2:size]*3+_b[eq2:e]+_b[eq2:he1]
+`h'*_b[eq2:hinc] + `h'*`h'*_b[eq2:hinc2]);
gen i2gp`l'=exp(_b[eq2:_cons]+hinc*_b[eq2:hinc] + hinc2*_b[eq2:hinc2]
+_b[eq2:head]+_b[eq2:size]*3+_b[eq2:e]+_b[eq2:he1]
+`a'*_b[eq2:age] + `a'*`a'*_b[eq2:agesq]);
save pred`k',replace;
#delimit cr
local l=`l'+1
use junk,clear
gen byte c2=country==12

```

```

local X3="c2"
probit dhgp `X1' `X2'
probit dhgp `X1' `X2' `X3'
mat ini1=get(_b)
nbreg hgp `X1' `X2'
nbreg hgp `X1' `X2' `X3'
mat ini2=get(_b)
mat iniv= ini1, ini2
ml model lf hnbreg (dhgp =`X1' `X2' `X3') (hgp =`X1' `X2' `X3') /lnalpha
ml init iniv, copy
ml max
test [eq1=eq2]
log close
local s=`s'+1
end
doit
/* graphics */
use predm,clear
gen byte sexo=1
append using predf
replace sexo=2 if sexo==.
gr agegp* age if sexo==1, c(1111111111) gap(2) xlab ylab sa(g11,replace)
gr agegp* age if sexo==2, c(1111111111) gap(2) xlab ylab sa(g12,replace)
gr incgp* age if sexo==1, c(1111111111) gap(2) xlab ylab sa(g21,replace)
gr incgp* age if sexo==2, c(1111111111) gap(2) xlab ylab sa(g22,replace)
gr using g11 g12 g21 g22, sa(g1,replace)

```

X. ADO FILES

Stata tiene organizados sus comandos en ficheros ado [\backslash STATA\ADO\BASE]. Veamos un ejemplo:

```
-----\STATA\ADO\BASE\L\LOGISTIC.DO-----
*! version 3.1.4 07jan1999
program define logistic, eclass
    version 6.0
    local options `"'Level(integer $S_level)'"
    if replay() {
        if `"'e(cmd)'"~="logistic" {
            error 301
        }
        syntax [, `options']
    }
    else {
        syntax varlist [fweight pweight] [if] [in] [, `options' log *]
        cap noi qui logit `varlist' `if' `in' [ `weight' `exp'], /*
        /* `options' nocoeff nolog
        if _rc {
            if _rc!=2000 & _rc!=2001 { exit _rc }
            * if _rc==1 { exit 1 }
            logit `varlist' `if' `in' [ `weight' `exp'], /*
            /* `options' nocoeff nolog
            /*NOTREACHED*/
            exit _rc
        }
        capture global S_E_vl = _b[_cons]
        if _rc {
            di in red `"'may not drop constant'"
            exit 399
        }
        /* we pick up other e() things from -logit- */
        est local wtype `"'weight'"
        est local wexp `"'exp'"

        if (1) { /* double save in S_E_ */
            global S_E_vl `"'varlist'"
            global S_E_if `"'if'"
            global S_E_in `"'in'"
            global S_E_wgt `"'weight'"
            global S_E_exp `"'exp'"
            global S_E_ll `"'e(ll)'"
            global S_E_nobs `"'e(N)'"
            global S_E_mdf `"'e(df_m)'"
            global S_E_cmd "logistic"
        }
        est local cmd "logistic"
    }
    if `level'<10 | `level'>99 {
        local level 95
    }
    logit, or level(`level')
end
```

```

-----\STATA\ADO\BASE\LOGIS_LF.ADO-----
*! version 2.0.1 25sep1998
program define logis_lf
    version 6.0
    args todo baug lnf gr D w1 w2
    quietly {

        local t="$ML_y1"
        local t0="$EREGt0"
        local d="$EREGd"
        tempvar L l et et0
        tempname b1 lng g negb
        mlevel `l'=`baug',eq(1)
        mat `negb'=-1*`baug'
        mlevel `lng'=`negb',eq(2) scalar
        scalar `g'=exp(`lng')
        replace `l'=-`l'
        gen double `L' = exp(`l')
        gen double `et' = (`L**t')^`g'
        gen double `et0' = (`L**t0')^`g'
        mlsun `lnf' /*
            /* = `d*(`l**g' + `lng'+(`g'-1)*log(`t')) /*
            /* -(1+`d)*log(1+`et') + log(1+`et0)'

        scalar `lnf' = `lnf' + $EREGa
        if `1'==0 {exit}
        /* Gradient Calculation */
        tempvar C
        tempname g2 dbdlg dlhdlg
        /* dl/db */
        qui replace `w1'=`d**g'-(1+`d)**et**g/(1+`et')
        qui replace `w1'=`w1' + `et0**g/(1+`et0)' if `t0'> 0
        mlvecsum `lnf' `gr' =-1*`w1',eq(1)
        /* dl/dlng */
        qui replace `w2'=`d*(`l**g' +1+`g*log(`t')) /*
            /* -(1+`d)**et**(`l+log(`t'))**g/(1+`et')
        qui replace `w2'=`w2' + `et0*(`l+log(`t0'))**g/(1+`et0)' /*
            /* if `t0'> 0

        mlvecsum `lnf' `g2' =-1* `w2',eq(2)
        matrix `gr'=(`gr',`g2')
            /* HESSIAN */
            /* d2l/dbdb */
            qui gen double `C'=-(`d**`g')/(1+`et')^2
            qui replace `C'=`C' + `et0**(`g'^2)/(1+`et0')^2 if `t0'>0
            mlmatsum `lnf' `D' = `C',eq(1)

        /* d2l/dbdlg */
        tempvar xblnt xblnt0
        qui gen double `xblnt'=(`l'+log(`t'))
        qui gen double `xblnt0'=(`l'+log(`t0')) if `t0'>0
        qui replace `C'=`d**g' /*
            /* -(1+`d)**et**g*(`xblnt**g'+1+`et')/(1+`et')^2
        qui replace `C'=`C' + /*
            /* `et0**g*(`xblnt0**g'+1+`et0')/(1+`et0')^2 if `t0'>0
        mlmatsum `lnf' `dbdlg' = `C', eq(1,2)
        /* d2l/dlhdlg */
        qui replace `C'=`d**g*(`xblnt)' /*
            /* -(1+`d)**et**g**`xblnt' /*
            /* / ((`xblnt**g')+1+`et')/(1+`et')^2
        qui replace `C'=`C' + `et0**g**`xblnt0' /*
            /* / * (`xblnt0**g'+1+`et0')/(1+`et0')^2 if `t0'>0
        mlmatsum `lnf' `dlhdlg' = `C',eq(2)
        matrix `D' = -(`D',`dbdlg',`dlhdlg')
    }
end
exit

```

EJERCICIOS

Los ejercicios del curso estarán basados en los datos del directorio EJEMPLO 4, que permiten replicar los resultados del artículo Mroz (1987), *Econometrica*.

PRINCIPALES NOVEDADES EN LA VERSIÓN 7 RESPECTO A LA 6

- **by varlist**: tiene ahora una opción **sort**. La sintaxis pasa a ser: **.by varlist, sort**:
- **Formato decimal Europeo**. En la nueva versión es posible que todos los resultados usen el formato coma europeo. Basta con introducir al principio del programa: **. set dp comma**
- **Los gráficos** permiten nuevos estilos de línea y etiquetado.
- **Nuevos (o reforma) comandos de estimación**.
- **boxcox, glm, nlogit** han sido sustancialmente modificados.
- **treatreg, truncreg** son nuevos comandos de estimación. **.treatreg** estima “treatment effects models” en 2E o ML.
- **xtabond**. Estima modelos de panel de datos usando el conocido estimador de Arellano y Bond (1991).
- El cálculo de efectos marginales ha sido sustancialmente mejorado. Véase **.help mfx**. Puede computar resultados como derivadas o elasticidades.
- Las funciones estadísticas han sido revisadas y, en algunos casos sustancialmente mejoradas.
- La impresión de gráficos ha cambiado sustancialmente, véase a este respecto. **.printing, printwin, translategph**.
- El nuevo comando **.net search** permite buscar aplicaciones escritas por otros usuarios y disponibles en la red.
- **.foreach y .forvalues** mejoran el control de flujos sustancialmente.

Para mayor detalle u otras novedades ejecutar:

whatnew6to7