Notas de clase – Taller de Stata[1]

Clase 3 – Variables II

Contenido

[Estas notas se diseñaron con base en la versión 16 de *Stata*, conforme el software cambie las notas deben ser actualizadas].

[1] Estas notas están basadas en la guía desarrollada previamente para este curso por Rodrigo Azuero Melo, Nicolás de Roux, Luis Roberto Martínez, Román Andrés Zárate y Santiago Gómez Echeverry.

1. Etiquetas

Las etiquetas (en inglés *labels*) son textos que permiten describir la información que se presenta en la base de datos. Hay tres tipos de etiquetas: (i) base de datos, (ii) variables y (iii) valores. A continuación, se explica en detalle cada uno de estos tipos de etiquetas.

1.1. Etiquetas de bases de datos

La etiqueta de base de datos es una descripción de lo que contiene el archivo. Esta información no puede tener más de ochenta (80) caracteres y puede verse en la parte de datos de la pantalla de propiedades, cada vez que se abre la base de datos, o cuando se emplea el comando describe para ver el contenido de la base.

Para asignar una etiqueta a la base de datos lo único que se debe hacer es emplear el comando label data. La sintaxis requiere el comando y entre comillas la etiqueta que se desea colocar. Esta construcción es muy similar a la de los comandos que se utilizan para las etiquetas de variables y las etiquetas de valores.

Ej. 1 ⟶ label data "Información de niños menores de 6 años en Colombia"

1.2. Etiquetas de variables

Para manejar fácilmente una base de datos es necesario que las variables tengan nombres cortos. Esto impone una limitación, dado que no podemos describir toda la variable con su nombre. Las etiquetas de variables solucionan este problema. Al igual que en el caso de las bases de datos, la descripción de las variables no puede superar los ochenta (80) caracteres y pueden ser vistas en la ventana de propiedades y cuando se emplean los comandos codebook o describe ya estudiados.

El comando label var se emplea para cambiar la etiqueta de las variables. La construcción de este es el comando, el nombre de la variable y entre comillas la etiqueta que se desea poner.

Ej. 2 ⟶ label var EDI "Indicador de desarrollo psicosocial"

1.3. Etiquetas de valores

Este tipo de etiquetas se utilizan en variables categóricas, donde la variable toma un valor de acuerdo con la categoría del individuo. Por ejemplo, 0 si es hombre, 1 si es mujer. La etiqueta de

valores de una variable da una descripción de las categorías de una variable. Al crear estas etiquetas quedan almacenadas en la memoria de la sesión y pueden ser asignadas solo a variables numéricas.

Ej. 3: Por medio del *do-file* de la clase se creó la variable *departamento*, en la base de datos. Esta es una variable categórica que indica el departamento de residencia del individuo. Se sabe que, si esta variable toma el valor de 05, 08 o 11 el individuo vive en Antioquia, Atlántico o Bogotá, respectivamente. Para asignar etiquetas a estos valores se deben hacer los siguientes dos pasos:

a. Generar la etiqueta: en este paso definimos las descripciones correspondientes a cada valor y el nombre de la etiqueta, mediante el comando label define. La etiqueta en este caso se llamará "depto." Se añade la opción replace para evitar que aparezcan errores en el caso en que ya exista una etiqueta con este nombre.

   Ej. 4 ⟶ label define depto 5 "Antioquia" 8 "Atlántico" 11 "Bogotá", replace

b. Una vez creada la etiqueta, esta se le asigna a la variable deseada, mediante el comando label values.

   Ej. 5 ⟶ label values departamento depto

Para ver las etiquetas de valor creadas en la sesión se utiliza el comando label list. Los datos son mostrados con diferentes colores en *Stata*. Las variables numéricas tienen color negro, las variables de texto o *strings* tienen color rojo y las variables cuyos valores tienen etiquetas aparecen en azul.

2. Funciones para variables de texto

En notas pasadas se introdujo la función de texto substr la cual fue empleada para extraer dígitos de identificadores de encuesta, que incluían algunas variables como muestreo y vivienda, con el fin de construir un identificador de hogar. En este ejemplo, se cuenta con códigos de estudiantes y se requiere extraer el año y el semestre de entrada a la universidad:

Ej. 6 ⟶ gen ingreso_u=substr(codigo,1,5)

2.1. Mayúsculas y minúsculas

Existen otras funciones para ajustar el texto a mayúsculas, minúsculas e incluso únicamente la primera letra de cada palabra en mayúscula. Estas funciones de texto son *strupper, strlower* y *strproper*.

En este caso se quiere reemplazar los valores de la variable de texto *programa* en mayúsculas.

Ej. 7 ⟶ replace programa=strupper(programa)

En este caso se quiere reemplazar los valores de la variable de texto *programa* en minúsculas.

Ej. 8 ⟶ replace programa=strupper(programa)

En este caso se quiere reemplazar los valores de la variable de texto *programa* en mayúsculas únicamente la primera letra de cada palabra.

Ej. 9 ⟶ replace programa=strupper(programa)

2.2. Expresiones regulares (*regex*)

En algunos casos es útil trabajar con expresiones regulares debido a que son menos restrictivas que la condición de igualdad para buscar o reemplazar valores de una variable de texto. En *Stata* hay tres funciones que usan expresiones regulares: regexm, regexs y regexr y las vamos a utilizar para limpiar la variable que contiene el nombre del programa en el que está inscrito cada alumno de un curso.

- **Regexm**

La función regexm es comúnmente utilizada para buscar elementos dentro de un variable de texto. A diferencia de la operación lógica de igualdad (==), esta función identifica si la expresión regular coincide con alguna parte de las cadenas de caracteres contenidas en la variable de referencia. Por ejemplo, suponga que usted quiere visualizar todos los valores de la variable *programa* que tenga la palabra "Gob". Para esto debe especificar la variable (*var)* y la expresión regular (*reg*, entre comillas) así: *regexm(var, re)*:

Ej. 10 ⟶ browse if regexm(programa,"Gob")

Como resultado, se visualiza que el mismo programa está escrito de dos formas diferentes: "Gobierno Y Asuntos Pub." y "Gob. Y As. Pub.". En caso de no emplearse esta función habría que especificar el nombre completo del programa escrito de las dos formas, así:

Ej. 11 ⟶ browse if programa=="Gob. Y As. Pub." | programa=="Gobierno Y Asuntos Pub."

- **Regexs**

Esta función se emplea para aislar una subsección de una larga cadena de caracteres. Para esto, debe haber un patrón común en la cadena de la variable. La sintaxis del comando combina regexm y regexs. La sintaxis del comando es: *regexs(n) if regexm (var_string, ("primera subexpresión") ("segunda subexpresión") … ("tercera subexpresión"))* donde n es el dominio entre 0 y 9. La subexpresión 0 está reservada para la cadena de caracteres que satisfacen la expresión regular. Por ejemplo:

Ej. 12 ⟶ gen prueba=regexs(0) if regexm(programa,"([A-Z][a-z][a-z])")

En este ejemplo, se está extrayendo en una nueva variable *prueba* los primeros tres elementos de la cadena de carácter[2]. En este caso, la expresión regular solo tiene 1 parte, que corresponde a lo que está en el paréntesis entre comillas. Cuando la expresión regular tiene mas paréntesis se puede especificar cuál parte de la expresión regular se desea. regexs(0) toma toda la expresión regular, regexs(1) toma la primera parte de la expresión regular, regexs(2) toma la segunda parte de la expresión regular, y así sucesivamente

- **Regexr**

La estructura de la programación de la función regexr es similar a regexm. En esta función, no solo pretende buscar en una variable (*s1*) una expresión regular *(re)* sino también reemplazar dicha expresión por un texto indicado (*s2*): *regexm(s1, re, s2)*. Como ejemplo:

[2] Para estudiar otras formas de identificación de expresiones regulares puede consultar el material de poyo recomendado:
Blog sobre el uso de las expresiones regulares en *Stata*: http://soc596.blogspot.com/
Ejemplos del uso de expresiones regulares: https://stats.idre.ucla.edu/stata/faq/how-can-i-extract-a-portion-of-a-string-variable-using-regular-expressions/

Ej. 13 ⟶ replace programa=regexr(programa,"Gob. ","Gobierno ")

En este ejemplo se realiza un reemplazo en la variable *programa*. En particular, cuando en la cadena de caracteres de la variable incluye la expresión regular "Gob. " es reemplazada por "Gobierno ".

3. Ejecutar comando por cada categoría de una variable con *by* y *bysort*

Suponga que nos interesa ver las notas promedio según el año en el que se ingresó a la universidad. Lo primero que hacemos es generar el indicador de año de ingreso y explorar la tabla de frecuencias de esta nueva variable:

Ej. 14 ⟶ gen *a_ing*=substr(*codigo*,1,4)
⟶ destring *a_ing*, replace
⟶ tab a_ing

Vemos que hay estudiantes que ingresaron desde el año 2001 hasta el 2010 con excepción del año 2002. Si quisiéramos ver la nota promedio de los individuos según el año de entrada a la universidad, una opción es utilizar el condicional if para cada año de la siguiente forma:

Ej. 15 ⟶ sum *n_def* if *a_ing*==2001
…
⟶ sum *n_def* if *a_ing* ==2010

Dado que hay una gran cantidad de grupos se utilizan muchas líneas de código y el proceso puede resultar tedioso. Los comandos by y bysort facilitan esta operación. Estas le indican a Stata que debe ejecutar el comando especificado para todos los grupos de valores de una variable. Por ejemplo, el comando indicado para ver la nota definitiva promedio según el año de ingreso a la universidad es:

Ej. 16 ⟶ by *a_ing*: sum *n_def*

Esta línea de código genera estadísticas descriptivas de la variable *n_def* para cada uno de los grupos de observaciones que corresponden con los valores únicos en la variable *a_ing*. Sin embargo, antes de utilizar el comando by la base de datos debe estar ordenada según la variable que diferencia los grupos, de lo contrario Stata arrojará un error. Es decir, se debe ejecutar el

comando sort antes del comando by, de la siguiente manera:

Ej. 17 ⟶ sort *a_ing*
⟶ by *a_ing*: sum *n_def*

El comando bysort, cuya abreviación es bys, realiza esta operación en un solo paso:

Ej. 18 ⟶ bysort *a_ing*: sum *n_def*

4. Creación de variables con *egen*

El comando egen tiene la misma función que el comando gen: generar nuevas variables. Sin embargo, el comando egen permite realizar diversas y más complejas operaciones, como la creación de variables a partir de estadísticas descriptivas de otras variables, crear variables categóricas de manera expedita y por medio del ordenamiento de valores. Esto hace que el comando egen sea considerado una extensión del comando gen.

4.1. Estadísticas descriptivas

El comando egen permite generar variables cuyos valores correspondan a estadísticas descriptivas de otras variables. Entre estas, el máximo, la mediana, la desviación estándar, el mínimo, el percentil en el que está ubicado y el rango en el que se encuentra. Creemos variables que contengan algunas de estas estadísticas descriptivas para las notas de los estudiantes:

Ej. 19

Máximo ⟶ egen *nota_maxima*=max(*n_def*)
Mínimo ⟶ egen *nota_minima*=min(*n_def*)
Desviación estándar ⟶ egen *desvest*=sd(*n_def*)
Promedio ⟶ egen *promedio*=mean(*n_def*)
Percentil 75 ⟶ egen *percentil*=pctile(*n_def*), p(75)

Estas estadísticas son más útiles cuando se estudian por grupos. En este caso, los valores de la nueva variable creada serán igual para todos los individuos de un grupo. Por ejemplo, para generar una variable que contenga la desviación estándar según los grupos de año de ingreso a la universidad se puede ejecutar el comando:

Ej. 20 ⟶ bys *a_ing*: egen *desvest1*=sd(*n_def*)

4.2. Variables categóricas

Para crear variables categóricas con el comando gen es frecuente la especificación de diferentes condiciones lógicas. Una ventaja del comando egen es lo práctico que resulta el proceso de creación de variables categóricas. En una sola línea se pueden crear variables dicotómicas o de más de dos categorías, con una sintaxis sencilla.

Un ejemplo es la función cut. Cuando se especifica, *Stata* crea una variable que toma valores enteros de 0 hasta n categorías señaladas dentro del comando. Suponga que queremos agrupar a los individuos que obtuvieron una calificación final entre 0 y 1, de 1 a 2, de 3 a 4 y de 4 a 5. La manera de hacerlo es la siguiente:

Ej. 21 ⟶ egen *grupo*=cut(*n_def*), at(1,2,3,4,5)

De esta manera, se crea una nueva variable de nombre grupo que separa los grupos de la variable *n_def* según la división establecida por los valores de [, at(valor1, valor2, valor3…)]. Así, para los individuos que obtuvieron una nota final entre 0 y 1, la variable grupo toma el valor de 1, para las notas entre 1 y 2 toma el valor de 2, para notas entre 3 y 4 y toma el valor de 3 y para notas entre 4 y 5 toma el valor 5. Este comando lo podemos usar para identificar a los individuos que aprobaron la materia:

Ej. 22 ⟶ egen *aprobado2*=cut(*n_def*), at(0,3,5)

Otro uso de la función cut es la creación de grupos de igual tamaño mediante la opción [, groups]. Por ejemplo, si queremos dividir a los estudiantes en 10 grupos de acuerdo con su nota, el comando que se debe utilizar es el siguiente:

Ej. 23 ⟶ egen *grupo2* =cut(*n_def*), groups(10)

De esta manera, los estudiantes se agrupan en 10 categorías de igual tamaño cada una. *Stata* empieza calculando el rango de la variable *n_def*=max(*n_def*)-min(*n_def*)=3.2. Después, dado que queremos diez grupos, el rango se divide en 10 (0.32). Así, cada grupo tiene una amplitud de tamaño 0.32 empezando en la nota mínima (0.675) hasta llegar a la nota máxima (3.875). Los primeros nueve grupos tienen 3 observaciones cada uno y para el último hay 4 observaciones.

Otra forma de crear grupos es a través de la función *group* de *egen*. Por ejemplo, si queremos comparar el desempeño de los estudiantes por género y por semestre de ingreso, podemos crear una variable tal que cada uno de sus valores represente una combinación de género y semestre usando el comando:

Ej. 24 ⟶ egen *semestre_genero*=group(*genero2 ingreso_u*)
⟶ bysort *semestre_genero*: sum(*n_def*)

En este ejemplo, se tiene que existen 22 grupos diferentes de las combinaciones de la variable género y semestre. El primer grupo corresponde a dos estudiantes hombres que estudian Economía y que ingresaron en el primer semestre de 2007 a la universidad.

4.3. Orden

La función rank de egen también nos permite ordenar a los estudiantes en términos de su desempeño académico. En particular, en el ejemplo se ordenan a los estudiantes de forma descendente por nota definitiva, así:

Ej. 25 ⟶ sum n_def
⟶ egen top=max(n_def)
⟶ gen temp=top-n_def
⟶ egen ranking=rank(temp), track

El comando egen tiene otro gran número de funciones. Para consultarlos puede escribir *"help egen"* en la ventana de comandos. La opción *"track"* asigna 1 al valor más bajo y sigue un orden ascendente. Tenga en cuenta que el estudiante con la mayor nota definitiva tendrá el valor más bajo en la variable *temp*.

5. Recodificar variables: *recode*

Muchas veces nos enfrentamos al problema de volver a codificar una variable de acuerdo con sus categorías. Por ejemplo, la variable llamada aprobado toma el valor de 0 si el individuo no aprueba la tarea y 3 si la aprueba. Suponga que nos interesa generar una variable que se llame reprobado que tome el valor de 1 si la persona reprueba y 0 en caso contrario. El comando recode nos permite hacer esto en una sola línea:

Ej. 26 ⟶ recode *aprobado* (0=1) (3=0), gen(*reprobado*)

El comando recode es particularmente útil en el caso de *missing values*. Podemos decirle a *Stata* que todos los valores faltantes en alguno de los tres exámenes los recodifique con el valor 0.

Ej. 27 ⟶ recode e_*final parcial1 parcial2* (*missing*=0), gen e_*final*_2 *parcial1*_2 *parcial2*_2

6. Manejo de *missing values*

6.1. Generalidades missing values

En las bases de datos es importante el tratamiento que se le da a los valores faltantes o missing values. *Stata* reconoce 27 valores faltantes en la base de datos. Estos son el símbolo punto (.) que es el más frecuentemente utilizado, y todas las letras de la a *a* la *z* sin incluir la ñ precedidas por un punto (.a , .b , .c ,….., .z). Es importante tener en cuenta que los missing values son reconocidos como mayores que cualquier otro valor numérico. Por ejemplo, si queremos ver cuántos individuos pasaron el examen final, intuitivamente utilizaríamos el siguiente comando:

Ej. 28 ⟶ count if *e_final*>=3.0

Sin embargo, estaríamos contando también al estudiante con código 200411969 que no presentó el parcial y por esta razón aparece el símbolo punto en la celda correspondiente al examen final. A su vez, todos los *missing values* cuentan con un ordenamiento donde el menor es el símbolo punto seguido por las letras precedidas por un punto (.<.a<.b<.c<….<.z). Los diferentes valores posibles para los valores faltantes son útiles cuando se presentan diferentes razones por las que una variable no reporta un valor. Por ejemplo, la persona se niega a responder, la persona no entiende la pregunta, la persona no sabe, etc. En nuestro ejemplo, tenemos que en la variable del examen final hay dos valores faltantes: .a y .b. Esto se debe a que hay estudiantes que faltaron al examen final con excusa (.a) y otros que no presentaron excusa (.b). Ahora, si queremos ver cuántos estudiantes pasaron el examen final, el comando adecuado sería:

Ej. 29 ⟶ count if *e_final* >=3.0 & *e_final*<.

Podemos asignarles una etiqueta a los valores faltantes del examen final:

Ej. 30 ⟶ label define missing .a "Faltó con excusa" .b "Faltó sin excusa"
⟶ label values *e_final* missing

Al utilizar el editor o el buscador podemos verificar si la etiqueta le fue asignada a la variable e_final con éxito. Podemos ver esto más claramente con una tabla de frecuencias utilizando el comando tab. Si no se especifica nada, el comando tab muestra una tabla de frecuencias que no incluye los valores faltantes. Sin embargo, al especificar la opción [, miss] nos reporta los valores faltantes con su respectiva etiqueta:

Ej. 31 ⟶ tab *e_final*, miss

6.2. Mvencode-Mvdecode

En general, cuando se tiene información de una base de datos que no está depurada, los valores faltantes, o *missing values*, están representados con un número como 98, 99, 998, 999, etc. Por ejemplo, la variable e_final2 contiene la información sobre el examen final, pero se utiliza una codificación diferente para las personas que no asistieron ese día. El número 98 se da en el caso en que la persona no presentó excusa médica y el 99 cuando sí presentó excusa médica. Si queremos que los valores donde aparecen los números 98 y 99 *Stata* los reconozca como *missing values* el comando es el siguiente:

Ej. 32 ⟶ mvdecode e_final2, mv(98)
⟶ mvdecode e_final2, mv(99)

Esta acción cambia los valores 98 y 99 por el símbolo punto. Sin embargo, si queremos diferenciar los dos tipos de valores faltantes, se puede asignar un tipo particular de valor faltante a cada uno de estos. El comando es el siguiente:

Ej. 33 ⟶ mvdecode e_final2, mv(98=.a \ 99=.b)

Los valores donde la variable tomaba el valor 98 fueron reasignados a un valor faltante tipo .a y aquellos que tomaban el valor 99 fueron reasignados a un valor .b.

Ahora suponga que queremos recodificar los valores faltantes a un valor numérico. En este caso, queremos reasignarles a los estudiantes que no asistieron al examen final una nota de cero en el examen. El comando es el siguiente:

Ej. 34 ⟶ mvencode e_final2, mv(.a=0\.b=0)