Instituto Tecnológico Autónomo de México

Econometría Aplicada

Clase Práctica

Exámen Parcial I

Marco Antonio Ramos Juárez

142244

Ciudad de México 2020

Índice

I. Introducción

a.

Un archivo *do* se refiere a una serie de comandos que se deben ejecutar en determinada secuencia. Sirven para documentar lo que se está realizando y permiten corregir errores detectados, por ello, es muy útil agregarles comentarios. Sin embargo, en los archivos *do* no se muestran resultados ni se ejecutan los comandos. Por el contrario los archivos *log* se refieren al output de trabajo en STATA. Incluyen los registros de lo que el usuario realizó así como los resultados.

b.

Asumiendo que el archivo está en el directorio de trabajo de STATA y que el *do file* se llama "intro.do", podemos usar el comando type.

type intro.do

c.

Para poder correr un archivo do necesitamos limpiar nuestro ambiente de trabajo local en STATA. Para ello, usamos el comando *clear all* que elmina cualquier información o resultado que hayamos tenido de cualquier ejercicio anterior.

clear all

d.

Para agregar un comentario al archivo do, vamos a la fila donde queremos señalar algo y agregamos el comentario con la sintaxis correcta. Existen 3 posibilidades: a) solamente indicar un * al inicio del comentario; b) solamente indicar un // al inicio; c) delimitar el comentario entre una /* y */.

a) *se supone que la función de producción del mercado minorista es una Cobb-Douglas

b) //se supone que la función de producción del mercado minorista es una Cobb-Douglas

b) /*se supone que la función de producción del mercado minorista es una Cobb-Douglas*/

II. Descripción de bases de datos

a.

Un comando para generar estadísticas descriptivas puede ser el commando *summarize* que nos muestra un conjunto de estadísticas univariadas de cada variable, a menos que especifiquemos lo contrario (indicando solamente las variables que queremos). En general, incluyen la cantidad de observaciones, la media, la desviación estandar, el mínimo y el máximo.

b.

Puedo seleccionar la información adecuada de varias maneras, la primera (a) es indicando las condiciones directamente en el comando *summarize* con operadores lógicos; la segunda (b) manera es manipulando directamente mis datos.

a) **summarize salario if inrange(años,1995,2005) & (estados == "Nuevo León" | estados == Jalisco)**

b) **keep if estados == "Nuevo León" | estados == Jalisco**

   drop if años>2005 & años<1995

   summarize salario

c.

Para crear una variable con el valor del salario promedio puedo usar el comando *egen* junto con la función *mean*.

egen salario_promedio = mean(salario)

Asimismo podría indicar que *salario_promedio* tomara el salario promedio por estado indicando al comando egen que defina la variable por grupo.

egen salario_promedio = mean(salario), by(estado)

d.

Para crear un escalar con el valor del promedio de la variable salario podemos optar por el comando *scalar*. Esto es util pues el escalar tiene un impacto diminuto en la memoria y capacidad de nuestra computadora. Asimismo, necesito correr el comando summarize para poder llamar a los escalares asociados mostrados en su lista *return list*:

summarize salario

return list

scalar salario_promedio_escalar = r(mean)

display salario_promedio_escalar

III. Regresión simple

a.

El comando utilizado para ejecturar una regresión lineal *regress*.

b.

La sintaxis correcta para ejecutar una regresión linear con variable independeinte *mpg* (combustible en millas por galón) y como regresores el precio (precio del automóvil en dólares) y el peso (peso del automóvil en kilogramos). asumiendo que las variables dependientes se llaman *precio* y *peso* respectivamente, es:

regress mpg precio peso

c.

Por default, el comando regress muestra dos trablas de resumen que podemos omitir si modifiamos las *display_options*. De esta manera, para que la terminal solo estime la regresión sin mostrar ningún output, la sintaxis adecuada sería:

regress mpg precio peso, noconstant notable

Asimismo, una alternativa más generalizable es usar el comando *quietly* antes de otro comando para suprimir el output. De esta manera la sintaxis sería:

quietly regress mpg precio peso

d.

Asumiendo que hemos estimado la regresión del punto anterior y que no hemos estimado ninguna otra ni modifcado o cambiado de base de datos, podemos usar el comando predict indicando que queremos estimar los residuales.

predict redisOLS, residuals

e.

Para calcular la R^2 de forma "manual" tenemos que calcular la Suma de Cuadratos Explicados ($SCE = \sum_{x=1}^{n}(\hat{y} - \overline{y})^2$) y la Suma de Cuadrados Residuales ($SCR = \sum_{x=1}^{n}(y - \hat{y})^2$) de tal manera que $R^2 = \frac{SCE}{SCE+SCR}$. Existen multiples maneras para calcular la R^2, para este ejercicio emplearemos dos:

El primer enfoque que es aprovechando el catálogo *ereturn list* de valores escalares de la regresión:

ereturn list

scalar r2 = e(mss)/(e(mss)+ e(rss))

display r2

El segundo enfoque es calculando manualmente la SCE y la SCR:

predict mpg_estimado

egen mpg_medio = mean(mpg)

egen CEi = (mpg_estimado-mpg_medio)^2

sum SCEi

scalar SCE=r(sum)

predict redisOLS, residuals

egen CRi == residOLS^2

sum CRi

scalar SCR=r(sum)

scalar r2 = SCE/(SCE+SCR)

display r2

Para finalizar compruebo que ambos calculos de *r2* sean correctos usando el escalar original del R^2:

display e(r2)

IV. Análisis post-regresión

a.

Se está ejecutando la estimación de una regresión con variable dependiente *ltotexp* y variables independientes *suppins, phylim, actlim totchr, age,* y *female.* Asimismo, dicha regresión se está

estimando con errores heterocedásticos y con el output suprimido.

b.

Para realizar esta prueba de hipótesis en primer lugar debemos correr el modelo pero sin los errores robustos. Esto debido a que cualquier test que realizemos se hará con base en el tipo de error que hayamos indicado en el modelo. En segundo lugar, podemos optar por una prueba con base en el estadístico t con el comando *ttest*:

quietly regress ltotexp suppins phylim actlim totchr age female

ttest phylim==actlim

De manera alternativa podemos optar también por el comando test que realiza la prueba con base en una distribución F (recordemos que $(t_k^2 = F_{1,k})$):

test phylim=actlim

c.

Como en este caso se planteó una prueba de dos colas, rechazaríamos la hipotesis nula cuando el valor del estadístico (t o F) que nos muestre la prueba que hicimos quede por afuera del rango delimitado por los valores criticos de la distribución pertinente (t o F con sus respectivos grados de libertad) que hayamos definido para un determinado nivel de significia α.

d.

Si suponemos nuevamente que los errores del modelo son homocedásticos, entonces la prueba de hipotesis se puede hacer con el comando test.

quietly regress ltotexp suppins phylim actlim totchr age female

test pphylim actlim totchr

e.

En este sentido, como la prueba test nos indica el estadístico F, rechazaríamos la hipotesis nula cuando el valor del estadístico F quede por afuera del rango delimitado por los valores criticos de la distribución $(F_{3,n-7})$ que hayamos definido para un determinado nivel de significia α.

V. Gráficos

a.

Para realizar una gráfica de disperción usamos el comando *scatter*.

scatter x1 x2

b.

La sintaxis para generar una gráfica de densidad kernel con un *bandwith* definido (.2) y con la superposición de la densidad normal considerando como variable objetivo *x1* es:

kdensity x1, bwidth(.2) kernel(epanechnikov) normal

c.

La sintaxis para generar un histograma superpuesto por una estimación de la densidad del kernel para *x2* es:

histogram x2, freq kdensity

VI. Bucles

Nota.

Para fines de esta tarea escrita, el acento diagonal invertido (es el que usamos para acentuar en español pero señalando al lado opuesto) se representó así ‘ ; mientras que el acento vertical se representó así: ’. Se hace está nota pues STATA es muy sensible a esta sintaxis mientras que el procesador de textos que se usó para esta tarea (R markdown) no hace tan clara la distinción.

a.

Para ese ejercicio podemos usar el comando *foreach* junto con *replace*, aprovechando que podemos invocar a la variable del bucle si la ponemos entre un acento diagonal y un acento vertical. Asimismo, primero necesitamos crear la variable suma para que el bucle trabaje sobre ella.

generate suma = 0

foreach v of varlist x1 x2 x3 x4 {

replace suma = suma+‘v’

}

b.

Ahora generamos las variables *x1-x4* de nuevo aprovechando que podemos invocar a la variable del bucle con el comando *foreach*:

set obs 100

forvalues i=1/4 {

gen x‘i’=‘i’

}

Posteriormente, corremos nuestro código del ejercicio anterior y mostramos el valor de la variable *suma*, junto con el valor de su promedio.

generate suma = 0

foreach v of varlist x1 x2 x3 x4 {

replace suma = suma+‘v’

}

display suma

sum suma

De esta manera la variable **suma** se compone por 100 observaciones con valor 10 (mínimo, máximo y promedio igual a 10).

c.

De manera similar al bucle del ejercicio a, conviene definir la variable **suma2** primero para después sobreescribirla con el bucle.

generate suma2 = 0

local i = 1

while x‘i’ <5 {

replace suma2 = suma2+x‘i’

local i=‘i’ + 1

}

d.

Asumiendo que no tenía definidas las variables *xi-x4*, creo las 100 observaciones de las variables pero ahora con el comando *while*:

set obs 100

local i = 1

while x‘i’ <5 {

gen x‘i’ =‘i’

local i=‘i’ + 1

}

display suma2

sum suma2

De esta manera la variable suma se compone por 100 observaciones con valor 10 (mínimo, máximo y promedio igual a 10)

e.

En cuanto a valores no existe ninguna diferencia. La única diferencia está en la manera en que las creamos : mientras que con el comando *foreach*, puedo sumar los valores de una lista predefinida finita, en el caso del comando *while* tengo que redactar una instrucción lógica que tiene que seguir una secuencia predefinida {i}, donde tengo que indicar un inicio y un final, de lo contrario nunca terminaría.