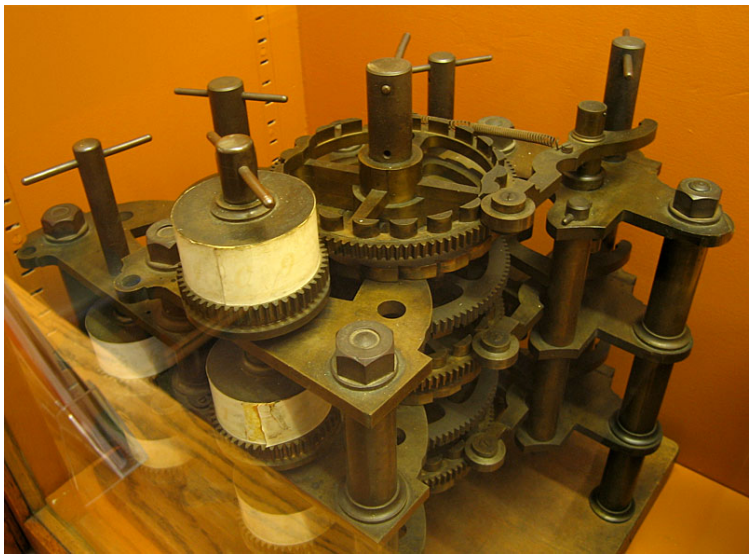
FACULTAD DE MEDICINA
UNIVERSIDAD DE CHILE

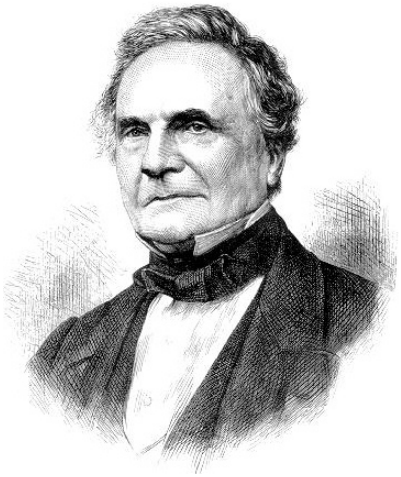
ESCUELA DE SALUD PÚBLICA "DR. SALVADOR ALLENDE G."

Administración y exploración de datos en Salud Pública usando Stata



Yuri Carvajal B.☎56-9-79598818
✉<ycarvajal@med.uchile.cl>

En la portada, máquina diferencial construida en 1991 a partir de los diseños presentados por Charles Babbage en la Royal Astronomical Society en 1822. Entre 1833 y 184 intentó una máquina analítica, contactando con Lady Ada Lovelace. Si Babbage es el padre de las computadoras y las impresoras, Lady Lovelace es la primera programadora (ver http://es.wikipedia.org/wiki/Charles_Babbage) Crédito de la imagen: Photograph ©Andrew Dunn, 5 November 2004. Website: `http://www.andrewdunnphoto.com/`



Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

El uso de programas computacionales para administrar datos se ha tornado una herramienta indispensable para el trabajo de investigación. Explorar registros de centenas de miles de casos y realizar una inspección visual de los mismos requiere destrezas sencillas, pero sistemáticas. Acciones rutinarias, cuando son realizadas por un humano siempre curioso de novedades, pueden transformarse en errores. Según Wikipedia, "Babbage intentó encontrar un método por el cual se pudieran hacer cálculos automáticamente por una máquina, eliminando errores debidos a la fatiga o aburrimiento que sufrían las personas encargadas de compilar las tablas matemáticas de la época". El problema de fondo sigue siendo el que Babbage se propuso abordar. Contamos hoy con procesadores y programas que él se afanó en producir. Aprender a usarlos es parte esencial de la producción y comunicación científica actual.

Este manual considera usos muy sencillos de un software para encarar ambas tareas y usa las herramientas gráficas como un primer acercamiento a los datos, que los examina y los interroga hipotéticamente sin forzarlos en modelos como lechos de Procusto.

Este texto surgió de las conversaciones, enseñanzas y amistad con tres estadísticos muy especiales y valiosos, ocupados de la variabilidad y de los datos: Carlos Henríquez, Sergio Alvarado y Claudio Silva. Ojalá que ello sepan que mi trabajo es un reconocimiento de lo que he aprendido de ellos y de la gratitud que les tengo. Y a Claudio, además, por su esfuerzo al revisar mis errores.

Finalmente a los alumnos de la primera cohorte del Magister de Salud Pública que sufrieron mi taller y que detectaron nuevos y viejos problemas.

Capítulo 2

Introducción a Stata

2.1. Abriendo el programa

2.1.1. La primera pantalla

Cuando abrimos Stata tenemos una pantalla dividida en cuatro zonas bajo la barra de herramientas (figura 2.1) figura 2.2.1. El primer recuadro a nuestra izquierda, llamado Review, contiene un registro de los comandos que hemos usado. Inmediatamente bajo él, las variables de la base de datos, bajo el encabezado de Variables. A la derecha, un espacio para introducir comandos en forma manual: Command. Y un gran recuadro con fondo negro en donde aparecen las salidas del procesamiento de datos. Este estilo de colores (en este caso Classic) puede cambiarse fácilmente posándose en cualquier lugar de la pantalla, apretando el botón derecho y seleccionando preferences, lo cual despliega varias alternativas de colores.
Hasta el momento todo está vacío (a excepción de la pantalla principal) pues nada hemos hecho.

2.2. Modo batch

Este tutorial va a trabajar en el modo batch , que significa que vamos a escribir cada comando que queramos ejecutar. Eso lo haremos creando un archivo do en el cual vamos a ir escribiendo paso a paso las tareas.

2.2.1. Archivo do

Vamos a abrir un archivo do que por supuesto no tiene nada escrito. Esto lo hacemos presionando el boton del mouse sobre el ícono que tiene una libreta y un lápiz (el otro que tiene un lápiz es un ícono con una tabla, pero ese es

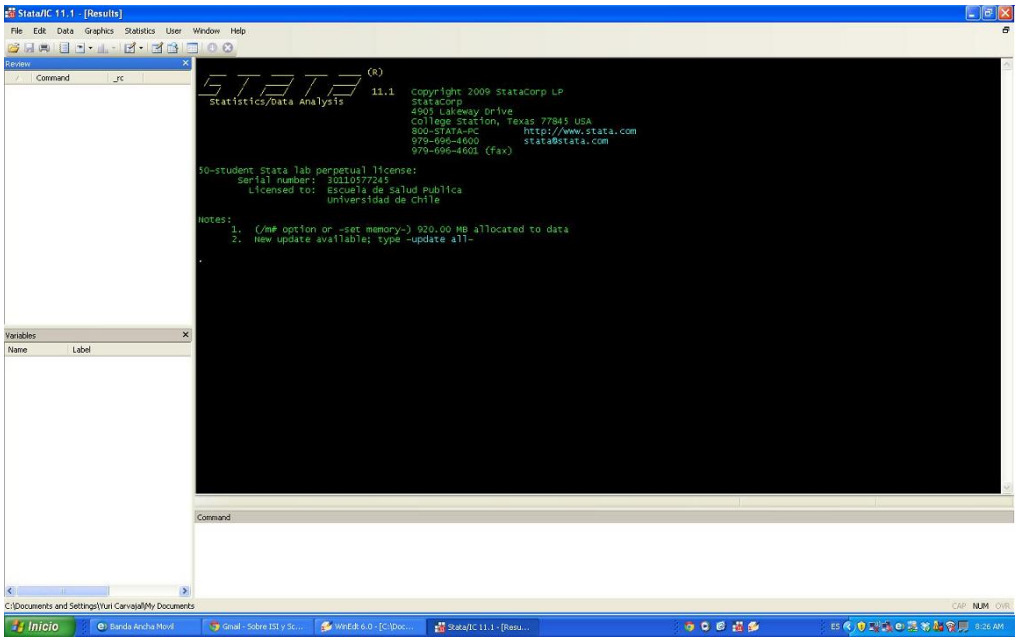


Figura 2.1: La pantalla de Stata

el editor de bases de datos). Cuando nos posamos sobre este ícono aparece desplegado el texto: `New Do-file Editor`.
Al presionar este ícono aparece un archivo como el que se muestra en la figura 2.2 2.2.1. El lugar y tamaño donde se ubica este archivo en la pantalla son, por supuesto, manipulables, pero hay algunas buenas razones para dejarlos como están..
Toda nuestra actividad con Stata la haremos escribiendo órdenes en este archivo, y ejecutando los comandos. Esto nos permitirá tener un registro de todo lo que hemos, intercambiar estos programas o códigos, documentar nuestro trabajo, homogeneizar y estandarizar el procesamiento de datos, generar trazabilidad de nuestras acciones y por supuesto, no repetir trabajos rutinarios ni acumular bases de datos modificados. De alguna manera estos archivos "do" pueden ser considerados plantillas o "templates" , lque podemos modificar de acuerdo a nuestras necesidades.

Plantilla Vamos a empezar a escribir algunos elementos preliminares en este archivo:

```
capture log close     //  cierra algun log abierto
log using manual1,replace text     //  define el log salida del trabajo

//nombre.do: manual1
```

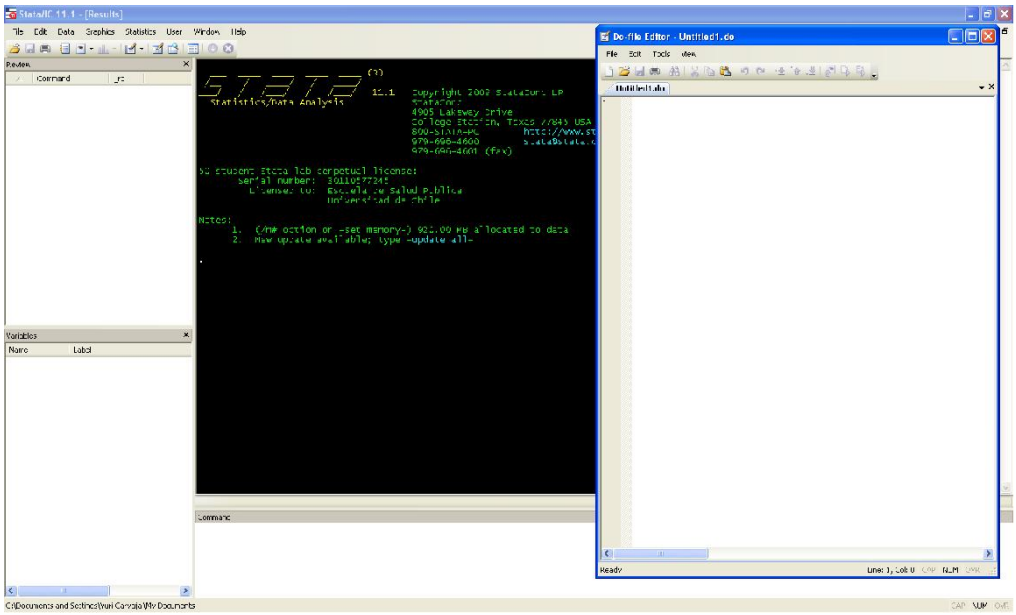


Figura 2.2: El archivo do

```
//carvajal yuri  abril2012     // para identificar el do

version 11
clear all
macro drop _all
set linesize 80    //  comandos para ordenar el trabajo

cd "C:\Documents and Settings\Yuri Carvajal\My
Documents\asegunda\1 explorabase2008"
```

El primer comando cierra cualquier archivo log abierto . El archivo log es un registro de las salidas: tablas y modelos, que luego podemos abrir con cualquier procesador de texto. En estos archivos no se guardan los gráficos, los cuales deben ser almacenados en otra forma. Partimos cerrando todo lo que pudiera estar interfiriendo. Ya vemos que los comandos tiene una acción y un sujeto. En este caso close es la acción de cerrar un sujeto log. Capture es un comando más robusto que funciona ya sea que exista o no un log abierto.

Tras escribir este primer comando uso dos slash (//)o dos backslash ( ) para poder escribir un comentario sin que Stata lo considere comando. Esta posibilidad de intercalar comentarios -que quedan en verde- permite anotar durante el trabajo tareas pendientes o dudas. Esto se lograría también antepo-

niendo un asterisco.
La segunda orden es precisamente generar un log y le damos un nombre: manual1. Puse este nombre porque a lo largo de este manual vamos a trabajar con varios archivos, que operarán con la base de datos de defunciones MINSAL Chile 2009, de nacimientos 2008 y 2009, entre otras. La labor exploratoria será documentada en un archivo log con este nombre.
La tercera anotación es más bien de orden; Aclara cómo se llama el do, quién es su autor y la fecha en que lo hicimos.
A continuación indicamos la versión de Stata en que estamos trabajando -en este caso 11-. Esto sirve para poder usar la comptabilidad de este do con versiones más actualizadas de Stata. Luego nos aseguramos que no haya nada en la memoria con `clear all` , borramos cualquier función macro que esté activa ordenando `macro drop_all` y finalmente fijamos el ancho del archivo de salida.
Ahora empezamos el trabajo, definiendo en qué directorio vamos a guardar las bases de datos, las salidas log, los gráficos y toda la actividad. El ordenamiento de los archivos y carpetas es todo un arte. Para esto hay excelentes guías que enseñan cómo ser ordenados con nuestros archivos [1],[2]. Pues si bien el orden no es un rasgo genético él puede ser aprendido y los templates sirven mucho para tornarse ordenado y dilapidar menos tiempo buscando archivos en el PC, los CD y los discos externos ¡!.
Por ahora, vamos a usar una carpeta que se llama `1 explorabase2008` y vamos a copiar la ruta , usando el explorador de Windows o el administrador de carpetas del sistema. La ruta que he escrito está tomada del administrador de carpetas, usando el texto que aparece escrito en la barra de herramientas, según muestro en la figura 2.2.1. Hemos puesto entre comillas el nombre de la ruta, pues es una forma para que Stata no se enrede con los espacios en blanco que quedan entre palabras. Los nombres que tienen varias palabras siempre deben ir entre comillas. La receta de cocina sería escriba `cd` (change directory, para los que recuerdan MS DOS) y luego dos signos de comilla pegados: . Tras eso, pegar entre las comillas el nombre.
Aprovechemos la imagen de la figura 2.2.1 para identificar los íconos de una base de datos en stata, de los archivos do y de los gráficos. Nos falta conocer el ícono de un archivo log.

2.3. Operando con bases

Luego de todo este preámbulo, vamos a empezar a trabajar con datos. Lo primero es traer la base a Stata. Tenemos, a lo menos, tres formas de hacerlo:

Usando una base que ya está en Stata Simplemente poniendo la orden de `use` seguida por el nombre de la base que debe tener el formato .dta .

Usando una base que no está en Stata En tal caso tenemos tres opciones:

Usando un programa para hacer la transferencia como STAT TRANSFER.

Usando la conectividad del sistema: Open Data Base Connectivity (odbc en panel de control, herramientas administrativas), para configurar el archivo de origen en User DSN, entonces podemos cargar en este caso los datos que por su tamaño están como archivos dbf files. Para eso, usamos el comando `odbc load, dialog(complete) dsn("dBASE Files") table("PAIS09")`.

Usando el comando `insheet using` . Este comando es muy útil para traer bases desde EXCEL . Para lo cual debemos guardar la base EXCEL como un archivo csv (coma separate values). El comando insheet requiere especificar el nombre del archivo que queremos traer con su ruta (si lo traemos desde otra carpeta) y el tipo de delimitador que usa el programa. En este caso, si es coma, la orden sería `insheet using archivo.csv, delimiter(",")`. En algunas versiones EXCEL reserva la coma para separar decimales (esto es configurable, recomiendo usar el punto) y eso significa que el delimitador usado es ;. Al traer una base en EXCEL es importante notar que las comas en stata no son separadores decimales.

Cargando los datos directamente Usando el comando input y definiendo las variables, uno puede cargar un pequeño número de datos:

```
input edad altura
12 1.64
8 1.50
end
```

Vamos a traer una base que está en dta y que nos evita tanta complicación, usando la instrucción que ya escribimos en el do `use pais2009`. El archivo do se puede correr línea a línea o en conjunto. Si marcamos todo el archivo y luego hacemos click sobre el ícono más a la derecha del do, la pantalla debe lucir como en la figura 2.3. En el mismo orden que describimos al inicio tenemos: un solo comando ejecutado, muchas variables, ningún comando ejecutado en forma manual y algunas salidas sobre el log y el cambio de ruta. Pero además tenemos el archivo do desplegado (Todo esto en modo Standard y no en Classic) y en la barra de herramientas de este archivo, en destacado, el ícono de Execute Selection(do).
Para hacer que las instrucciones escritas se ejecuten, marcamos los códigos y luego presionamos ese ícono o hacemos control D.

2.3.1. Variables

Si volvemos a reexaminar la zona de variables , veremos que hay una larga lista de ellas. En cuanto a los type , algunas son string , otras son int. Stata tiene básicamente tres grandes tipos de variables:

- String, aquellas que no tienen valor numérico, compuestas por letras, palabras o conjuntos de palabras. También códigos muy largos, que aunque

Cuadro 2.1: Tipos y características de variables numéricas

Tipo	Mínimo	Máximo	Bytes	Formato
`byte`	-127	100	1	%8.0g
`int`	-32 767	32 740	2	%8.0g
`long`	-2 147 483 647	2 147 483 620	4	%12.0g
`float`	$-1{,}7014117331901 \times 10^{38}$	$1{,}70141173319 \times 10^{38}$	4	%9.0g
`double`	$-8{,}9884656743 \times 10^{307}$	$8{,}9884656743 \times 10^{307}$	8	%10.0g

numéricos, poseen significado por partes, por ejemplo región-provincia-comuna-distrito, es preferible usarlos como string [1]

- Date: Dedicaremos un apartado a estudiar la versatilidad de Stata en el uso de las fechas.
- Numeric: reconocemos 5 tipos distintos, como señalamos en la tabla 2.1

Las variables string pueden tener un largo de hasta 244 caracteres. Es posible especificar su largo mediante una declaración de su tamaño strxx var, en que xx es el tamaño. El trabajo con este tipo de variables será mejor ejemplificado cuando empecemos a trabajar con los códigos de causa de muerte que son típicamente una variable string.
Usamos a menudo variables numéricas con valores muy pequeños. Es muy raro que necesitamos una variable double o una variable float . El conocimiento de estos detalles es importante para poder administrar bien los recursos del programa.

2.3.2. Etiquetas

De la base

Lo primero es especificar qué base de datos tenemos entre manos. Para eso asociamos a ella una etiqueta o label a la data de datos de interés: :

```
label data \"Base defunciones descargada el 2/4/2012.
Sitio web MINSAL. Formato dbf\"
```

Y luego miramos nuestra base, en forma general, usando:

```
describe
```

El despliegue en pantalla queda interrumpido y en azul aparece more , para completar el listado de las 31 variables. Si queremos que salga todo el listado (si fueran muchas variables sería un pantallazo fugaz) escribimos antes del comando.

```
set more off
```

[1] La razón es que guardar estos códigos como número implica una codificación binaria no exacta

De las variables

Muchas veces las variables requieren etiquetas para conocer mejor su contenido. Por ejemplo, en el registro de defunciones la edad está codificada a través de dos variables. Una de ellas es el tipo de edad llamada edad_tipo. Para saber qué valores tiene, escribimos:

```
tab edad_tipo
```

esto nos despliega una tabla que muestra que el 97.83 % de las defunciones pertenecen a la edad tipo 1. Esto sucede porque la edad se codifica en dos campos. El primero, toma valor 1 cuando la edad está registrada en años, 2 cuando es en meses, 3 en días y 4 en horas. Sin duda, esto debemos anotarlo para evitar errores. Usamos el comando label para crear una etiqueta de la variable . Usamos las comillas por las razones que ya explicamos.

```
tab edad_tipo
```

y agregamos una nota . Finalmente guardamos la base y la nota.

```
note: Material de la primera clase
```

Si queremos ver las notas registradas, escribimos

```
notes _dta
```

De los valores

Pero si queremos explicar qué significa cada valor de una variable, usamos entonces una definición de etiquetas para cada uno de los valores, definiendo un registro (diccionario) que, en este caso, se llamará edadt y luego indicamos en orden cada valor y su etiqueta. Y finalmente aplicamos a los valores (values) de la variable (edad_tipo) las definiciones guardadas en (edadt). Si ahora volvemos a tabular, tendremos la etiqueta de la variable y la etiqueta de los valores.
Otra forma de ver esto es mirar la base de datos. Una forma sencilla es el comando browse o br. Si no agregamos nada vemos toda la base; si agregamos algunos nombre de variables, vemos sólo las variables que nos interesan. Veamos `br edad_tipo edad_cant`. Los valores siguen siendo numéricos, pero aparecen en azul, pues tienen etiqueta. Cuando están en negro son numéricos. En rojo, son string. Este comando br permite mirar la base sin editarla. Equivale al ícono de una tabla con lupa. Si queremos escribir, usamos edit (ícono tabla con lápiz).

```
label var edad_tipo "tipifica la edad en años días meses u horas"
label define edadt 1 años 2 meses 3 dias 4 horas
label values edad_tipo edadt
```

Afortunadamente las valores a etiquetar eran sólo 4, pero etiquetar una larga lista es más trabajoso, por ejemplo los servicios de salud de las defunciones. En este caso usaremos un pequeño truco y un poco de orden. Lo primero que haremos será usar `# delimit ;` para cambiar el salto de línea usual de Stata (control return, la tecla grande a la derecha del teclado) por `;`. Con eso, hacemos nuestro listado leíble como una sola línea, que define las etiquetas de los valores correspondientes a serv ; luego ponemos un `;`, cerramos la línea y volvemos al cr (control return) como salto de línea. Entonces asignamos las etiquetas a los valores.

```
#delimit ;
label define serv
1 Arica
2 Iquique
3 Antofagasta
4 Atacama
5 Coquimbo
6 "Valparaíso San Antonio"
7 "Viña del Mar Quillota"
8 Aconcagua
9 "Metropolitano Norte"
10 "Metropolitano Occidente"
11 "Metropolitano Central"
12 "Centro de Referencia de Salud Maipú"
13 "Metropolitano Oriente"
14 "Centro de Referencia de Salud Cordillera (Peñalolén)"
15 "Metropolitano Sur"
16 "Metropolitano Sur Oriente"
17 "Del Libertador B.O'Higgins"
18 "Del Maule"
19 Ñuble
20 Concepción
21 Arauco
22 Talcahuano
23 "Bio Bío"
24 "Araucanía Norte"
25 "Araucanía Sur"
26 Valdivia
27 Osorno
28 "Llanquihue, Chiloé y Palena"
29 Aisén
30 Magallanes
33 Chiloe
;
#delimit cr
label val res_serv serv
```

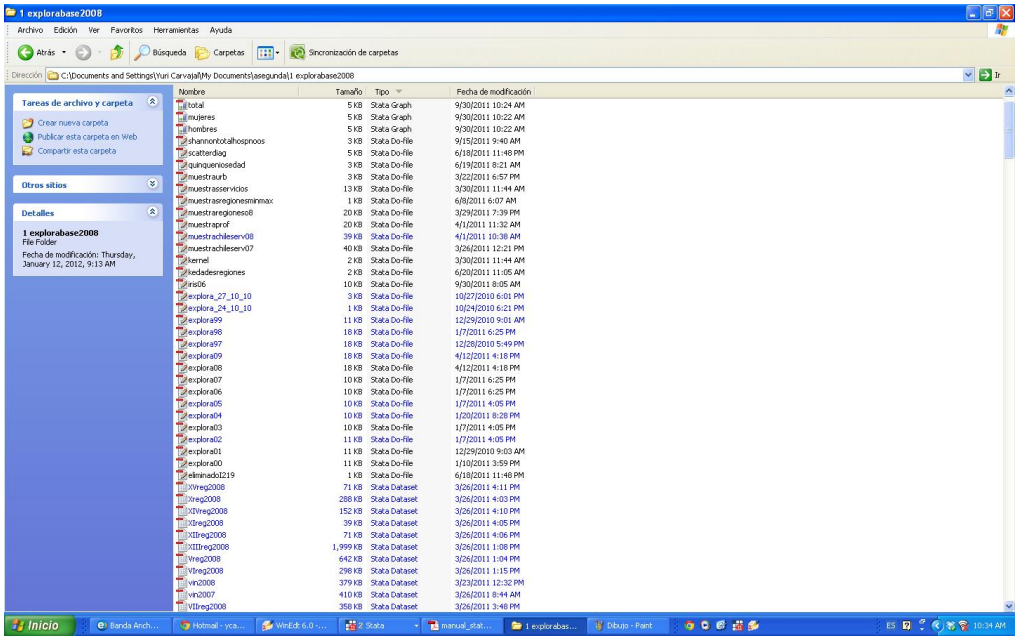


Figura 2.3: Cómo encontrar la ruta

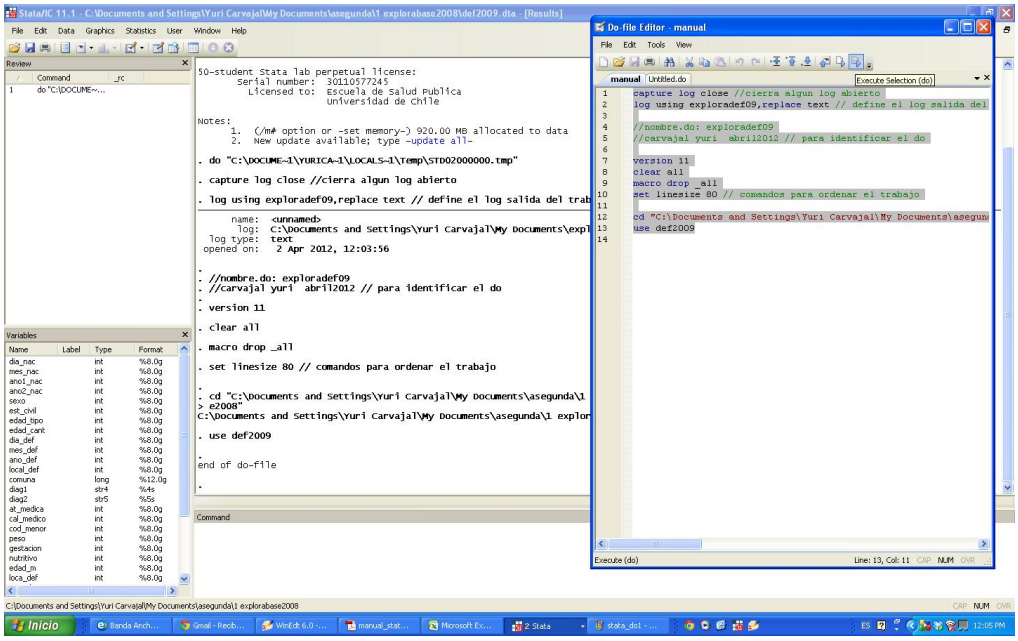


Figura 2.4: El archivo do

Capítulo 3

Exploración de datos

3.1. Herramientas generales

3.1.1. duplicates

Una primera cuestión es identificar si tenemos casos duplicados .

```
duplicates report
```

Aparecen 91959 casos únicos y 6 observaciones con 2 copias. Es decir hay tres casos duplicados. Podemos pedir un `duplicates examples` o un `duplicates list`.
Podemos marcar los duplicados mediante `duplicates tag` y por supuesto, eliminarlos mediante `duplicates drop`.

3.1.2. gen

Para hacer uso de duplicates tag hemos generado una nueva variable que toma valor 0 cuando un caso es único y 1 cuando es duplicado. Muchas funciones al no ser especificadas toman un valor por default o «por defecto». En este caso se trata de la especificación de 0 y 1 o entender que buscamos duplicados para todas las variables (o casos únicos). Si uno no quiere aceptar el default debe especificar lo que quiere. Dos lecciones se desprenden de este comando: se escribe gen y entre paréntesis se pone el nombre de la nueva variable generada, y se escriben después deuna coma. Muchos de los comandos de Stata usan esta sintaxis, en que tras la coma hay una serie de opciones que enriquecen la versatilidad del comando.
Pero gen es por si solo un comando muy poderoso, que permite generar variables en forma muy amplia. Cuando se usa como comando, es decir empezando la sintaxis, no requiere paréntesis para nombrar la nueva variable.

ordinales numéricos

Si queremos generar un número correlativo para saber de que modo están presentados los casos en la versión originaria basta que escribamos:

```
gen id= \_n
```

Existe el comando _N que genera un valor que es igual al mayor correlativo en todos los datos. En esta base de defunciones con 91 965 casos (habría que decir a esta altura con 91 962 casos) la nueva variable tomaría valor 91 965. Parece banal, pero si lo consideramos en una lista que ordene por alguna variable y haciendo subgrupos, este comando nos permite identificar los valores más altos dentro de cada grupo. Por ejemplo, listando en cada grupo los casos que cumplen la condición

```
by edad_tiponat: list edad_cant if _n==_N
```

sort y by

En esta operación hemos introducido el comando by , que es un mecanismo para ordenar los datos de acuerdo a subgrupos, en este caso grupos de edad. Elcomando by requiere que se ordenen los datos de acuerdo a la variable de creación de grupos. Eso se hace con el comando `sort` que ordena los datos de acuerdo a esa variable en orden creciente. Existe el comando gsort que permite hacer esto en orden creciente o en orden decreciente, anteponiendo un signo − a la variable por la cual se ordenará.

list

Hemos también desplegado el comando list que permite mostrar los casos de acuerdo a todas o algunas variables especificadas. Por default se listan todas las variables.

if

Especifica una condición y tiene amplia utilidad para seleccionar caso , probar consistencias y transformar variables. Si lo combinamos con la expresión «y» (que se escribe &) o la expresión «o» (—) podemos aumentar su potencia.

equal equal

El predicado de la frase usa un doble signo igual . Cuando generamos una variable usamos un solo =, pero cuando ponemos una condición a verificar, el signo debe ser doble ==.

3.1.3. search

Ahora que estamos más enterados de la sintaxis de comandos es importante saber que podemos buscar ayuda mediante el comando help ya sea a través de la barra de herramientas o tipeando help y agregando el comando o el tópico que buscamos. También podemos buscar ayuda en search , que se puede extender a la web, agregando net tras una coma o poniendo net search. O tipear findit.

3.1.4. valores perdidos

Ahora vamos a buscar valores perdidos . Vamos a mirar las fechas de nacimiento. Le ponemos etiqueta a la variable y luego hacemos un listado condicionado a que el año de nacimiento (que también está codificado en dos campos) sea cero para el día, el mes y las dos variables de año. Nos arroja un listado de 7 casos.
En esta ocasión los valores perdidos han sido escritos como 0

```
label var dia_nac "dia de nacimiento"
list ano1_nac ano2_nac mes_nac dia_nac edad_tipo
edad_cant if ano1_nac==0 & ano2_nac==0
```

Es costumbre anotarlos como 9 o 99 o 999. Es una mala práctica porque cuando se trata de variables numéricas esos missing son indistinguibles de los valores y aparecen confundidos en cálculos de medidas de resumen o de variabilidad.
Este es el caso de la atención médica, aunque se trata de una variable nominal (nuevamente está en forma de decenas y no de unidades).

```
tab at_medica
```

Stata usa «.» para codificar los valores perdidos. El punto es leído como infinito, de modo que para excluir esos valores basta escribir que considere los valores que son menores que . o if *variable* ¡. y tendremos fuera los missing. Pero es probable que algunos missing correspondan a valores ignorados y otros a no respuestas. Stata puede codificar 27 valores distintos de missing, agregando una letra minúscula al punto. En ese caso el orden de magnitud es:

```
valores no perdidos< . <.a <.b <\ldots <.z
```

Para pasar de valores numéricos (999 por ejemplo) a puntos, usamos mvdecode. El comando -transcrito del help de Stata- para familiarizarnos con su sintaxis-es:

```
 mvdecode varlist [if] [in], mv(numlist | numlist=mvc [\ numlist=mvc...])
```

Esto se lee del siguiente modo: el comando mvdecode puede aplicarse a varias variables a la vez , cuyos nombres deben seguir al comando. El comando soporta condicionales del tipo if y también in. Estas últimas permiten señalar a cuáles observaciones queremos aplicar, si sólo a los 5 primeras, anotamos textttin 1/5.

Continúa una coma y luego siguen las opciones. El agregado mv es obligatorio pues debe especificar la transformación que vamos a realizar. La opción mas sencilla es mv(\numlist) que significa poner en este caso mv(90) para llevar los casos que tienen 90 a «.». Si vamos a codificar varios tipos de missing usando las letras, entonces puede requerir especificar cada una de las definiciones. En este caso

```
mvdecode at_medica, mv(90=.a \30=.b.
```

El comando para la operación inversa es `mvencode`.

3.1.5. irecode

Con las edades nos ocurre que muchas veces debemos usar quinquenios. Una forma sencilla de resolver esto es usando el comando `irecode` .
Este comando permite crear grupos, usando puntos de corte. La interpretación del punto es menor o igual. Se adjudica el valor 0 al primer grupo y de allí hasta el final, que queda abierto.

```
 gen gedad if edad_tipo==10=irecode(edad_cant/10 ,4,9,14,19,24,29,34,
 39,44,49,54,59,64,69,74,79,84,89)
sum gedad
sum edad_cant if gedad==0
#delimit;
label define quinquenios
0 "<5a"
1 "5-9a"
2 "10-14"
3 "15-19"
4 "20-24a"
5 "25-29a"
6 "30-34a"
7 "35-39a"
8 "40-44a"
9 "45-49a"
10 "50-54a"
11 "55-59a"
12 "60-64a"
13 "65-69a"
14 "70-74a"
15 "75-79a"
16 "80-84a"
17 "85-89a"
18 "90 y mas";
#delimit cr
```

```
label values gedad quinquenios

tab gedad
```

Hemos generado 18 grupos y los hemos etiquetado. Pero sólo hemos trabajado con las edades en años. Esto significa que el grupo de menores de un año ha quedado excluído. Más adelante veremos como resolver esto expresando las edades en una sola medida. Por ahora, ilustramos este caso que permite trabajar las tablas y los gráficos de una manera más organizada.

3.2. Trabajando con los diagnósticos

La base de datos contiene códigos de CIE 10, pero no los diagnósticos especificados. Una primera cuestión tiene que ver con los grandes grupos de enfermedades o la primera letra del código.

3.2.1. string y substr

Ya sabemos que diag1 es una string de 4 caracteres. Si queremos sólo usar el primer carácter, necesitamos construirr una variable que tenga sólo el primer carácter. Eso se hace mediante una substring .

```
gen primercaracter= substr(diag1,1,1)
```

Esta orden dice que tome de la variable diag1, a partir de la columna 1, un total de 1 carácter. Esto lo puede hacer a partir del último caráacter y contar en sentido inverso. Las funciones de string son muy útiles para trabajar códigos. Recomiendo buscar en help functions y allí explorar las string functions.

3.2.2. Pegando bases

Ahora bien, cómo tener los diagnósticos uno a uno alienados frente a los códigos. Lo primero es tener una base que tenga los codigos y su correpondiente texto. Eso está disponible como una planilla excel en la web[1]. Guardamos esta planilla como csv y lo transformamos en un archivo stata. Esta base tiene una columna con los códigos alfanuméricos de la CIE 10 y otra columna donde están los diagnósticos en palabras. El nombre de diag1 debe ser común a ambas bases, pues es la columna por la cual pegaremos ambas bases. En Stata hay dos formas de pegar bases: append y merge .
Append es poner más filas en una matriz de casos. Merge es poner más columnas en el mismo tipo de matriz.

[1] hay una bastante buena en `deis.minsal.cl/deis/Estandares/CIE10_Egresos.xls`

merge

El esquema de lo que hace merge está dibujado como dos matrices que teniendo los mismos casos, agregan nuevas columnas. Por supuesto que esto debe pivotear sobre una columna en común. Pueden ser los casos o los RUT de los pacientes.

$$\begin{bmatrix} caso_1 & var_1 & \dots & var_p \\ caso_2 & var_1 & \dots & var_p \\ \vdots & \vdots & \ddots & \vdots \\ caso_n & var_1 & \dots & var_p \end{bmatrix} + \begin{bmatrix} caso_1 & var_{n+1} & \dots & var_q \\ caso_2 & var_{n+1} & \dots & var_q \\ \vdots & \vdots & \ddots & \vdots \\ caso_n & var_{n+1} & \dots & var_q \end{bmatrix} = \begin{bmatrix} caso_1 & var_1 & \dots & var_{p+q} \\ caso_2 & var_1 & \dots & var_{p+q} \\ \vdots & \vdots & \ddots & \vdots \\ caso_n & var_1 & \dots & var_{p+q} \end{bmatrix}$$

El comando requiere que ambas bases estén ordenadas (sort) por la variable de unión. En este caso no pegaremos usando pacientes ni RUT (esta base no lo incluye), sino los códigos alfa-numéricos de la CIE 10 (llamada diag1 en ambas bases). Luego, mientras tenemos nuestra base de defunciones abierta, traemos la otra desde algún archivo:

```
merge m:m diag1 using cie10
```

El comando considera que la base en que estamos trabajando es el master sobre la cual vamos a pegar una base que denomina using. Si los valores son únicos para el master y únicos en el using, entonces escribimos `merge 1:1`. Si los valores son múltiples en master `merge m:1`. En este caso hay códigos CIE 10 repetidos en los casos de la base pais09 y también en la base cie10 de códigos, por lo tanto el merge es `m:m`.
Cuando hacemos merge se genera una nueva variable `_merge` que toma valor 1 si las observaciones sólo están en el master , 2 si sólo están en using y 3 si están en ambas bases. En nuestro caso, se parearon 96 464 casos.
Luego de esto revisamos y buscamos los nombres que no estaban en el codificador nuestro por antiguedad, cambiamos los nombres y tenemos todos los casos con sus descriptores.

3.2.3. append

Append indexappend es poner más filas en una matriz de casos.

$$\begin{bmatrix} caso_1 & var_1 & \dots & var_p \\ caso_2 & var_1 & \dots & var_p \\ \vdots & \vdots & \ddots & \vdots \\ caso_n & var_1 & \dots & var_p \end{bmatrix}$$

$$+$$

$$\begin{bmatrix} caso_{n+1} & var_{n+1} & \dots & var_p \\ caso_{n+2} & var_{n+1} & \dots & var_p \\ \vdots & \vdots & \ddots & \vdots \\ caso_{n+m} & var_{n+1} & \dots & var_p \end{bmatrix}$$

$$=$$

$$\begin{bmatrix} caso_1 1 & var_1 & \dots & var_p \\ caso_2 & var_1 & \dots & var_p \\ \vdots & \vdots & \ddots & \vdots \\ caso_{n+m} & var_1 & \dots & var_p \end{bmatrix}$$

Este comando es esencial para tomar cada base de defunciones anuales y hacer una serie de tiempo por ejemplo. Hay que considerar que las mismas variables pueden tener distinta denominación de año en año y por eso deberíamos homogenizar los nombres y los valores que toman antes de pegarlas. De igual modo no aconsejo una serie de tiempo que tenga algunos diagnósticos en CIE 9 y otros en CIE 10, pues aunque hay conversores, su utilidad analítica es dudosa.
El comando es entonces

```
append using pais08
```

Con eso estamos pegando la base del 2008 a la del 2009. Tenemos que tener la base en este directorio o poner toda la ruta del modo que describimos al principio.
Este es momento de hablar de `save` . Hasta ahora todo lo que hemos hecho ha sido ejecutado en tiempo real y no hemos guardado nada. Es una excelente práctica, porque de este modo la base original está intacta y todos los cambios se pueden repetir con sólo ejecutar el do.
Pero pudiera ocurrir que queremos guardar la base con las modificaciones. No lo recomiendo, pero es posible. Entonces podemos `save` *nombre de la base*. Si queremos podemos guardarla con otro nombre o con el mismo y agregar tras un coma `save` *`nombre de la base`*`, replace` .

3.2.4. reshape

Las bases de datos pueden estar en forma wide o en forma long . Usualmente trabajamos con forma wide es decir cada línea representa un caso. Las mediciones de alguna variable en el tiempo para cada caso quedan registradas como variables en columnas distintas.
En forma long en las línea los casos aparecen tantas veces como valores repiten. La operación de reshape traspone filas por columnas.

forma long			$\leftrightarrows$	*forma wide*		
$Diagnosticos(i)$	$ano(j)$	$frecuencia$		$diagnostico(i)$	$frecuencia1$	$frecuencia2$
Iam	2008	$casos$		Iam	$casos2008$	$casos2009$
Iam	2009	$casos$		AVE	$casos2008$	$casos2009$
AVE	2008	$casos$				
AVE	2009	$casos$				

Para ir de long a wide:

```
reshape wide casos, i(diagnósticos) j(años)
```

Para ir de wide a long

```
reshape long frecuencia, i(diagnósticos) j(años)
```

3.2.5. egen

El comando egen es muy poderoso y, aunque complejo de usar, presta mucha utilidad.
Una de sus principales aplicaciones aprovecha su capacidad de trabajar a lo largo de las filas. Un ejemplo clásico es una base de datos de pacientes en forma wide, con repetidas mediciones de alguna variable de interés. Si buscamos conocer la glicemia más alta, no podemos usar el comando gen, pues éste trabaja por columnas.
En este caso, podemos simplemente generar, con egen, una variable que considere el valor menor de una lista de variables, por ejemplo de glicemias que van de glicemia1 a glicemia12

```
egen glicemiaminima= rowmin(glicemia1-glicemia12)
```

Siendo egen una herramienta muy útil, puede ser usada para obtener medias, medianas, kurtosis y un sinnúmero de funciones. Si hay alguna función que no es obvia en su forma de evaluarla, aconsejo mirar el help de , porque allí puede estar la clave.

3.2.6. collapse y contract

Si queremos tener una base de datos con el percentil 25 de las defunciones por grupo de causa según primera letra CIE 10, arreglamos la edad para tenerla en decenios, nos quedamos sólo con las edades en años, usando el comando collapse:

```
gen edad_cantn=edad_cant/1
collapse (p25)edad_cantn if edad_tipo==1, by(primercaracter)
```

¿Qué ha pasado? Si digitamos `br`, veremos que ahora hay sólo dos columnas: En la primera están los diagnósticos -primera letra- y en la segunda el percentil 25 de la edad.
Otro comando muy parecido es contract , que genera frecuencias o porcentajes. El problema con ambos comandos es que desaparece la base existente y es reemplazada por los datos solicitados. Muy útil, por ejemplo, si queremos graficar. La base original esta intacta, pero para usarla debemos volver arriba y llamar de nuevo a la base orginal.
Otra manera de resolver ese problema es con los comandos preserve y restore.

3.2.7. preserve y restore

Antes de ejecutar contract o colapse podemos pedir preserve . Este comando permite restaurar la base original usando restore , una vez que hemos usado los datos.

```
preserve
collapse (p25)edad_cantn if edad_tipo==1, by(primercaracter)
save p25,replace
restore
```

Hemos guardado en un archivo aparte la evaluación de estos percentiles.
Para lectores inquietos esto puede hacerse de mejor manera usando postfile.

3.3. Trabajando con fechas

3.3.1. tostring y destring

Vamos a calcular las edades a partir de las fechas de nacimiento y defunción y las vamos a comparar con las edades que la base informa .

Vamos a pegar los dos campos en que está codificado el año de nacimiento. Primero eliminamos la primera parte de los años si ellos son missing (drop signifca botar la variable). Luego usamos el comando tostring para llevar a string una variable numérica. ¿Por qué? Porque debemos pegar las unidades de cien y de mil con las decenas y unidades. Tostring es la función inversa de destring , que convierte de string a número.
Generamos variables nuevas que son string. Aprovechando las versatilidades de string, generamos una nueva string que pone un 0 como string (por eso las comillas) para nacidos antes del año 1910. Luego para los nacidos después de 1910 dejamos el valor original, usando el comando replace, que reemplaza los valores de acuerdo a las indicaciones. Para no tener problemas con los puntos de missing al retornar esta string a numérico, los excluimos.
Generamos una nueva string que pega la unidad de mil, la unidad de cien (+) con las decenas y las unidades, para todos los valores distintos de cero.
Finalmente transformamos esta variable string en numérica.

```
 drop if ano1_nac==.
tostring ano1_nac, gen(ano1)
tostring ano2_nac, gen(ano2)
gen ano2_2="0"+ano2 if ano2_nac <10
replace ano2_2=ano2 if ano2_nac>9 &ano2_nac <.
gen anonac= ano1+ano2_2 if ano1_nac!=0 | ano2_nac!=0
destring anonac,gen(anoac_n)
```

3.3.2. Fechas

Generamos una string que tiene el formato dd/mmm/aaaa, correspondiendo al nacimiento. Lo mismo hacemos con la defunción.
Luego generamos valores date, en formato dia mes año ("DMY") para la defunción y el nacimiento. El formato date tiene también mucha potencialidad, pues hace del tiempo un registro discreto. Esto significa que todas las fechas son cifras que tienen su origen el 1 de enero de 1969. Si contamos en días, el 2 de enero de 1960 es 2. Si contamos en meses es 1. Fechas puede contar desde milisegundos hasta años. Se pueden usar semanas, meses, trimestres, semestres, y tiene una posibilidad genérica de programar lo que se desee también). En este caso vamos a usar un formato que es de días-meses- año. Construímos ambas fechas en ese formato, las restamos, convertimos esa diferencia en su equivalente en años dividiendola por 365.25 y luego generamos la diferencia entre las edades anotadas en la base y las que nosotros calculamos.
Pedimos un listado de los casos en que la diferencia entre las dos cifras es superior a un año , Encontramos 22 casos en que las edades calculadas superan en más de un año los valores registrados y 97 casos en que las edades calculadas son menores que las edades registradas, con una diferencia de más de un año.

```
  gen fecha_nac= string(dia_nac)+"/"+string(mes_nac)+"/"+anonac
gen fecha_def= string(dia_def)+"/"+string(mes_def)+"/"+"2009"
gen date_nac=date(fecha_nac,"DMY")
gen date_def=date(fecha_def,"DMY")
gen edad= date_def-date_nac
gen edadmuerte=edad/365.25
sum edadmuerte
g dif_edad =  edad_cant - edadmuerte

list edadmuerte dif_edad anonac ano2 edad_cantn if edad_tipo == 1
 & dif_edad > 1 & dif_edad  != .
list edadmuerte dif_edad anonac ano2 edad_cantn if edad_tipo == 1
 & dif_edad < -1 & dif_edad  != .
```

El signo de exclamación antepuesto al signo igual significa diferente (texttt!=).

Al cerrar este capítulo, terminamos con nuestro do manual1 y cerramos el log, terminando el do con la expresión `log close`.

Capítulo 4

Descripción de un data set

Ya hemos mirado un poco lo que contiene un data set. Pero ahora vamos a examinarlo con nuevas herramientas. Vamos a trabajar con el do manual1.

4.1. codebook

El primer acercamiento lo hicimos con `describe`, pero ahora podemos usar codebook , que aporta más información sobre las variables, incluyendo los missing.
Podemos primero mirar toda la base con un `codebook, compact`, que va a darnos información resumida de cada variable. Considerando el tamaño de nuestra base, esto toma un poco de tiempo. Podemos pedir un `codebook edad_padre` . Aquí vemos que esta variable tiene un sólo valor, que aparece en un solo caso (la edad ya sabemos es 17 años). El resto son valores missing.

4.2. summarize

Otra forma de acercarnos a una variable es mediante `summarize` o, en forma resumida, `sum` . Mediante este comando listamos todas las variables (por default) o alguna especificada.
Sum sirve para variables numéricas y nos entrega la media, la desviación estándar, el mínimo y el máximo: `sum edad_tipo`. Si le agregamos la expresión `,detail o ,d`, tenemos kurtosis, percentiles, simetría y una riqueza de valores: `sum edad_tipo,d` Stata guarda los valores que se han desplegado como return o como stata señala son r(). Esto significa que los valores están guardados y uno puede volver a verlos mediante un `return list`. Esto que parece trivial, al combinarlo con macros (local y global), permite que uno guarde un valor por ejemplo de media o de desviación estándar y luego lo use para graficar, recurriendo al valor calculado. Copiarlo o anotarlo en una hojita puede ayudarnos a cometer errores.

4.3. Tablas

Las tablas son de gran utilidad para estudiar las variable y evaluar su consistencia. El comando `tab` es un abreviatura de `tabulate` y construye una tabla con las frecuencias y los porcentajes. Si al final del comando tras la coma ponemos `m,` nos mostrará la frecuencia de missing. El resultado de este comando es:

```
. tab servicio
```

servicio	Freq.	Percent	Cum.
Arica	1,065	1.16	1.16
Iquique	1,196	1.30	2.46
Antofagasta	2,628	2.86	5.32
Atacama	1,363	1.48	6.80
Coquimbo	3,605	3.92	10.72
Valparaíso San Antonio	3,186	3.46	14.18
Viña del Mar Quillota	6,167	6.71	20.89
Aconcagua	1,423	1.55	22.44
Metropolitano Norte	4,593	4.99	27.43
Metropolitano Occidente	5,896	6.41	33.84
Metropolitano Central	4,526	4.92	38.76
Centro de Referencia de Salud Maipú	6,866	7.47	46.23
Metropolitano Oriente	6,546	7.12	53.35
Centro de Referencia de Salud Cordiller	5,963	6.48	59.83
Metropolitano Sur	4,720	5.13	64.96
Metropolitano Sur Oriente	5,917	6.43	71.40
Del Libertador B.O'Higgins	3,054	3.32	74.72
Del Maule	3,270	3.56	78.27
Ñuble	2,147	2.33	80.61
Concepción	2,304	2.51	83.11
Arauco	4,398	4.78	87.90
Talcahuano	2,469	2.68	90.58
Bio Bío	1,588	1.73	92.31
Araucanía Norte	2,114	2.30	94.61
Araucanía Sur	529	0.58	95.18
Valdivia	981	1.07	96.25
Llanquihue, Chiloé y Palena	975	1.06	97.31
Aisén	1,425	1.55	98.86
Chiloe	1,051	1.14	100.00
Total	91,965	100.00	

```
.
end of do-file
```

Si enumeramos dos variables, STATA construirá una tabla de doble entrada. Para pedir porcentajes, habría que escribir, tras una coma, `row` , `col` o `cell` si queremos, respectivamente, el porcentaje de la fila, de la columna o de la celda en relación al total.
En este caso vamos a usar la primera letra del código, como grandes grupos,. vamos a tabular las frecuencias y los porcentajes se calculan por columnas.
Hemos usado el o condiciona (—). El slash con asterisco y luego el asterisco con slash sirven para indicar la continuidad de la línea del comando.

```
. tab2  gedad primercaracter if primercaracter=="I" |primercaracter=="J"/*
 |primercaracter=="C"> */|primercaracter=="S"|primercaracter=="K",col

-> tabulation of gedad by primercaracter if primercaracter=="I"
|primercaracter=="J" |primercaracter > =="C"|primercaracter=="S"
|primercaracter=="K"

+-------------------+
| Key               |
|-------------------|
|     frequency     |
| column percentage |
+-------------------+
```

gedad	primercaracter					Total
	C	I	J	K	S	
<5a	21	5	31	3	39	99
	0.09	0.02	0.35	0.04	1.30	0.15
5-9a	42	6	8	3	31	90
	0.19	0.02	0.09	0.04	1.03	0.14
10-14	42	12	7	4	30	95
	0.19	0.05	0.08	0.06	1.00	0.14
15-19	61	35	24	9	162	291
	0.27	0.14	0.27	0.13	5.39	0.44
20-24a	80	48	28	17	285	458
	0.35	0.19	0.32	0.25	9.48	0.69
25-29a	110	57	31	28	240	466
	0.49	0.23	0.35	0.42	7.98	0.71
30-34a	168	95	46	59	200	568

	0.74	0.38	0.52	0.88	6.65	0.86
35-39a	278	162	79	128	182	829
	1.23	0.65	0.89	1.91	6.05	1.26
40-44a	434	331	118	267	198	1,348
	1.92	1.33	1.34	3.98	6.58	2.04
45-49a	832	622	144	442	187	2,227
	3.68	2.51	1.63	6.59	6.22	3.37
50-54a	1,284	892	208	529	174	3,087
	5.67	3.59	2.35	7.88	5.79	4.68
55-59a	1,693	1,160	271	621	176	3,921
	7.48	4.67	3.07	9.25	5.85	5.94
60-64a	2,315	1,667	412	728	150	5,272
	10.23	6.72	4.66	10.85	4.99	7.99
65-69a	2,850	2,137	580	845	160	6,572
	12.60	8.61	6.56	12.59	5.32	9.96
70-74a	3,074	2,644	769	749	119	7,355
	13.59	10.65	8.70	11.16	3.96	11.14
75-79a	3,424	3,613	1,227	781	134	9,179
	15.13	14.56	13.88	11.64	4.46	13.91
80-84a	2,975	4,209	1,624	684	180	9,672
	13.15	16.96	18.38	10.19	5.99	14.65
85-89a	1,857	3,712	1,510	485	174	7,738
	8.21	14.96	17.09	7.23	5.79	11.72
90 y mas	1,086	3,411	1,721	330	186	6,734
	4.80	13.74	19.47	4.92	6.19	10.20
Total	22,626	24,818	8,838	6,712	3,007	66,001
	100.00	100.00	100.00	100.00	100.00	100.00

```
.
end of do-file
```

Cuadro 4.1: Número de defunciones por cáncer de labio y lengua. Chile 1997-2009

Año	sitio							
	Labio	%	Lengua	%	Otras de lengua	%	Total	%
1997	9	9.9	9	6.6	20	7.0	38	7.4
1998	10	11.0	8	5.9	17	5.9	35	6.8
1999	8	8.8	11	8.1	13	4.5	32	6.2
2000	4	4.4	11	8.1	24	8.4	39	7.6
2001	5	5.5	18	13.2	18	6.3	41	8.0
2002	3	3.3	15	11.0	23	8.0	41	8.0
2003	7	7.7	10	7.4	21	7.3	38	7.4
2004	6	6.6	7	5.1	22	7.7	35	6.8
2005	6	6.6	9	6.6	27	9.4	42	8.2
2006	7	7.7	14	10.3	23	8.0	44	8.6
2007	7	7.7	7	5.1	25	8.7	39	7.6
2008	10	11.0	13	9.6	25	8.7	48	9.3
2009	9	9.9	4	2.9	29	10.1	42	8.2
Total	91	100.0	136	100.0	287	100.0	514	100.0

Fuente: Datos DEIS Minsal

Estas tablas pueden tener también pruebas estadísticas de independencia estadística.
Las tablas se benefician ampliamente del comando by y de los condicionales . Pot ejemplo podemos hacer tablas por servicio de la edad de las personas que fallacieron de H1N1:

```
bysort servicio: tab edad_cantn if diag1=="J09X"
```

El comando `bysort`, ordena (sort) y hace la función de by.

Tabout

Ian Watson[3] ha realizado un excelente trabajo al programar el comando tabout en STATA, que funciona permitiendo salidas en LaTeXde tablas de frecuencias, estadísticas de resumen, medias con intervalos de confianza a partir de survey. Las celdas de la tabla pueden ser manejadas en forma sencilla, usar comas en decimales, insertar símbolos comunes. Imprescindible para quienes trabajan en estadísticas.
A continuación mostramos una tabla desarrollada mediante tabout. Usa además los paquetes bookstab y tabularx. El traslado automático desde STATA es sencillo y se puede estudiar en [3].

4.4. assert

El comando `assert` es una buena ayuda para chequear coherencia en los datos. Podríamos suponer que todas las muertes por cáncer de próstata son hombres. Entonces podemos verificar si esa suposición es verdadera o falsa.

```
assert sexo==1 if diag1=="C61X"
```

4.5. format

El comando format `format` especifica cuantos caracteres y decimales se muestran de los valores de una variable, el uso de comas para separar miles y su alineamiento. Cuando miramos el comando variables, después de Type está Format. Aparecen los de string como `str1` y `str4` y las numéricas, como %8.0g o %9.0g. El formato (textit %fmt) de todas las variables numéricas son variaciones del tipo más común denominado formato %f, que se especifica como %#.#f o %w.d**f** , en donde w es un número que cuantifica el ancho total de la salida, incluyendo signo y punto decimal y d es el número de digitos a la derecha del punto decimal. El número queda justificado a la derecha.
El comando format permite pedir que Stata muestre formatos o fijarlos. En este último caso, tras el comando y las variables que vamos a fijar con ese tipo, debemos especificar el %fmt que deseamos usar y la cantidad de dígitos.

4.6. Programación básica

Toda el cúmulo de acciones que hemos visto es en cierta forma programación en Stata. La forma más sencilla de confeccionar un programa es escribiendo una secuencia de acciones dándole un nombre. Esto nos permite transformar esa secuencia en un comando que ejecuta todas las acciones de una sola vez sobre distintas bases. La ayuda de program es un buen camino para acercarse al problema.

4.6.1. macros

Las macros son formas de guardar información en forma resumida. Una macro puede guardar varias variables, de modo que luego las llamemos sólo escribiendo el nombre que le dimos en la macro, puede guardar medidas de resumen (las que devuelve el return por ejemplo) o una serie de especificaciones (especialmente útil en gráficos).

local

Podemos calcular, por ejemplo, la media de una variable y guardarla para insertar una línea en un gráfico o para mostrarla.

```
summarize edad_cantcorr if edad_tipo==1
local media_edad= r(mean)
display "La media de edad es:‘media_edad’ años"
```

lo que obtenemos es

```
La media de edad es:70.33251822870354 años
```

Al momento de calcular los valores de summarize le indicamos a Stata que guarde el valor de r(mean) como una macro local que se llama `media_edad`. Cuando llamamos a esa macro usamos el acento grave y el apóstrofe (usalmente cerca del abrecorchete y bajo el signo de interrogación, respectivamente). Las macro local corren sólo en su do, cuando pasamos a otro, ya no funcionan. Al correr el do línea a línea tampoco se mantienen. Si guardamos variables mediante una macro, podemos hacer:

```
La media de edad es:70.33251822870354 años
```

global

Las macro global pueden funcionar en diferentes do. Se nombran con un signo $.

```
summarize edad_cantcorr if edad_tipocorr==1
global de_edad= r(sd)
display "La desviación estándar de edad es:$de_edad"

La desviación estándar de edad es:18.33178180527749
```

Para resolver las cifras decimales:

```
summarize edad_cantcorr if edad_tipocorr==1
global de3d_edad= string(r(sd), "%8.3f")
display "La desviación estándar es $de3d_edad"
```

que nos da:

```
La desviación estándar es 18.332
```

Por supuesto que esto puede hacerse de igual modo para las macro local.

4.6.2. loops

A partir del dominio de las macros, es posible entrar en dos comandos que permiten realizar acciones iterando (también se pueden hacer mediante while, pero puede tener problemas). O lo que se conoce como loops. Básicamente hay dos tipos de loops: para variables y para valores.

forvalues

Cuando yo quiero hacer una misma tarea sobre distintos valores numéricos de una variable, creo una macro que va incluir esos diferentes valores y luego realizo la acción mediante el comando forvalues . El loop se abre con una llave después del forvalues ({)y se cierran con la llave también (}).
Si quiero tener el promedio de edad en años 8 es decir matyores de un año, par cada región, genero una macro que llamo i, que ya sabemos va de 1 a 15 y ordeno que vaya de valor en valor, desde 1 a 15, realizando esa operación summ. Para saber a qué región corresponde la cifra, le pido que haga un display de la macro. Antes genero una nueva variable que corresponde a la edad sólo de los mayores de un año de cada servicio. dado que pongo la macro al final del nombre, cada variable va a tomar el nombre edad_cant y el número del servicio.

```
 clear
use def2009
tab res_reg
forvalues i=1(1)4{
gen edad‘i’=edad_cant if edad_tipo==‘i’
 summarize edad_cant‘i’
}
```

Luego calculo los promedios por servicio, mediante:

```
forvalues i=1(1)15{
summarize edad‘i’ if res_reg==‘i’&edad_tipo==1
display ‘i’
}
```

Hemos realizado nuestra primera iteración, usando forvalues. Este ejemplo se podría haber realizado mediante `bysort:`, pero esta forma parece más prolija. El gran valor de este comando está en los muestreos, en los cuales puede tomar n muestras de igual tamaño y calcular parámetros en cada una de ellas.

foreach

El loop foreach es my valioso ya que puede trabajar con una lista de variables. Dado que el formato de las variables es muy largo, vamos a reducir su tamaño y ahorrar memoria. Luego mediante otro loop las convertimos en int y finalmente hacemos los gráficos de kdensity mediante un loop, los guardamos y exportamos Lo que hacemos es que, para cada una de las variables que están alteradas, generamos una nueva variable que tendrá la expresión corr tras su nombre, para indicar que están corregidas. Entonces generaremos valores que son la décima parte de los valores originales. Feres permite ver como se va realizando la iteración por cada variable.

```
foreach v of varlist dia_nac-activ{
display ‘"ferest() is |‘ferest()’|"’
```

```
format ‘v’  %2.0g
}

foreach v of varlist dia_nac-activ{
display ‘"ferest() is |‘ferest()’|"’
 recast int ‘v’
}

foreach v of varlist dia_nac-activ{
kdensity ‘v’,saving(kd‘v’)
graph export kd‘v’.png
}
```

4.6.3. ado files

Existe otra forma de programar qur consisten en crear un programa y hacer de él un comando. Esta alternativa nos permite crear un help de la misma factura que los que despliega Stata para ayudar en su uso.
Antes de crear un programa de este tipo, verifique si ya ha sido desarrollado por otra persona. Si no es así, adelante!!.
Un programa de este tipo tiene la extensión ado . Lo puede guardar en su Stata o subirlo a un sitio desde donde pueda ser descargado.

4.6.4. mata

Finalmente un tipo diferente de programación se puede realizar usando mata . Es un subprograma de Stata que trabaja con álgebra matricial. Esto permite hacer cálculos complejos y específicos de manera mucho más rápida.
Por ejemplo, el modelo de regresión lineal usando la base auto.dta, puede ser ejecutado mediante:

```
clear all
sysuse auto.dta
mata
st_view(y=., ., "price")
st_view(x=., ., ("mpg", "length","displacement"))
b= invsym(x’x)*x’y
e= y-x*b
n= rows(x)
k= cols(x)
s2=(e’e)/(n-k)
v= s2*invsym(x’x)
se= sqrt(diagonal(v))
(b, se, b:/se, 2*ttail(n-k, abs(b:/se)))
end
```

La primer columna nos da el el vector de βs estimados, la segunda la estimación de su desviación estándar, la tercera el valor t asociado y la última el p value.

Capítulo 5

Gráficos para describir y explorar datos

Los gráficos en Stata y particularmente en modo batch parecen ser fatigosos. Pero el esfuerzo de prepararlos mediante códigos de programación es ampliamente recompensado por las ventajas de poder manejar muchos detalles de una acción orientada. El valor de los gráficos es esencial en la producción científica. Un buen gráfico de puntos (scatter) ó una caja con bigotes (boxplot) ó un histograma de frecuencias, aclaran muchas dudas y hacen que las controversias se resuelvan con sencillez y elegancia. El manual preparado por Mitchell es un texto indispensable [4]. Tiene la ventaja de desplegar todos los gráficos y a la vez escribir los comandos, además trae un índice de temas de primera calidad. En lo personal he sacado mucho provecho de su estudio. Es un libro para tener siempre a la mano.
Sobre cuestiones más teóricas sobre gráficos, recomiendo estudiar material muy valioso y entretenido en [5], [6], [7], [8], [9]. No desperdicie la posibilidad de describir sus datos usando gráficos para expresar las seis dimensiones básicas de un conjunto de datos [10]:

1. Forma
2. Posición
3. Dispersión
4. Outlier
5. Conglomerados
6. Granularidad

Pero también desde el punto de vista analítico, un gráfico que asocie dos variables a través de puntos, líneas permite encontrar patrones, irregularidades, tendencias, estructuras, asociaciones. El sitio de Edward Tufte aborda estos

problemas con sentido además estético y es un autor esencial para empezar a pensar estas materias [11], sobre todo por su énfasis en la sencillez gráfica de la información.
Usar un software para producir los gráficos, haciéndolos en modo batch, permite construir una plantilla personal de fondos, definir colores y proponer un estilo dentro de los márgenes aceptables en la escueta comunicación científica. Preparar los gráficos mediante un archivo do permite pensarlos, re-pensarlos y luego re-escribirlos. Dada la importancia de los gráficos en el debate científico, no pueden ser librados a la improvización o a la deriva de una mano clickeando sobre el mouse. Eso sólo implica que es el software quien decide. Stata ejerce cierta resistencia sobre nuestra acción intelectual, mediante consideraciones de la lógica de la investigación, sensaciones manuales y conceptuales.
STATA puede guardar manualmente los gráficos como png. Si los realizamos automáticamente mediante un comando do en modo "batch" lo exportamos desde gph a gpn.

5.1. Una variable

Si queremos explorar la distribución de una variable, lo ideal son las funciones kernel y los histogramas de frecuencia.

5.1.1. kdensity

```
kdensity edad_cantcorr if edad_tipo==1,saving(kd1)
graph export kd1.png
```

La función kernel nos permite explorar la simetría y tener una visión de un ajuste muy suave a los datos. Hemos agregado un comando de guardar el gráfico, para poder trabajar con él. Además la función `kdensity` permite mediante una opción tras las comas, generar una variable en las x y en las y (d) en donde almacenar los puntos de la curva, de modo que si bien uno no puede graficar inmediatamente otras variables, mediante un sencillo arreglo podemos lograrlo. Lo veremos en los gráficos de dos variables.
Hemos agregado un comando de exportación de gráficos a formato png para poder usarlos en forma más sencilla en este texto y en presentaciones.

5.1.2. histogram

Esta misma exploración se puede hacer con un histograma de frecuencia, que se construye con el comando `histogram`. Si la ponemos tras una coma normal, nos dibuja lo que sería una distribució gaussiana. Esta orden funciona también con la función kernel.

```
histogram  edad_cant if edad_tipo==1,saving(h1)
graph export h1.png
```

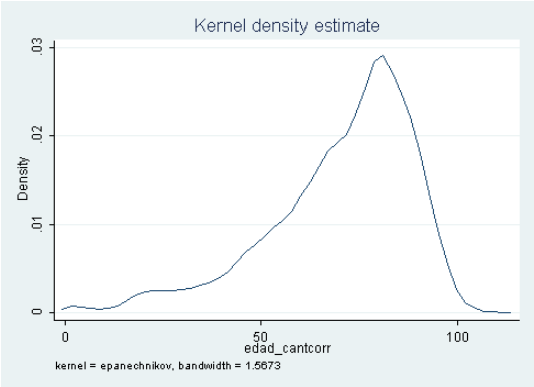


Figura 5.1: Función kernel para las edades en años

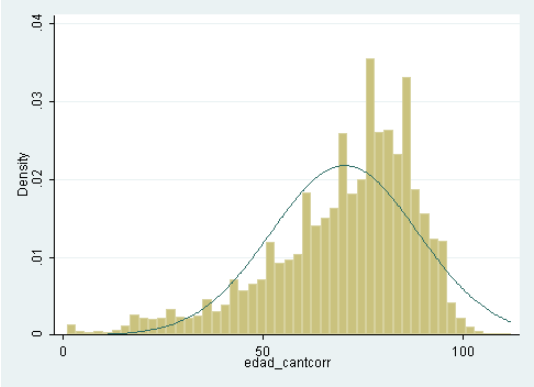


Figura 5.2: Histograma de la edad en años con curva de Gauss

```
histogram  edad_cant if edad_tipo==1,normal saving(hn1)
graph export hn1.png
\section{dos o mas variables}
```

5.2. pnorm y qnorm

En la búsqueda de distribuciones gaussianas, los pnorm y los qnorm son de utilidad. El primero grafica la distribución de probabilidad de los datos versus una distribución gaussiana estandarizada.

El segundo grafica los quantiles de los datos versus los quantiles de una distribución gaussiana. En ambos casos un buen ajuste sería que los datos coincidieran en la diagonal del gráfico.

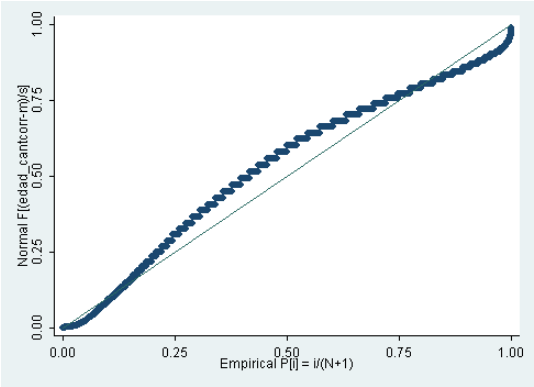


Figura 5.3: Gráfica de pnorm de edad cantidad

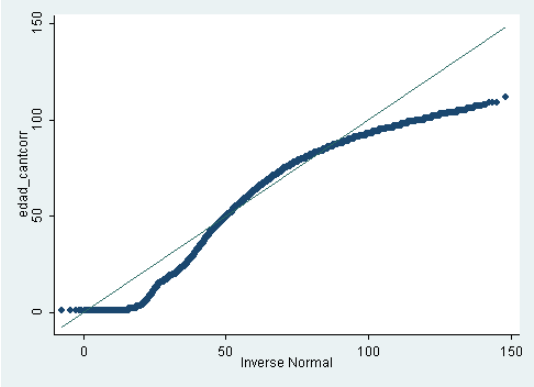


Figura 5.4: Gráfica de qnorm de edad cantidad

5.3. Una variable entre grupos

Comparar una variable en dos grupos es fácil con los gráficos de caja o caja con bigotes.

5.3.1. graph box

El gráfico de cajas muestra quartiles y valores outliers para una variable. Si tenemos dos grupos, permite establecer comparaciones.
En este gráfico comparamos los pesos de los niños fallecidos en los servicios de salud agrupados por zonas. Hemos usado el comando `recode` para esa tarea y hemos puesto label. Veremos que el gráfico despliega en el correspondiente eje las etiquetas y no el nombre original de la variable.
La línea de en medio marca la media, los bordes de la caja los percentiles 25 y 75. Las líneas horizontales que rematan las líneas verticales, son los últimos

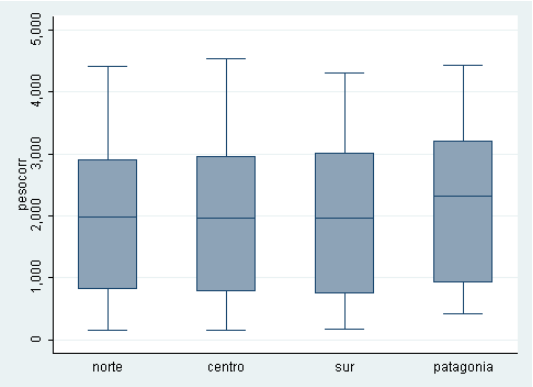


Figura 5.5: Box Plot de pesos de los niños fallecidos por zonas

valores adyacentes. Cuando existen outliers se grafican como puntos mas allá de esas líneas. Si escribimos hbox, entonces las cajas se dibujan verticalmente.

```
recode res_servcorr (1/5=1) (6/16=2) (17/23=3) (18/33=4),gen (zonas)
label define zt 1 norte 2 centro 3 sur 4 patagonia
label values zonas zt
graph box peso if edad_tipo>3 &peso>0, over(zonas) saving(bp1)
graph export bp1.png
```

5.3.2. dotplot

Dijimos que el gráfico boxplot puede ser limitado si la distribcuión tiene modas. El gráfico `dotplot` es muy útil para identificar distribuciones multimodales, además de asimetrías.

```
dotplot peso if edad_tipo>1 &peso>0, over(zonas)saving (dp1)
graph export dp1.png
```

5.3.3. kernel con dos variables

Si queremos comparar en un mismo gráfico las funciones kernel de dos grupos, entonces guardamos cada una de ellas y luego las graficamos juntas. Por ejemplo, para dibujar la curva de densidad de las edades de defunciones de hombres y mujeres, graficamos la kernel de los hombres y generamos mediante el comando `generate` dos variables nuevas que guardan los puntos de esa kernel. El primero (en este caso `hx` guarda los puntos en el eje x y el segundo (`hd`)las densidades.
Hacemos lo mismo con las mujeres. Y finalmente graficamos mediante líneas, las dos variables. Para ordenar el comando hemos partido por las variables de

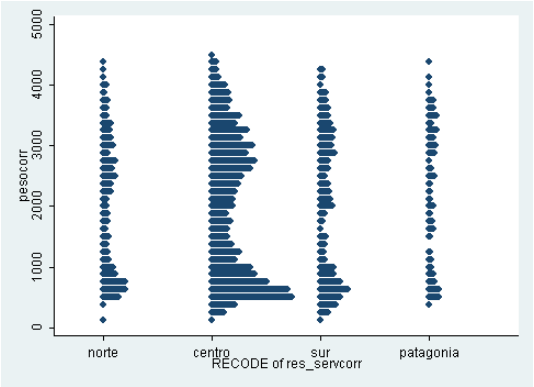


Figura 5.6: Dotplot de pesos de los niños fallecidos por zonas

densidad pues son las que van en el eje y. Nótese que stata siempre (en las regresiones principalmente) considera como variable resultado la primera que anotamos y la grafica en las y.

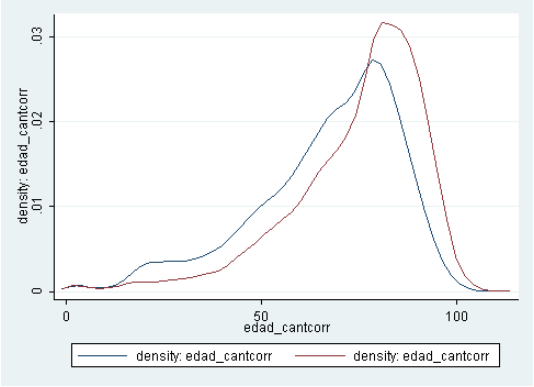


Figura 5.7: Funciones kernel de las defunciones de hombres y mujeres Chile 2009

```
kdensity edad_cant if edad_tipo==1 & sexo==10, generate( hx hd)
kdensity edad_cant if edad_tipo==1 & sexo==20, generate( fx fd)
twoway (line hd hx) (line fd fx),saving(kd2)
graph export kd2.png
```

5.4. Asociación de dos o más variables

5.4.1. scatter

Mediante gráficos de puntos podemos graficar la asociación entre dos variables: La edad de gestación y el peso de los fallecidos.

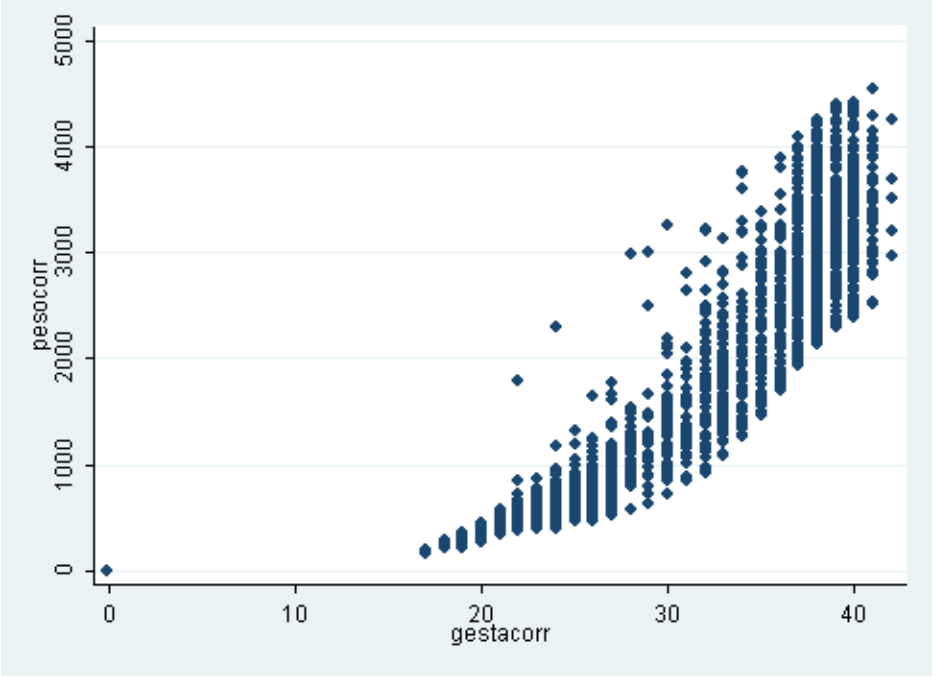


Figura 5.8: Gráfico de puntos del peso al nacer de los niños fallecidos y la´edad gestacional

```
gen gestacorr= gestacion/10
scatter peso gesta, saving(sc1)
graph export sc1.png
```

Más de dos variables

Gráficos de puntos con by Para graficar más de dos variables, es posible usar el comando by , de modo de tener un despliegue visual de varios gráficos a la vez.

```
scatter peso gesta, by(zonas) saving(scz1)
graph export scz1.png
```

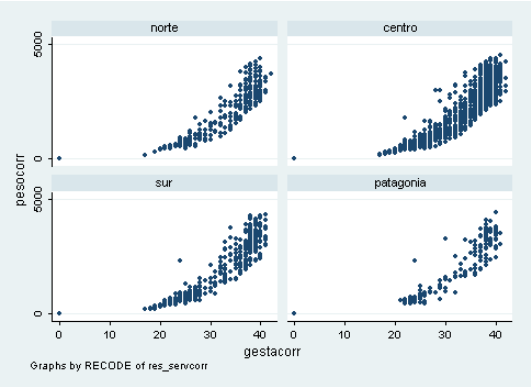


Figura 5.9: Gráfico de puntos del peso al nacer de los niños fallecidos y la edad gestacional, por zonas

Gráficos de puntos manipulando los marcadores Otra alternativa para incorporar más variables a un gráfico de puntos es usar marcadores de colores diferentes.
Graficando de la base de defunciones del 2009, el peso y la edad gestacional de los fallecidos en los Servicios de Salud de Aconcagua y Aysén. Tambien podemos

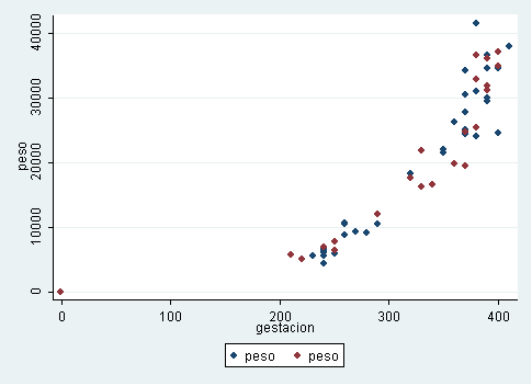


Figura 5.10: Gráfico de puntos del peso al nacer de los niños fallecidos y la edad gestacional.Aconcagua y Aysén

manipular la forma y el tamaño de los marcadores . La expresión Oh significa círculo vacío, Th triángulo vacío, S es cuadrado, el signo + y la x se grafican como tales (ver symbolstyle en `marker_options` dentro del help de `graph two way scatter`.

```
clear
use pais09
```

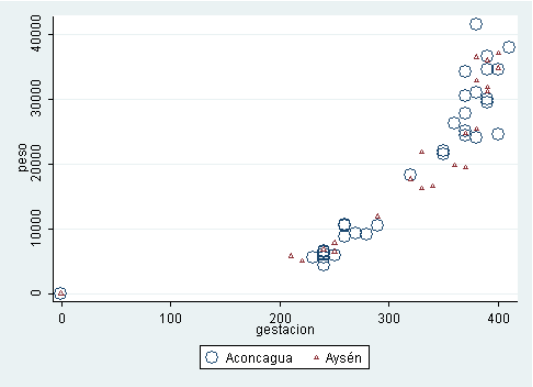


Figura 5.11: Gráfico de puntos del peso al nacer de los niños fallecidos y la edad gestacional.Aconcagua y Aysén

```
tab res_serv

twoway (scatter peso gestacion if res_serv==8)/*
*/(scatter peso gestacion if  res_serv==29),saving (scac)
graph export scac.png

twoway (scatter peso gestacion if res_serv==8, msymbol(Oh)msize (vlarge))/*
*/(scatter peso gestacion if  res_serv==29, msymbol(Th)/*
*/msize(small)), legend(cols(2)/*
*/ label(1 "Aconcagua") label( 2 "Aysén")) saving (scash)
graph export scash.png
```

Gráficos de puntos en matriz Una buena alternativa en la exploración de los datos es graficar las variables que suponemos asociadas en forma de matriz . Usando los nacimientos del año 2009, graficamos las variables semanas, peso, talla y las edades de la madre y el padre. Este gráfico puede ser comprendido como una matriz de gràficos de correlaciones entre pares de variables. Usamos el comando half tras la coma para hacer sólo la mitad de la matriz.

```
graph matrix semanas  edad_m edad_p peso talla, half saving(hmatrix)
graph export hmatrix.png
```

5.4.2. otros gráficos

Stata posee funciones para construir otros tipos de gráficos. Daremos ejemplos de cada uno de ellos.

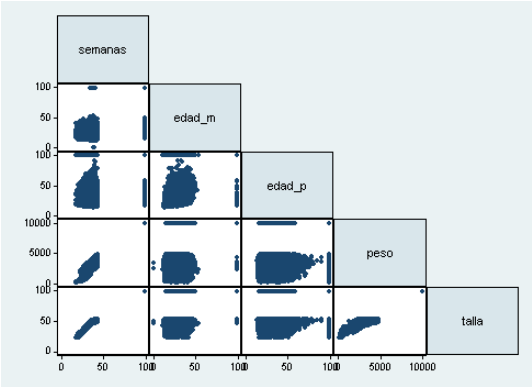


Figura 5.12: Matriz de puntos de edad gestacional, edad de la madre y del padre, peso y talla de los nacidos el año 2009

Barras

El gráfico de barras nos permite hacer una comparación visual muy sencilla de valores. Para tener una visión de los pesos de los recién nacidos por sexo y zona usamos el over después de la coma, pudiendo acumular varios over por gráfico. En este caso lo que graficamos es el promedio de valores de peso. Recodificamos nuevamente los servicios (estamos trabajando con otra base de

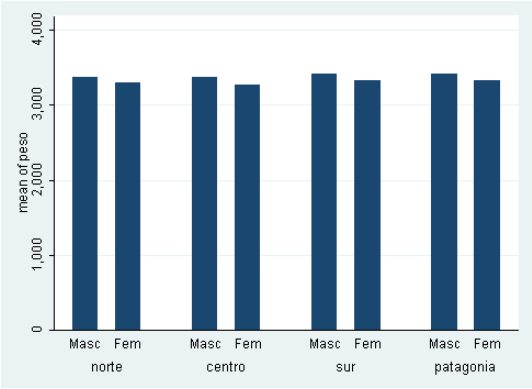


Figura 5.13: Gráfico de barra de peso y sexo de los recién nacidos por zonas

datos), etiquetamos las variables a graficar y procedemos:

```
recode serv_res (1/5=1) (6/16=2) (17/23=3) (18/33=4),gen (zonas)
label define zt 1 norte 2 centro 3 sur 4 patagonia
label values zonas zt
graph bar peso, over(sexo) over (zona) saving(b2)
graph export b2.png
```

Barras horizontales Si usamos el comando `hbar`, la gráfica se hace horizontal . En este gráfico detectamos que en la zona norte hay algunos nacimientos masculinos con valor 0 en la codificación urbano rural; lo verificaremos mediante un tab.

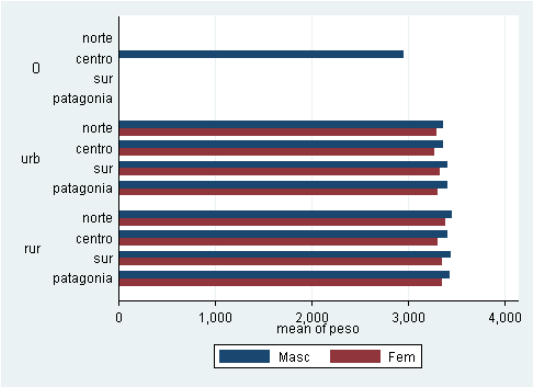


Figura 5.14: Gráfico de barras horizontales de peso y sexo de los recién nacidos por zonas y área urbano rural

```
graph hbar peso, over(sexo) over (zona) over(urb_rural) saving(b3)
graph export b3.png
```

Podemos usar el comando `stack` para poner sobre la misma barra los valores de peso según la variable sexo.

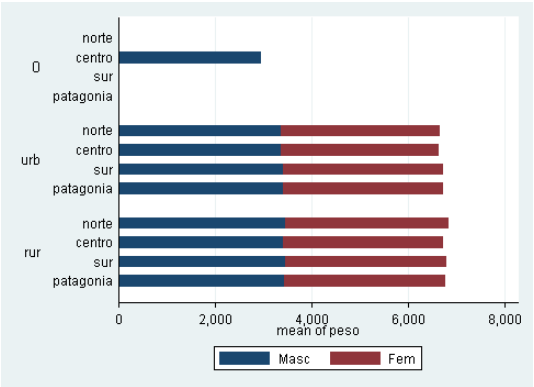


Figura 5.15: Gráfico de barra de peso y sexo de los recién nacidos por zonas y areas urbano rural

```
graph hbar peso, over(sexo) over (zona) over(urb_rural) stack saving(b4)
graph export b4.png
```

spike

Los gráficos de espiga permiten diferenciar valores que están por sobre o debajo de un umbral . En este caso colapsamos los valores de peso de los recién nacidos por servicio y luego los graficamos, fijando el promedio como base. Además escribimos en el eje de las x las etiquetas y no los nombres de las variables, para lo cual rehicimos unas etiquetas acortadas de los servicios y giramos en 90 grados su ángulo de aparición en el gráfico.

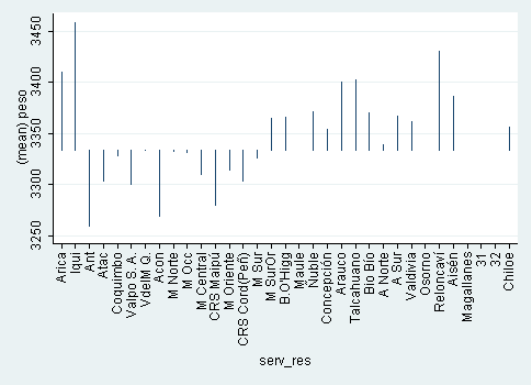


Figura 5.16: Peso de los recién nacidos por Servicios, comparados al promedio nacional

```
twoway spike peso serv_res, xlabel(1(1)33, valuelabels angle(90))/*
*/base(3333.276) saving(sp)
graph export sp.png
```

dropline

El gráfico de dropline es muy parecido, pero tiene más posibilidades de diseño:

```
twoway dropline peso serv_res, xlabel(1(1)33, valuelabels angle(90))/*
*/base(3333.276) saving(dl)
graph export dl.png
```

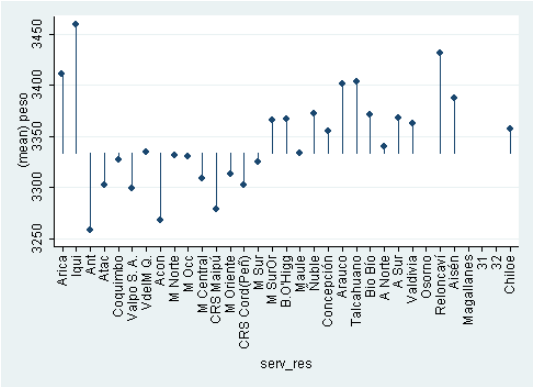


Figura 5.17: Peso de los recién nacidos por Servicios, comparados con el promedio nacional

pcarrow

Si queremos comparar dos pares de variables, es muy útil `pcarrow` . Para examinar la evolución del peso y talla promedio de los nacimientos entre servicios de salud de un año a otro, podemos colapsar los datos de ambos años, guardando los promedios de talla y peso. Pegamos ambas bases mediante un merge, usando la identificación de cada servicio. Hemos usado el comando `rename` para renombrar la variable y tener dos nombres diferentes al momento de graficar. También usamos `mlabel` para poner en la punta de la flecha el nombre de cada servicio. Obtenemos el siguiente gráfico:

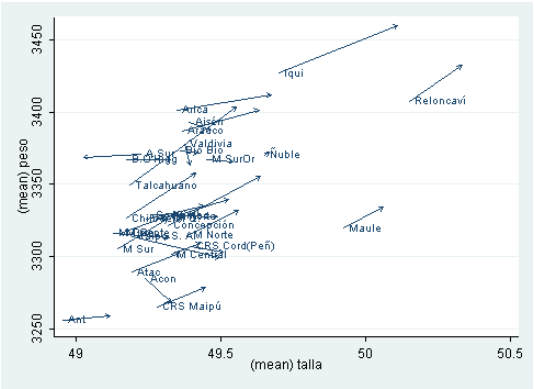


Figura 5.18: Evolución del peso y talla de los nacimientos por Servicios entre el 2008 y el 2009

```
clear
use nac08
```

```
collapse (mean)peso (mean)talla , by (serv_res)
sort serv_res
save cnac08,replace
clear
use nac09
collapse (mean)peso (mean)talla , by (serv_res)
rename peso peso9
rename talla talla9
sort serv_res
merge 1:m serv_res using cnac08
 twoway pcarrow peso talla peso9 talla9, mlabel(serv_res) saving(pcarr)
graph export pcarr.png
```

areaplot

Para destacar las diferencias entre un servicio y otro podemos hacer un gráfico de área :

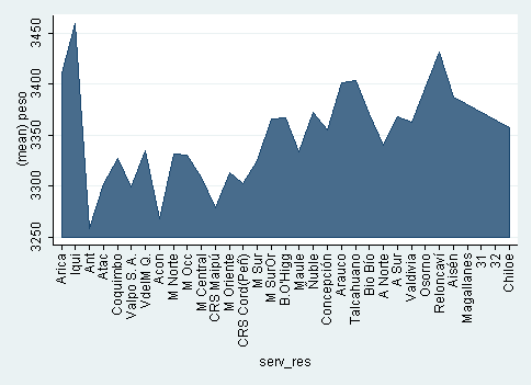


Figura 5.19: Peso de los nacimientos por Servicios de Salud, 2009

```
 twoway area peso serv_res,xlabel(1(1)33, valuelabels angle(90))saving(ar)
graph export ar.png
```

Capítulo 6

Organizando un gráfico

Hemos mostrado algunos de los gráficos que se pueden hacer con Stata. Pero hasta ahora los hemos manipulado muy poco. Ahora vamos a trabajarlos para hacerlos más presentables.

6.1. Títulos

6.1.1. title y subtitle

Poner nombre a los gráficos es muy simple. Tras la coma, escribimos entre paréntesis el título. Si queremos escribir un poco más, podemos hacer un subtítulo ,

6.1.2. xtitle e ytitle

Para titular los ejes, podemos recurrir a `xtitle, ytitle` . También es posible cambiar la orientación de estos títulos.

6.1.3. region

Para sacar el fondo azul, podemos poner un fondo blanco .Además hemos pintado la línea y el color del área rojo y puesto la línea de cero en el.promedio. Usando todas estas modificaciones redibujamos nuestros gráficos del siguiente modo:

```
twoway area peso serv_res, bcolor(red) base(3333) xlabel(1(1)33, /*
*/valuelabels angle(90)) title(Peso de los nacimientos por Servicios/*
 de Salud)subtitle( 2009) xtitle(Servicios de Salud) ytitle/*
*/ ("promedio" "de peso", orientation(horizontal)) /*
/*graphregion(fcolor(white))saving(arm)
graph export arm.png
```

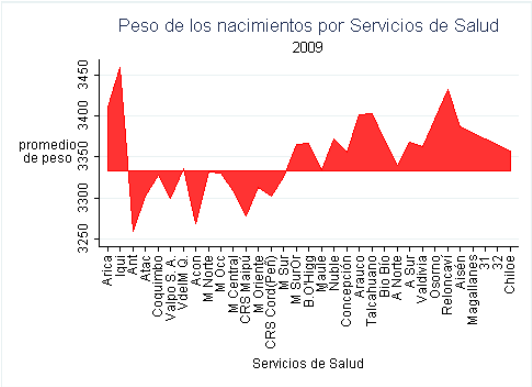


Figura 6.1: Peso de los nacimientos por Servicios de Salud, 2009

6.1.4. Líneas, leyendas y textos

Podemos transformar este gráfico en uno de líneas . No usamos el twoway porque hay sólo dos variables involucradas. Si queremos graficar varias variables mediante líneas, escribimos cada par de variables entre paréntesis.Hemos modificado el tipo de línea y su color, mediante los comandos lpattern y lcolor.

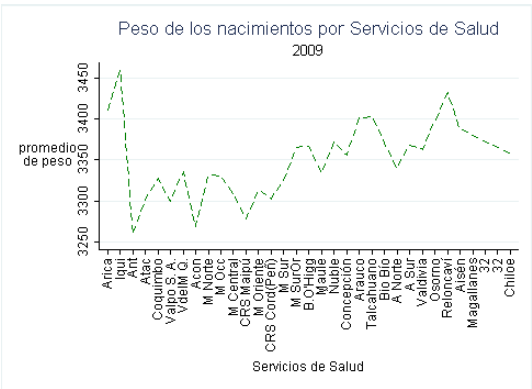


Figura 6.2: Peso de los nacimientos por Servicios de Salud, 2009

```
line peso serv_res, lpattern(dash) /*
*/ lcolor(green) xlabel(1(1)33, valuelabels angle(90))/*
*/ title(Peso de los nacimientos por Servicios de Salud)/*
*/subtitle( 2009) xtitle(Servicios de Salud) ytitle/*
*/("promedio" "de peso", orientation(horizontal))/*
*/ graphregion(fcolor(white)) saving(lg1)
graph export lg1.png
```

Segundo eje y

Para incluir las tallas en el gráfico, construímos un segundo eje y . Además ordenamos la leyenda bajo el gráfico poniendola en una sola columna y le especificamos sus textos. Modificamos los anchos de línea y sus colores.

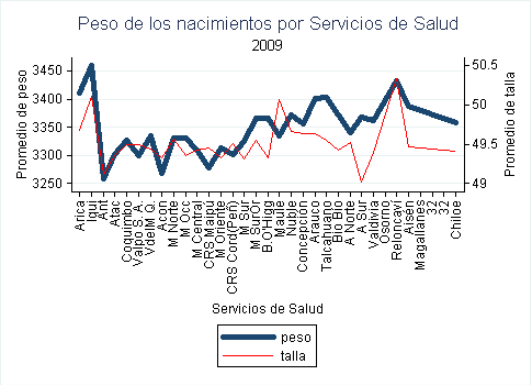


Figura 6.3: Peso y tallas de los nacimientos por Servicios de Salud, 2009

```
twoway(line peso serv_res, lwidth(vthick))/*
 /*(line talla serv_res, lcolor(red)lwidth(thin)yaxis(2)),/*
 */ xlabel(1(1)33, valuelabels angle(90))
 title(Peso de los nacimientos por Servicios de Salud)/*
*/subtitle(2009) xtitle(Servicios de Salud) ylabel(,angle(0)) /*
*/ylabel(,angle(0) axis(2))  ytitle(Promedio de peso,/*
*/ angle(90)axis(1)) ytitle(Promedio de talla, axis(2))/*
*/graphregion(fcolor(white))legend(cols(1) label(1 "peso")/*
*/label(2 "talla"))saving(lg2)
graph export lg2.png
```

6.1.5. Insertar textos

Es posible poner una leyenda arbitraria mediante `text` . Las coordenadas se ajustan de acuerdo a las magnitudes de los ejes: Primero el eje y y luego el eje x.

```
sysuse auto
local  area graphregion(fcolor(white))
scatter price weight, text(8000 4000 "ojo con estos datos",
orientation(vertical))‘area’ saving(text)
graph export text.png
```

6.1.6. Insertar líneas

Queda claro que insertar una línea debe ser muy sencillo:

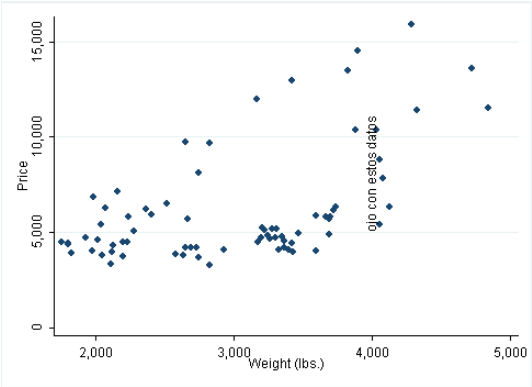


Figura 6.4: Insertando un texto en un gráfico

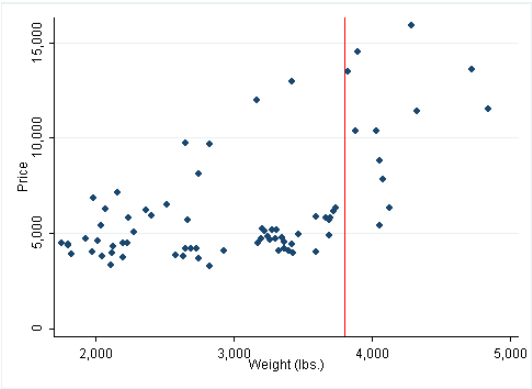


Figura 6.5: Insertando una línea en un gráfico

```
sysuse auto
local  area graphregion(fcolor(white))
scatter price weight, xline(3800, lcolor(red))‘area’ saving(line)
graph export line.png
```

Una línea usando local

Para insertar un valor a través de una local , generamos la local para guardar una media y luego la insertamos como línea.

```
sum price
local line= r(mean)
local  area graphregion(fcolor(white))
scatter price weight, yline(‘line’)‘area’ saving(linelocal)
graph export linelocal.png
```

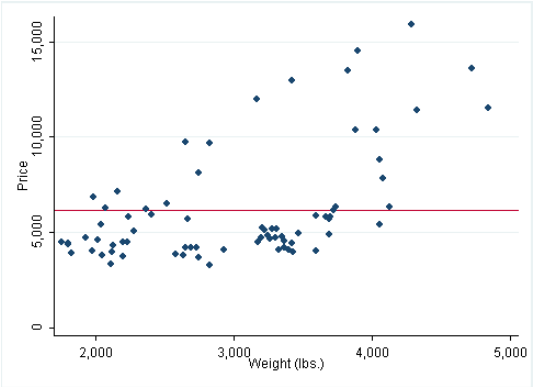


Figura 6.6: Media de precios insertada dede una local

6.1.7. Moviendo objetos en el gráfico

El comando position permite especificar una localización usando una analogía con un reloj. La especificación `ring(0)` ubica el título dentro del gráfico.

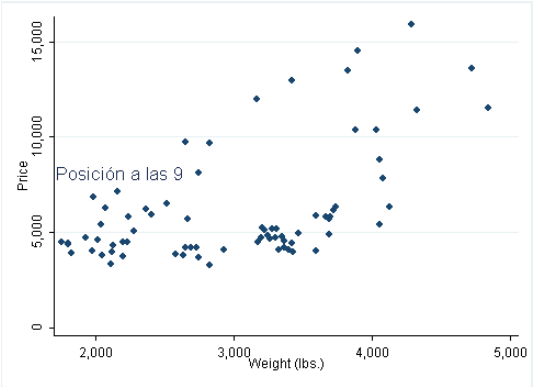


Figura 6.7: Título a las nueve

```
local  area graphregion(fcolor(white))
scatter price weight, title(Posición a las 9, position(9) ring(0))/*
*/‘area’ saving(position)
graph export position.png
```

El comando placement permite especificar la situación de acuerdo a los puntos cardinales:

```
local  area graphregion(fcolor(white))
scatter price weight, title(Situado al sur, placement (s) ring(0))‘/*
*/area’ saving(placement)
```

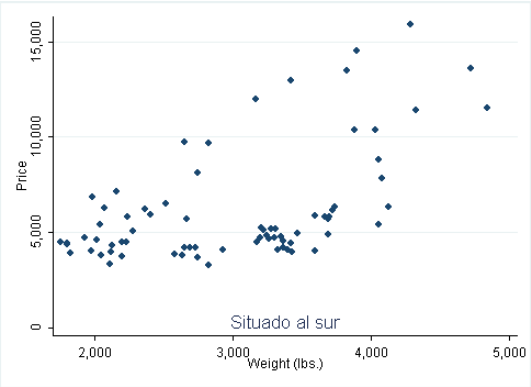


Figura 6.8: Título al sur

```
graph export placement.png
```

6.1.8. Modificando las proporciones entre ejes

Si queremos alterar la proporción de los ejes, basta especificar las magnitudes relativas que deseamos en los ejes .
El siguiente gráfico muestra la modificación de la población aborigen americana, entre 1492 y 1633, y ha sido construido mediante una especificación de los ejes en una proporción del eje x:y de 3:4. Poniendo entre comillas dos partes del título se ha ubicado en dos líneas el mismo y se ha modificado el tamaño de las letras, mediante el comando `size(medsmall)`. Los datos han sido tomados de Cook [12].

```
clear
input año porcentaje
1492     100
1514     80
1519  82
1528 53.3
1531 54.1
1534 40.6
1545 27.8
1546 29.3
1563 23.4
1576 24.9
1591 13.2
1595 13.5
1597 12.4
1611 13.3
1614 12.2
```

```
1630 13.2
1633 11.9
end
twoway line porcentaje año, title("Modificación de la población aborigen /*
*/americana" "1492-1633", size(medsmall))ylabel(0(25)100, angle(0))/*
*/ ytitle("") xtitle("")xsize(1.5) ysize(2)/*
*/graphregion(fcolor(white)) saving(indios9233)
graph export ind.png
```

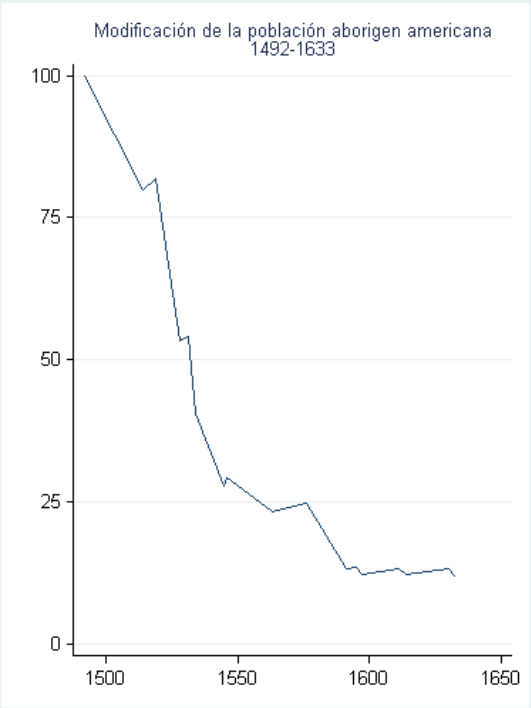


Figura 6.9: Modificación de la población aborigen americana entre 1492 y 1633

Capítulo 7

Gráficos especiales

7.1. Pirámide poblacional

Es posible construir una pirámide poblacional mediante un gráfico de barras horizontales, que tenga su eje vertical en el cero de las x. Traemos desde un archivo excel.csv los datos del censo por grupo de edad y sexo, graficamos los hombres en cifras negativas (a la izquierda del cero x) y positivas las mujeres (a la derecha del cero x), cambiando sólo el nombre de esos números. Para etiquetar las edades, sobreponemos puntos con marcador invisible alineadas con el valor 0 del eje x, para usarlo con las etiquetas.

```
clear
insheet using proyecc_censo.csv, delimiter(";")
gen hommil02= -v6/100000
gen mujmil02= v7/100000
*piramide censal 02*
gen id=_n
# delimit;
label define t
1 "Bajo 5"
 2 "5 a 9"
 3 "10 a 14"
 4 "15 a 19"
 5 "20 a 24"
 6 "25 a 29"
 7 "30 a 34"
 8 "35 a 39"
 9 "40 a 44"
 10 "45 a 49"
 11 "50 a 54"
 12 "55 a 59"
 13 "60 a 64"
```

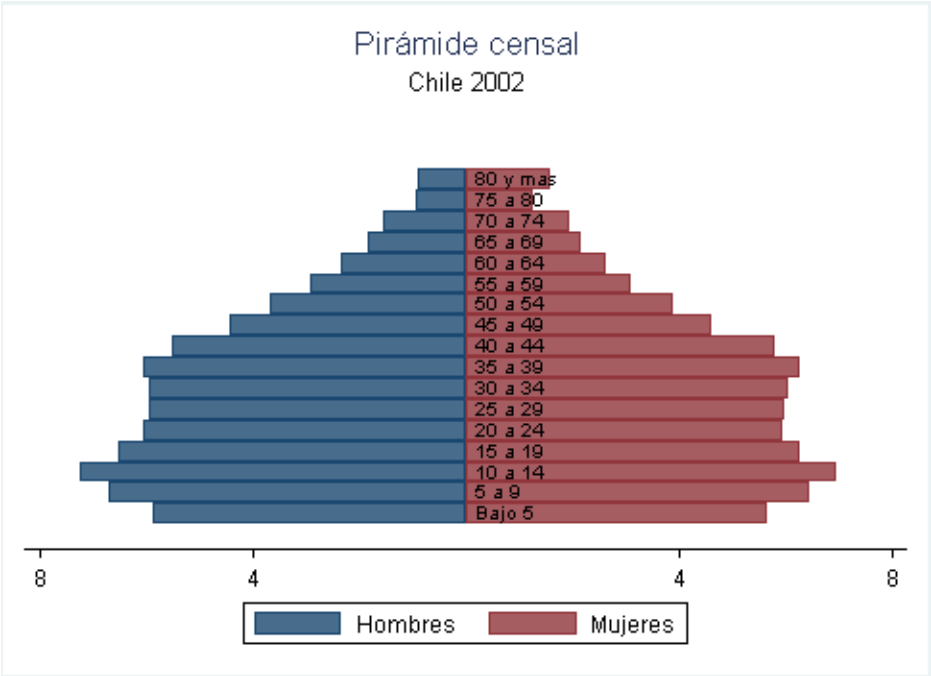


Figura 7.1: Pirámide poblacional censal. Chile 2002

```
 14 "65 a 69"
 15 "70 a 74"
 16 "75 a 80"
 17 "80 y mas";
 # delimit cr
label value id t
gen x=0
twoway (bar hommil02 id, horizontal) (bar mujmil02 id, horizontal)/*
*/(scatter id x, msymbol(i) /*
*/ mlabel(id) mlabcolor(black)), xlabel(-8 "8" -4 "4" 4 8) /*
*/title(Pirámide censal) subtitle(Chile 2002)/*
*/ yscale(off) ylabel( ,nogrid) legend(order(1 "Hombres" 2 "Mujeres"))/*
*/graphregion(fcolor(white)) saving(pir1)
graph export pir1.png
```

7.1.1. Combinando gráficos

Cuando guardamos gráficos, esto nos permite combinarlos . Al juntar dos gráficos podemos especificar si queremos el eje x o el y común y además podemos disponerlo de modo que queden en una sola columna o en varias. Las áreas se pueden mover también. Es importante diseñar el gráfico teniendo en vista que

luego se combinará. En este caso pese a indicar un fondo blanco, el celeste original no se modifica.

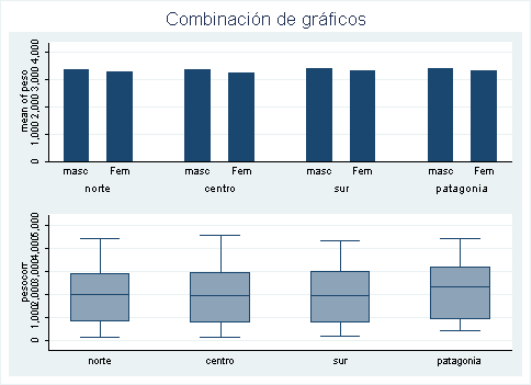


Figura 7.2: Combinación de dos gráficos en una sola columna

```
local  area graphregion(fcolor(white))
graph combine b2.gph  bp1.gph, title( Combinación de gráficos)/*
*/ saving(combine) xcommon cols(1) ‘area’
graph export combine.png
```

7.2. Gráficos post modelo

7.2.1. Ajustar una recta a una nube de puntos

Usaremos una local para escribir todo el ajuste del fondo . Dibujaremos una recta estimada por mínimos cuadrados ordinarios, a una nube de puntos.

```
clear
sysuse auto
local  area graphregion(fcolor(white))
scatter price mpg ||lfit price mpg, ‘area’ saving(lfit)
graph export lfit.png
```

7.2.2. Agregando el intervalo confidencial

Dibujaremos en sombreado el intervalo confidencial.

```
local  area graphregion(fcolor(white))
twoway(lfitci price mpg) (scatter price mpg), ‘area’saving(lfitci)
graph export lfitci.png
```

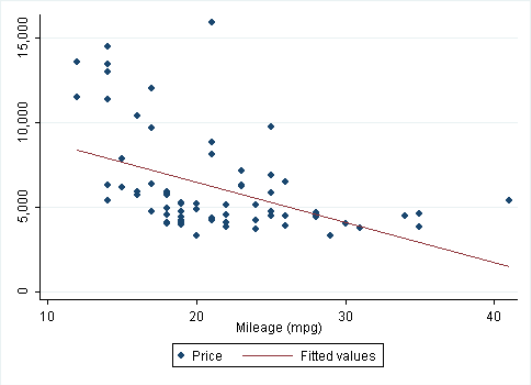


Figura 7.3: Precios y millajes y regresión lineal

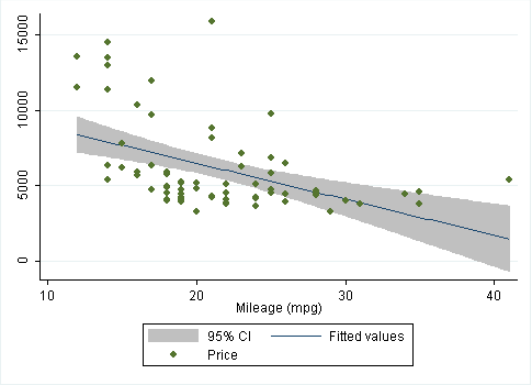


Figura 7.4: Precios y millajes en regresión lineal con C.I.

7.2.3. Examen de residuos post regresión

Stata propone varias estadísticas post regresión y gráficos para examinar la bondad de ajuste o estudiar el comportamiento de los residuos. En regresión lineal mencionaremos los gráficos de residuos contrastados con los valores ajustados (rvfplot) y los de leverage .

```
reg price mpg headroom trunk weight
local  area graphregion(fcolor(white))
rvfplot, 'area' title(Residuos versus valores predichos) saving(rvfplot)
graph export rvfplot.png
lvr2plot, 'area' title(Residuos versus valores predichos) saving(lvr2plot)
graph export lvr2plot.png
```

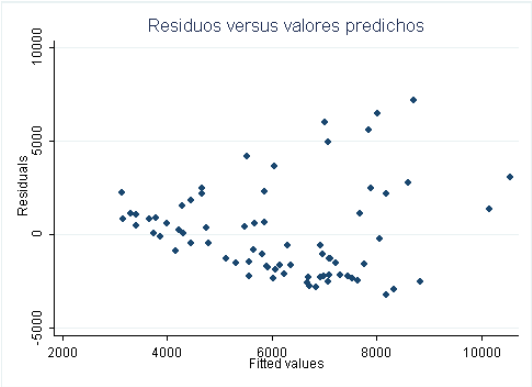


Figura 7.5: Residuos versus valores ajustados

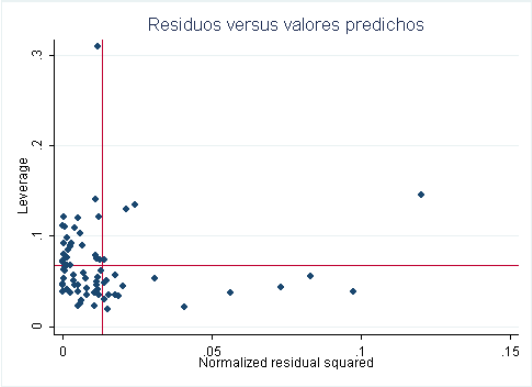


Figura 7.6: Apalancamientos versus residuos estandarizados

7.2.4. ROC

El comando tabroc permite dibujar la curva receiver operating characteristic(ROC) que permite comparar pruebas. También es posible obtenerla tras una regresión logística usando el comando `lroc` como un postestimation comand. Aprovechamos para mostrar como se puede construir una dummy en forma sencilla.
Además generamos una variable nueva definiéndola como formada sólo por valores missing, y luego la reemplazamos específicamente con los valores que queremos, para no cometer errores.

```
clear
use nac08
tab tipo_atenc, gen(dummy)
gen bp=.
replace bp=0 if peso<2500
```

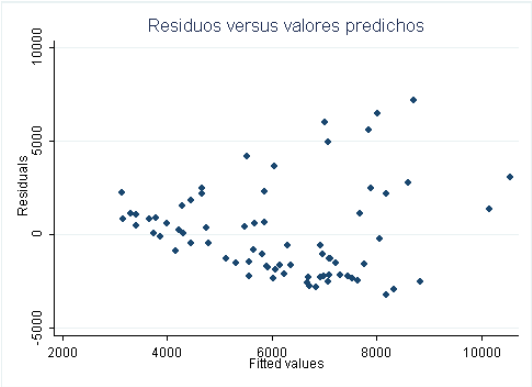


Figura 7.7: Residuos versus valores ajustados

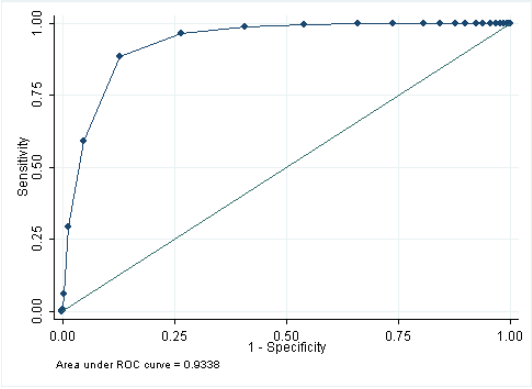


Figura 7.8: Curva roc de las semanas sobre el bajo peso

```
replace bp=1 if peso>=2500
logit bp semanas
local  area graphregion(fcolor(white))
lroc, graph 'area' saving(roc)
graph export roc.png
*o también
roctab bp semanas, graph 'area' saving(roc)
graph export roc.png
```

7.2.5. correlogramas

En modo ts (time series) Stata grafica los valores de los coeficientes de autocorrelación simple y parcial , marcando en gris la zona de confianza de esos estimadores.

```
use panel
```

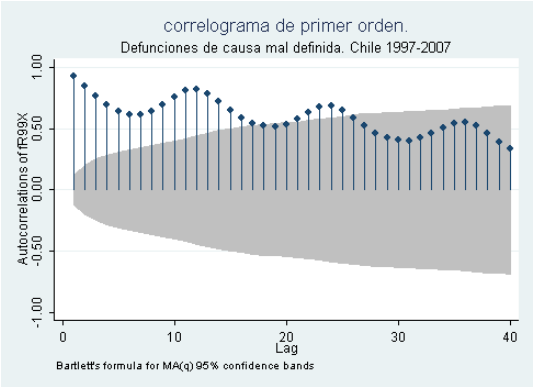


Figura 7.9: Autocorrelación simple

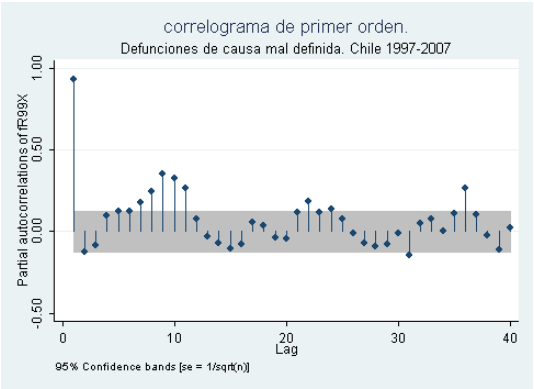


Figura 7.10: Autocorrelación parcial

```
tsset id mes_
ac fR99X if id==1, title (correlograma de primer orden. )/*
*/ subtitle(Defunciones de causa mal definida. Chile 1997-2007)saving(a)
graph export pc.png
pac fR99X if id==1, title (correlograma de primer orden. )/*
*/ subtitle(Defunciones de causa mal definida. Chile 1997-2007) saving(pac)
graph export pac.png
```

Capítulo 8

Utilidades varias

8.1. set memory

En algunas ocasiones, cuando la base de datos tiene muchos casos o variables, la memoria puede resultar insuficiente. Stata permite especificar la memoria que usaremos. El comando es `set mem` . Para que la asignación sea permanente, debemos agregar tras una coma `,perm`; para saber cuanta memoria está asignada, escriba `memory`

Bibliografía

[1] Baum C. An Introduction to Stata Programming. Texas: Stata Press; 2009.

[2] Long S. The Workflow of Data Analysis Using Stata. Texas: Stata Press; 2009.

[3] Watson I. Publication quality tables in STATA. un tutorial for the tabout program. mail@ianwatson.com.au; 2005.

[4] Mitchell M. A Visual Guide to Stata Graphics. Texas: Stata Press; 2012.

[5] Cleveland W, McGill R. Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods. Journal American Statistician Association. 1984;79:531–553.

[6] Friendly M. Re-Visions of Minard. Statistical Computing and Graphics Newsletters. 1999;11.

[7] Gelman A, Pasarica C, Dodhia R. Let's Practice What We Preach: Turning Tables into Graphs. The American Statistician. 2002;56:121–130.

[8] Lewi PJ. Speaking of Graphics[Internet]; [updated 2009 september 26]; [cited 2011 Oct 7]. Avalaible from: http://www.lewi.be/.

[9] Wainer H, Velleman P. STATISTICAL GRAPHICS: Mapping the Pathways of Science. Annu Rev Psychol. 2001;52:305–335.

[10] Boggs R. Exploring Data;. [internet][cited 20 0ctubre 2011] Avaliable from: http://exploringdata.net/six_char.htm.

[11] Tufte E. THE WORK OF EDWARD TUFTE AND GRAPHICS PRESS[internet]; [updated 2003 september]; [cited 2011 Oct 7]. Avalaible from: http://www.edwardtufte.com/tufte/.

[12] Cook N. 14: Epidemias y dinámica geográfica. In: Pease F, editor. Historia General de América. vol. II. Madrid: Ediciones Unesco/Editorial Trotta; 2000. p. 301–318.

Índice alfabético