Convertire l'output di Stata in codice html

Nicola Tommasi
C.I.D.E.
Centro Interdipartimentale di Documentazione Economica
Università degli Studi di Verona

Contatto:
nicola@cide.univr.it
tel. 045 802 80 48

1. Introduzione

Capita spesso di dover fornire i risultati di elaborazioni a persone che non sono in possesso di un background tale da capire ed apprezzare in maniera immediata l'output testuale di Stata, sia esso in formato testo puro (log file) che SMCL. Solitamente questo soggetti sono abituati a lavorare con tabelle inserite in fogli di calcolo e rimangono disorientati da output testuali. In questi casi il passaggio dall'output di Stata a spreadsheet è un fatto imprescindibile e, specialmente se la mole di tabelle è consistente, spesso rappresenta un handicap. Una possibile soluzione al problema consiste nel produrre un output direttamente in formato .html, da cui poi sia agevole il passaggio dei risultati all'interno di un foglio di calcolo con le funzioni copia e incolla o tramite la funzione importa dati esterni.

2. I "limiti" dei log di Stata

Oltre alla idiosincrasia descritta nell'introduzione per tutto ciò che non è inserito in un foglio di calcolo, l'output di Stata presenta alcuni limiti oggettivi. Alcuni di questi sono legati direttamente ai comandi. Il comando `tabulate` per esempio tronca le label dei valori se queste eccedono in lunghezza.

```
. tab  cod_esame tipo_esame

    Esami sostenibili |
    presso le diverse |
              facoltà |
   dell'università di |       Tipologia dell'esame
    Verona per codici |   Scritto      Orale  Orale + S |     Total
----------------------+---------------------------------+----------
0115C LINGUA SPAGNOLA |         0          0        169 |       169
0117C LINGUA TEDESCA  |         0        112          0 |       112
0166C STORIA DEL PENS |         0         14          0 |        14
0169C STORIA DELL'IND |         0         51          0 |        51
0135C ECONOMIA DEI TR |         0          5          0 |         5
0137C ECONOMIA DELL'A |         0          4          0 |         4
0138C ECONOMIA DELLO  |         0          2          0 |         2
0140C ECONOMIA INDUST |         0          1          0 |         1
0143C EVOLUZIONE DELL |         0         43          0 |        43
0144C GEOGRAFIA DEI S |         3          0          0 |         3
0754N ISTITUZIONI DI  |         0        449          0 |       449
....
```

Inoltre, sempre il comando `tabulate`, salta le specificazioni della variabile che hanno frequenza zero e questo talvolta può rappresentare un problema[1]. Sempre in questo ambito, problematiche simili sono riscontrabili per il comando `summarize`[2], `table`, `tabstat` e altri.

Si aggiunga infine l'impossibilità di inserire nei log files i grafici creati in Stata.

La conversione dell'output direttamente in formato .html permetterebbe di risolvere molti di questi problemi a patto di riuscire a produrre facilmente il codice. Infatti, sarebbe possibile produrre sia tabelle di risultati formattabili a piacere, sia inserire i grafici come immagini.

3. L'applicazione pratica

L'esigenza di superare le limitazioni insite nei .log di Stata è nata in seguito alla necessità di produrre un report fruibile dal personale medico, infermieristico e dalla direzione sanitaria che sintetizzasse i risultati di una indagine sulla soddisfazione del paziente condotta dall'Ospedale Sacro Cuore Don Calabria di Negrar (VR). Tale indagine, condotta con frequenza annuale, si sostanzia in un questionario di 25 domande per la maggior parte a risposta chiusa. Vengono prodotti 19 report, specifici per ciascun reparto coinvolto, oltre a un ventesimo con alcune peculiarità per la direzione sanitaria. In totale vengono prodotte per ciascun anno di indagine, circa 1100 tabelle e 1200 grafici.

La proposta iniziale di fornire i risultati dell'elaborazione solo come tabelle in formato testo e senza

1 Per ovviare a questo inconveniente si veda, per esempio il comando `tabcount` scritto da Nicholas J. Cox.

2 Per `summarize` si veda il comando `fsum` di Fred Wolfe.

grafici ha suscitato diverse perplessità da parte della direzione dell'ospedale proprio in relazione alla fruibilità dei risultati. Da qui la necessità di trovare una via alternativa che coniugasse semplicità nella produzione dei risultati e nella comprensione e facilità di utilizzo da parte dell'utente finale.

In pratica si tratta di implementare un sistema che rappresenti i singoli risultati di Stata racchiusi in tag html relativamente ai comandi:

`-tabulate` sia `oneway` che `twoway` e con la presenza anche delle specificazioni a frequenza zero

`-summarize`

`-graph` nelle sue varie specificazioni

3.1. Conversione dell'output di tabulate

In generale l'idea è quella di catturare il vettore/matrice delle frequenze, salvarlo in una matrice e poi accedere ai singoli elementi di essa in fase di costruzione del codice html. Per l'intestazione della tabella, delle righe e delle colonne si fa ricorso alle Macro Extended Functions (cfr. manuale Programming alla voce macro). Di seguito viene presentato il codice commentato relativamente ai casi di `tabulate twoway` semplificato, ovvero senza considerare le specificazioni a frequenza zero e il caso di `tabulate twoway` con espressa considerazione anche delle frequenze zero. Il caso di `tabulate oneway` è "facilmente" deducibile adattando il caso twoway, altrimenti è possibile contattare l'autore.

tabulate twoway semplificato

```
capture program drop tab_tw_semp;
program define tab_tw_semp;
syntax varlist [if] [in] [, miss(string)];
preserve;
local row_var `1';
local col_var `2';
qui tab `row_var' `col_var', `miss' matcell(mtx_values) matrow(rowname)
     matcol(colname);
```

`tabulate` delle due variabili, con eventuale specificazione dell'opzione per considerare i missing values (`miss')[3]. Qui si sfrutta la possibilità offerta da `tabulate` di salvare in matrici distinte la matrice delle frequenze (matcell), il vettore con le label per riga (matrow) e il vettore con le label per colonna (matcol)

```
*** add col of total x row;
local dim_c = colsof(mtx_values);
local dim_r = rowsof(mtx_values);

mat ones = J(`dim_c',1,1);
mat TOT_r = mtx_values*ones;
mat mtx_values = (nullmat(mtx_values),TOT_r);
matrix drop TOT_r ones;

*** add row of total x col;
local dim_c = colsof(mtx_values);
local dim_r = rowsof(mtx_values);
mat ones = J(`dim_r',1,1);
mat TOT_r = mtx_values'*ones;
mat mtx_values = (nullmat(mtx_values)\TOT_r');
matrix drop TOT_r ones;

local dim_c = colsof(mtx_values);
local dim_r = rowsof(mtx_values);
```

Purtroppo l'opzione matcell() non salva né i totali per riga né quelli per colonna. Per ottenerli si crea un opportuno vettore con valori 1 (mat ones...) conformato con la matrice mtx_values alla quale viene moltiplicato per ottenere il vettore dei totali. L'operazione viene eseguita prima per riga e poi per colonna. I vettori così ottenuti vengono aggiunti alla matrice mtx_values per mezzo della funzione `nullmat`. Infine, nelle local dim_c e dim_r viene salvata la dimensione per riga e per colonna della matrice.

```
/*****************************
mata;
TMP = st_matrix(mtx_values);
TMP = TMP , rowsum(TMP);
```

3 Si consiglia di modificare il formato standard dei missing values nel formato .a ed aggiungere questo nel label define.

```
TMP = TMP \ colsum(TMP);
st_matrix("matx_values",TMP);
end;
local dim_c = colsof(mtx_values);
local dim_r = rowsof(mtx_values);
*******************************/;
```

Se si possiede la versione 9 di Stata si può semplificare la creazione dei totali per riga e per colonna usando le funzionalità offerte da Mata.

```
/**** TABLE DEFINE ***/;

**** INTESTAZIONE;
di "<table border='0' cellspacing='0' bordercolordark='white'
     bordercolorlight='black'>";
di      "<tr bgcolor='$sfo1'>";
di         "<td width><p> </p></td>";
**** END INTESTAZIONE;
```

Qui inizia il codice html relativo alla tabella. Si apre il tag per la tabella <table>, si definisce il colore di sfondo della prima riga ricorrendo ad una global e si definisce la prima cella (vuota) della tabella[4]. La global sfo1 è definita come colore esadecimale (`global sfo1 #7B68EE`).

```
**** DEF.  COLS;
local top_col_var = colsof(colname);
foreach col of numlist 1/`top_col_var' {;
local pos = colname[1,`col'];
local li : label (`col_var') `pos';
di         "<td width='70'><p class=t88> `li' </p></td>";
};
di         "<td width='70'><p class=t88> TOTALE </p></td>";
di     "</tr>";
**** END DEF. COLS;
```

In questa sezione si definisce l'intestazione delle singole colonne attraverso un ciclo `foreach`. L'intestazione viene estratta tramite la funzione `label` dalla label associata alla variabile in colonna e poi visualizzata con il comando `display`. Alla fine del ciclo si crea la colonna per i totali.

```
**** DEF. ROWS;
local top_row_var = rowsof(rowname);
foreach row of numlist 1/`top_row_var' {;
   local pos = rowname[`row',1];
   local li : label (`row_var') `pos';

     if  int(`row'/2)==`row'/2 {;
          local tr "tr";
                              };
     else {;
          local tr "tr bgcolor='$line'";
          };

   di "<`tr'>";
   di "<td width><p>`li'</p></td>"  /*** intestazione riga ***/;
```

Per la costruzione di una tabella in html per prima cosa si deve definire la riga e all'interno di questa le colonne. Per implementare questa procedura si ricorre ad un ciclo `foreach` per costruire la riga e si annida al suo interno un altro ciclo `foreach` per costruire le colonne. Qui si definisce il primo ciclo. Attraverso una struttura `if ... else` si definisce il colore di sfondo delle righe, trasparente per le righe pari, colorato per le dispari. L'intestazione delle righe è analoga a quanto visto per le colonne.

```
foreach col of numlist 1/`top_col_var' {;
di "<td width><p class=t8>"  %03.2f
     mtx_values[`row',`col']/mtx_values[`dim_r',`col']*100 "</p></td>";
};

di "<td width><p class=t8>" %03.2f
     mtx_values[`row',`dim_c']/mtx_values[`dim_r',`dim_c']*100 "</p></td>" /**
     Aggiunge cella del totale per riga  **/;
di "</tr>";
};
```

4 Solitamente i vari attributi dei tag in html (border='0' per esempio) vengono definiti tra virgolette. Poiché il comando display non consente che ci siano virgolette annidate, si deve ricorrere al simbolo dell'apostrofo.

Ora si passa a popolare le celle della riga definita precedentemente. Attraverso gli indici di riga e di colonna `row' e `col' si individua l'elemento corretto della matrice mtx_values e lo si trasforma in percentuale rispetto al totale di colonna.

```
/*** aggiunta riga dei totali ***/;
   di "<tr bgcolor='$sfo1'>";
   di "<td width><p> TOTALE </p></td>"  /*** intestazione riga ***/;
   foreach col of numlist 1/`top_col_var' {;
         di "<td width><p class=t8>" %3.0f mtx_values[`dim_r',`col'] "</p></td>";
   };
         di "<td width><p class=t8>" %3.0f mtx_values[`dim_r',`dim_c'] "</p></td>"
     /** Aggiunge cella del totale  **/;
    di "</tr>";

di "</table>";
```

Infine si aggiunge la riga dei totali per colonna come frequenza.

```
end;
```

tabulate twoway con specificazioni a frequenza zero

Questa particolare versione del programma nasce dalla necessità di visualizzare in questo contesto anche le specificazioni di una variabile categorica con frequenze pari a zero. È noto che in questo caso il comando `tabulate` omette la voce, come si può notare dall'esempio riportato:

```
. tab d14e_rc

     Orario |
      pasti |      Freq.     Percent        Cum.
------------+-----------------------------------
          2 |         13        2.51        2.51
          3 |        209       40.43       42.94
          4 |        130       25.15       68.09
          5 |        165       31.91      100.00
------------+-----------------------------------
      Total |        517      100.00
```

Per visualizzare anche la specificazione mancante ci viene in soccorso il comando `tabcount` scritto da Nicholas J. Cox, che nell'ultima versione implementa anche la possibilità di salvare i risultati in una matrice:

```
. tabcount d14e_rc, v(1/5) zero

----------------------
Orario    |
pasti     |      Freq.
----------+-----------
        1 |          0
        2 |         13
        3 |        209
        4 |        130
        5 |        165
----------------------
```

Ora verrà presentato il codice per questo particolare caso. Si avverte che verrà commentato solo ciò che differisce in maniera sostanziale rispetto a quanto visto per il `tabulate twoway` semplificato, al quale si può far riferimento per tutto il resto.

```
capture program drop tab_tw_zero;
program define tab_tw_zero;
preserve;
local row_var  `1';
local col_var  `2';
local l_row_var : value label `row_var';
local l_col_var : value label `col_var';

   qui label l `l_row_var';
   qui local bot_row_var = r(min);
```

```
   qui local top_row_var = r(max);
   qui label l `l_col_var';
   qui local bot_col_var = r(min);
   qui local top_col_var = r(max);

qui count if `row_var' >=.a;
if  r(N) > 0 {;
          local new = `top_row_var'+1;
          qui recode `row_var' (.a=`new');
          label define `l_row_var' `new' "Non Risponde", add;
          qui label l `l_row_var';
          qui local bot_row_var = r(min);
          qui local top_row_var = r(max);
             };

qui count if `col_var' >=.a;
if  r(N) > 0 {;
         local new = `top_col_var'+1;
         qui recode `col_var' (.a=`new');
         label define `l_col_var' `new' "Non Risponde", add;
         qui label l `l_col_var';
         qui local bot_col_var = r(min);
         qui local top_col_var = r(max);
             };
```

Per le variabili che presentano valori missing, si ricodifica quest'ultimo e lo si aggiunge dopo l'ultima specificazione della variabile stessa. Ovvero se una variabile arriva fino a 4 specificazioni, il missing viene aggiunto come quinta. Questo serve per avere coerenza tra la numerazione delle label e la numerazione degli indici per riga e per colonna della matrice delle frequenze.

```
local col_matrix=0;
foreach col of numlist `bot_col_var'/`top_col_var' {;
  local col_matrix=`col_matrix'+1;
};
local row_matrix=0;
foreach row of numlist `bot_row_var'/`top_row_var' {;
  local row_matrix=`row_matrix'+1;
};
```

Qui si definisce la dimensione della matrice dei risultati. Servirà in seguito per costruire il vettore nullo dei casi a frequenza nulla.

```
foreach col of numlist `bot_col_var'/`top_col_var' {;
qui capture tabcount `row_var' if `col_var'==`col', v(`bot_row_var'/`top_row_var')
     matrix(vc_`col') zero;
qui count if `col_var'==`col';
if r(N)>0 {;
          mat vc_`col' = (nullmat(vc_`col')\r(N));
          };
else  {;
         mat zero_`col' = J(`row_matrix',1,0);
         mat vc_`col' = (nullmat(zero_`col')\r(N));
         };

mat `row_var' = (nullmat(`row_var'),vc_`col');

};
```

Come già accennato qui si ricorre al comando `tabcount` che viene invocato per la variabile in riga condizionata per ciascuna specificazione della variabile in colonna. Nell'opzione v($\#_{min}$/$\#_{max}$) si specifica il range dei valori ammissibili per la variabile in riga e il vettore delle frequenze viene salvato tramite l'opzione matrix(). Ciascun vettore così ottenuto viene assemblato nella matrice totale delle frequenze. Nel caso di frequenza nulla della variabile in colonna, il ciclo `if ... else` provvede a costruire un vettore con valori pari a zero da assemblare alla matrice delle frequenze (mat zero_`col' =...).

```
*** add col of total x row;
local dim_c = colsof(`row_var');
local dim_r = rowsof(`row_var');

mat ones = J(`dim_c',1,1);
mat TOT_r = `row_var'*ones;
mat `row_var' = (nullmat(`row_var'),TOT_r);
local dim_c = colsof(`row_var');
```

```
local dim_r = rowsof(`row_var');
matrix drop TOT_r ones;

/**** TABLE DEFINE ***/;

**** INTESTAZIONE;
di "<table border='0' cellspacing='0' bordercolordark='white'
     bordercolorlight='black'>";
di     "<tr bgcolor='$sfo1'>";
di         "<td width><p> </p></td>";
**** END INTESTAZIONE;

**** DEF. NUMB. OF COLS;
foreach col of numlist `bot_col_var'/`top_col_var' {;
local li : label (`col_var') `col';
di         "<td width='70'><p class=t88> `li' </p></td>";
};
di         "<td width='70'><p class=t88> TOTALE </p></td>";

di     "</tr>";
**** END DEF. NUMB. OF COLS;

**** DEF. ROWS;
foreach row of numlist `bot_row_var'/`top_row_var' {;
   local li : label (`row_var') `row';

     if  int(`row'/2)==`row'/2 {;
          local tr "tr";
                              };
     else {;
          local tr "tr bgcolor='$line'";
          };

   di "<`tr'>";
   di "<td width><p>`li'</p></td>"  /*** intestazione riga ***/;

   foreach col of numlist `bot_col_var'/`top_col_var' {;
         di "<td width><p class=t8>"  %03.2f
     `row_var'[`row_t',`col']/`row_var'[`dim_r',`col']*100 "</p></td>";
   };
         di "<td width><p class=t8>" %03.2f
     `row_var'[`row_t',`dim_c']/`row_var'[`dim_r',`dim_c']*100 "</p></td>" /**
     Aggiunge colonna del totale  **/;
di "</tr>";
};

/*** aggiunta riga dei totali ***/;
   di "<tr bgcolor='$sfo1'>";
   di "<td width><p> TOTALE </p></td>"  /*** intestazione riga ***/;
   foreach col of numlist 1/`top_col_var' {;
         di "<td width><p class=t8>" %3.0f mtx_values[`dim_r',`col'] "</p></td>";
   };
         di "<td width><p class=t8>" %3.0f mtx_values[`dim_r',`dim_c'] "</p></td>"
     /** Aggiunge cella del totale  **/;
    di "</tr>";

di "</table>";
restore;

end;
```

3.2. Conversione dell'output di summarize

Viene qui presentato il codice per produrre tabelle contenenti le medie condizionate di variabili. Per produrre la matrice dei valori da visualizzare si ricorre al comando `makematrix`, anche questo scritto da Nicholas J. Cox.

```
capture program drop table_sum;
program define table_sum;
syntax varlist [if] [in] [, cond(string)];
```

```
qui label l `cond';
qui local bot_col_var = r(min);
qui local top_col_var = r(max);
local n_var : word count `varlist';

matrix drop mtx_mean;
foreach col of numlist `bot_col_var'/`top_col_var' {;

    qui count if `cond'==`col';
    if r(N)>0 {;
    qui capture makematrix vec, from( r(mean) ) listwise vector: su `varlist' if
      `cond'==`col';
    qui mat mtx_mean = (nullmat(mtx_mean) , vec);
              };
     else  {;
         mat vec = J(`n_var',1,.);
         qui mat mtx_mean = (nullmat(mtx_mean) , vec);
           };
    };
```

Con il comando `makematrix` si crea il vettore (vec) con le medie condizionate delle variabili di interesse. L'opzione listwise consente di considerare ciascuna variabile separatamente. In assenza di tale opzione il comando produrrebbe i risultati solo sulle osservazioni che siano contemporaneamente diverse da missing per tutte le variabili considerate.

```
local dim_c = colsof(mtx_mean);
local dim_r = rowsof(mtx_mean);

/** Vettore medie incondizionate  **/;
    qui capture makematrix vec, from( r(mean) ) listwise vector: su `varlist';
    qui mat mtx_mean = (nullmat(mtx_mean) , vec);
local row_t = `dim_c' + 1;


/**** TABLE DEFINE ***/;

**** INTESTAZIONE;
di "<table border='0' cellspacing='0' bordercolordark='white'
     bordercolorlight='black'>";
di      "<tr bgcolor='$sfo1'>";
di          "<td width><p> </p></td>";
**** END INTESTAZIONE;

**** DEF. NUMB. OF COLS;
foreach col of numlist `bot_col_var'/`top_col_var' {;
local li : label (`cond') `col';
di          "<td width='70'><p class=t88> `li' </p></td>";
};
di          "<td width='70'><p class=t88> TOTALE </p></td>";

di     "</tr>";
**** END DEF. NUMB. OF COLS;

**** DEF. ROWS;
foreach row of numlist 1/`n_var' {;
   local row_var : word `row' of `varlist';
   local li : variable label `row_var';

     if  int(`row'/2)==`row'/2 {;
          local tr "tr";
                              };
     else {;
          local tr "tr bgcolor='$line'";
          };

   di "<`tr'>";
   di "<td width><p>`li'</p></td>"  /*** intestazione riga ***/;

   foreach col of numlist `bot_col_var'/`top_col_var' {;
         di "<td width><p class=t8>"  %03.2f mtx_mean[`row',`col'] "</p></td>";
   };
         di "<td width><p class=t8>" %03.2f mtx_mean[`row',`row_t'] "</p></td>" /**
     Aggiunge cella del totale  **/;
di "</tr>";
```

```
};
di "</table>";
end;
```

3.3. Inserimento dei grafici

L'inserimento dei grafici non presenta particolari problematiche. Una volta creato il grafico esso viene esportato in formato .png attraverso il comando

```
graph export <path>/<name>.png, replace
```

Nell'ultimo aggiornamento di Stata, che porta la versione al numero 9.1, vengono introdotte anche le opzioni width() e height() per il comando `graph export` che consentono di specificare la grandezza del grafico in pixels.

4. Conclusioni

I programmi che sono stati presentati non hanno una qualità sufficiente per essere considerati dei veri e propri comandi. Essi sono nati per rispondere ad esigenze contingenti e con poca fatica possono essere modificati per produrre tabelle chiare e con un aspetto gradevole con colori di sfondo, testo in grassetto o in italico e bordi. Resta ancora da risolvere il problema di una variabile categorica con valori minori o uguali a zero. Inoltre essi non sono ancora sufficientemente generali da poter rispondere alla più ampia varietà possibile di situazioni. Naturalmente le osservazioni e tutti i consigli per migliorare la qualità del codice sono molto graditi.

Le pagine html prodotte sono di tipo statico. È allo studio un sistema che permetta all'utente di scegliere tra alcune opzioni predeterminate con le quali, tramite il linguaggio php, costruire “al volo” un file .do da lanciare in batch, per poi visualizzarne il risultato. Questo tentativo è ancora in fase sperimentale ma una versione funzionante è testabile all'indirizzo http://pilar.univr.it/simlab.

5. Bibliografia

http://www.w3.org/

http://www.w3.org/Style/CSS/

Mitchell, M. N. (2004): *A Visual Guide to Stata Graphics*. Stata Press

StataCorp. (2005): *Stata Statistical Software: Release 9*. College Station, TX: Stata Press.