The Stata Journal

The *Stata Journal*, electronic version (ISSN 1536-8734) is a publication of Stata Press, and Stata is a registered trademark of StataCorp LP.

The Stata Journal (2006)
6, Number 1, pp. 22–39

Automatic generation of documents

Rosa Gini
Regional Agency for Public Health of Tuscany
Florence, Italy
rosa.gini@arsanita.toscana.it

Jacopo Pasquini
Regional Agency for Public Health of Tuscany
Florence, Italy
jacopo.pasquini@arsanita.toscana.it

Abstract. This paper describes a natural interaction between Stata and markup languages. Stata's programming and analysis features, together with the flexibility in output formatting of the markup languages, allow generation and/or update of whole documents (e.g., reports, presentations on screen or web). We give examples for both LaTeX and HTML.

Keywords: pr0020, doutput, format, report, LaTeX, HTML, markup language

1 Introduction

Stata's commands are dedicated mainly to on-screen data analysis, the output of which is stored in a log file available to researchers for later reading.

However, users may need to produce output in different formats and collaborate with professionals who are unfamiliar with Stata log files.

The most common means for presenting analysis results are text on paper, presentations on screen, and web sites. Although visual programs like Microsoft Office and OpenOffice are a common way to display results, Stata cannot produce documents this way. Stata lacks eyes to format a table and hands to hold a mouse for cutting and pasting graphs. Nevertheless, markup languages can create all these presentation formats.

> According to *Wikipedia*, a markup language is a kind of text encoding that represents text as well as details about the structure and appearance of the text.[1]

To publish on the web, HTML is one of the best and most compatible formats. On the other hand, LaTeX is a complete language for editing and text formatting either on paper or on screen (most commonly with PDF files). Both languages are easy to learn, free, and well documented.

1. *Wikipedia*, s.v. "Markup language", http://en.wikipedia.org/wiki/Markup_language/ (accessed October 2004).

Stata can readily write text, such as the instructions for a markup language to create a report, a sequence of slides, or web pages containing tables and graphs.

Several authors have addressed the problem of formatting the output of a command in LaTeX and/or HTML. The most comprehensive reference on this issue is the work of Newson (2003), which also provides tools for printing in markup language the list of a Stata dataset so that variable labels, value labels, significant figures, etc., are formatted however you want.

More generally, we can use Stata's ability of writing text files to produce markup language code appropriate for a variety of data analyses. Stata text files can encode not only tables and graphs but also other output such as lists and trees.

By generating code and assembling the ingredients, we make Stata produce a whole document that can be browsed, printed, or displayed on screen, as appropriate. Any document so produced can be immediately updated when the figures in the database change.

This key feature is particularly suitable when the user needs to produce a great deal of output or routinely performs analyses on the same dataset structure, such as administrative databases or collection of data from a long-lasting study.

As an example of those facilities, we describe a do-file automatically constructing a web site for the Regional Agency for Public Health of Tuscany.

To apply this method, Stata commands must store their results in memory—at least as many as necessary to reproduce the screen output. This storing process is standard, with some notable exceptions (e.g., `dstdize`, `svyprop`).

2 Using file open, file write, and file close: simply writing text

The simplest thing one can imagine doing with the results of a statistical analysis is writing them to a text file, just the way one would do when manually taking notes. Instead of copying and pasting from the Results window, one can make Stata take notes in a text file itself by using the commands `file open`, `file write`, and `file close`.

Every Stata command produces output on the screen from the instructions performed by that command. At the same time, with no visible sign for the user, the same command temporarily stores some information in data structures in the computer's memory. These data structures are called *saved results*—named, for example, `r(`*name*`)`. You can obtain the list of macros, scalars, and matrices saved by a command by typing `return list` after the command. For example, when the user `summarize`s a numerical variable, the mean, minimal, and maximal values are stored in the scalar results `r(mean)`, `r(min)`, and `r(max)`, respectively.

▷ Example

Suppose that a teacher inputs in a Stata dataset, `height.dta`—the height of the children in her class in the variable `height`—and that she wants to write the mean of those values to a text file called `height.txt`. A do-file performing this task will look as follows:

```
use height

capture mkdir files
quietly summarize height

local mean=string(‘r(mean)’,"%4.1f")
local min=string(‘r(min)’,"%4.1f")
local max=string(‘r(max)’,"%4.1f")

file open height using files/height.txt, write replace
file write height ‘"The mean height of the children in my class is "’
file write height ‘"‘mean’ cm, the minimum "’ _n
file write height ‘"value is ‘min’ and the maximum value is ‘max’."’ _n
file close height
```

The do-file will provide a text file called `height.txt`, whose content will read

```
The mean height of the children in my class is 116.1 cm, the minimum
value is 103.2 and the maximum value is 128.5.
```

where Stata has inserted the number 116.1, which is the computed mean, rounded to the first decimal figure, as well as the maximum and minimum values.

Each school year, the same do-file will produce a text file with a new value for the mean, the maximum, and the minimum.

◁

3 Markup languages for output formatting

3.1 Writing markup language code: tables

The most direct way to get a LaTeX or HTML table is via Roger Newson's `listtex` command. In the philosophy of Newson (2003), we can manipulate the data to construct a DTA archive in which the content reproduces the table we want to print. We then print the table to a file in HTML, LaTeX, or another format by means of the `listtex` command.

Directly writing the table code can be useful, for instance when generating the Stata archive is more time-consuming than using the macro language or if one fancies some unusual type of tables.

The syntax for writing tables both in LaTeX and HTML is rather easy, and its basics can be summarized as follows.

In LaTeX,

- a table is declared by a `\begin{tabular}{...}` command;[2]
- each table entry is separated by an ampersand (`&`);
- each table line is separated by a double backslash (`\\`); and
- the table is closed by an `\end{tabular}` command.

In HTML,

- a table is declared by a `<table>` tag;
- each table line is contained between `<tr>` and `</tr>` tags;
- each table entry is contained between `<td>` and `</td>` tags; and
- the table is closed by a `</table>` tag.

For instance, to write a simple table of frequencies for a categorical variable, the best way is to combine `xcontract` and `listtex`, both created by Roger Newson and available from SSC. Starting from the original dataset and `xcontract`ing the variable of interest, we obtain a small dataset, with one observation per value of the categorical variable, and variables containing its frequency, both absolute and relative. The `listtex` command can then easily print this archive in either LaTeX or HTML. If the `format()` option had been specified when `xcontract`ing, the printed table will have a "human-readable" number of decimal figures. If for instance we want evidence of the modal characteristic, we need to further manipulate this archive, or we can instead write the code directly. The two alternatives are shown in the following code, which generates table 1.

```
use brfeed

preserve
local title:variable label brfeed
xcontract brfeed, nores
gen perc=string(_perc,"%3.1f")+"\%"
gen freq=string(_freq)
decode brfeed, gen(brfeed_label)

/* mark the modal observations */
qui sum _freq
qui gen byte modal=(_freq==r(max))
qui replace brfeed_label="\rowcolor[gray]{.9} "+brfeed_label if modal==1

/* FIRST ALTERNATIVE: use listtex*/
listtex brfeed_label freq perc using tex/tab_brfeed_listtex.tex,            ///
     rstyle(tabular) headlines("\begin{tabular}{p{5cm}rr}"                  ///
     "\hline \multicolumn{3}{c}{\bf `title'} \\\\ "                         ///
     " & Frequency& Percentage \\\\ \cline{2-3} ")                          ///
     footlines("\hline \end{ tabular}") replace
```

2. The parentheses should simply describe the table, by specifying how many columns it contains and whether the contents should be right-aligned, etc. See Lamport (1994) for more details.

```
/* SECOND ALTERNATIVE: write the code directly*/
file open brfeed using tex/tab_brfeed_code.tex, write replace
file write brfeed `"\begin{tabular}{p{5cm}rr}"' _n
file write brfeed `"\hline \multicolumn{3}{c}{\bf `title'} \\\\ "' _n
file write brfeed`" & Frequency& Percentage \\\\ \cline{2-3} "' _n
local lines=_N
forvalues s=1/`lines'{
        foreach desc in brfeed_label freq perc{
                local `desc'=`desc'[`s']
                }
        * this macro is set to the string that colors the row for the modal
        * observation(s)
        file write brfeed `" `brfeed' &`freq'&`perc' \\\\ "' _n
        }
file write brfeed `"\hline \end{tabular}"' _n
file close brfeed
restore
```

Table 1: Frequency table, generated by `listtex` or by a scratch code, automatically detecting the modal characteristic

Breastfeeding		
	Frequency	Percentage
Bottle feeding	180	32.7%
Partial breastfeeding	88	16.0%
Almost exclusive breastfeeding	11	2.0%
Exclusive breastfeeding	268	48.7%
NA	3	0.5%

Since the possibilities of markup languages are ample, we can make Stata produce—and automatically update—any kind of table we might need.

3.2 Writing markup language code: graphs

Stata produces beautiful graphs, which can be exported in formats ready for inclusion in documents. Again we can enhance this possibility by writing code that further automates the production of graphs and their inclusion in formatted documents.

LaTeX accepts images in the following formats:

- PDF, if `pdflatex` is used to generate a PDF document; the syntax for including the image file is

  `\includegraphics{`*namefile*`.pdf}`

- EPS or PS, if `latex` and `dvips` are used to generate a PS document; the syntax for including the image file is

  ```
  \includegraphics{namefile.eps}
  ```

HTML accepts images in several formats, the most common being PNG. The syntax is

```
<img src="namefile.png">
```

If the `graph export` function of Stata is inadequate (for example, if one needs to export a high-resolution graph in PNG format), you can use the free software Ghostscript (see http://www.cs.wisc.edu/~ghost/).

A do-file generating a graph to be included in a document will look as follows:

```
use archive.dta, replace
/* omitted syntax to produce the graph*/
/*export to eps (or other) format*/
graph export namefile, as(eps)
/*if needed invoke ghostscript*/
/* to high-resolution png*/
winexec gs -q -dEPSCrop -dNOPROMPT -dBATCH -dNOPAUSE -sDEVICE=png16m -r100  ///
  -dGraphicsAlphaBits=4 -sOutputFile=namefile.png namefile.eps
/* or to pdf*/
winexec epstopdf namefile.eps
/*open a file and write the code for inclusion*/
file open page using page.htm, write replace
file write page `"<img src="namefile.png">"' _n
file close page
```

❑ **Technical note**

When asked, Stata will write any kind of text—even code in Stata's own language. For example, to produce a graph whose code is conditioned by the data (for instance, a graph superimposing a line for each year recorded in the data) with a code that does not need to be updated every year, we can write a do-file that first makes Stata write a (possibly temporary) do-file generating the graph and then asks Stata to execute it. ❑

3.3 Writing markup language code: lists

A common nontabular way of presenting information is through lists, and markup languages have an ad-hoc syntax.

In LaTeX,

- an unnumbered list is declared by a `\begin{itemize}` command and closed by an `\end{itemize}` command;
- a numbered list is declared by a `\begin{enumerate}` command and closed by an `\end{enumerate}` command; and

- each element of the list begins with an `\item` command.

In HTML,

- an unnumbered list is declared by a `<ul>` tag and closed by a `</ul>` tag;
- a numbered list is declared by an `<ol>` tag and closed by an `</ol>` tag; and
- each element of the list begins with an `<li>` tag and ends with an `</li>` tag.

3.4 Writing markup language code: trees

The powerful LaTeX package XY-pic, which can draw any kind of diagram, gives a different type of code altogether. In particular, we show the generation of a tree to describe partially ordered paths.

▷ Example

Every year a survey is conducted on a set of college students. The main subjects of study for the college students, as well as the main subjects of study during secondary school, are classified as "Scientific", "Artistic", or "Technical". To represent the pattern of study by the students, a tree is built, and some characteristics of the students are summarized on the tree.

To automate the job, the following Stata code is written, which generates and compiles LaTeX code (in the language of the package XY-pic). In this example, the tree is also decorated with some LaTeX tables, which are generated in the same do-file.

```
capture mkdir files
local format "ps"
use tree0
/*foreach student the path of study is generated*/
sort id ord
by id, sort: gen path=string(event) if ord==1
by id, sort: replace path=path[_n-1]+string(event) if _n>1
/*
omitted code that generates the tables to be included in the tree
*/
/*contracts the archive to the existing paths*/
contract path
/*set the dimension of the page where the tree has to be printed*/
local height=300
local width=200
/*starts computing the coordinates of the nodes of the tree*/
gen int pos=length(path)
qui sum pos
local maxpos=r(max)
gen int weight=1
local i=‘maxpos’
```

```
while `i'>0{
        qui gen parent`i'=substr(path,1,`i')
        by parent`i', sort:egen len`i'=max(pos)
        by parent`i', sort:gen count`i'=_N
        by parent`i', sort:egen count_child`i'=sum(pos==`i'+1)
        by parent`i', sort:egen weight`i'=sum(weight*(pos==`i'+1|`i'==len`i'))
        replace weight=weight`i'+cond(count_child`i'>1,2,0) if pos==`i'
        local i=`i'-1
        }
gen str1 parent0=""
local stepabs=(`width'-5)/(3*`maxpos')
local relabscissa=string(`stepabs',"%3.1f")
qui sum weight
local stepord=(`height'-5)/r(max)
qui gen  relordinate=""
local i=`maxpos'
while `i'>0{
        local ip=`i'-1
        gen str1 event`i'=substr(path,`i',1)  if pos==`i'
        sort parent`ip' event`i'
        by parent`ip', sort:gen int sumweight`i'=weight*(_n==1 & pos==`i')
        by parent`ip', sort:replace sumweight`i'=weight[_n]+sumweight`i'[_n-1]  ///
             if >  _n>1 & pos==`i'
        by parent`ip', sort:egen weightpar=max(sumweight`i')
        replace relordinate=cond(weightpar==1,"0",                              ///
     string(((weightpar+cond(mod(weightpar,2)==0,1,0))/2-sumweight`i')*         ///
     `stepord',"%3.1f")) if pos==`i'
        drop weightpar
        local i=`i'-1
        }
gen parent=cond(pos>1,substr(path,1,pos-1),"")
/*start writing a tex file containing xypic code*/
tempname tree
file open `tree' using files/tree.tex, write replace
file write `tree' `"\documentclass{article}"' _n
file write `tree' `"\usepackage[all]{xypic}"' _n
file write `tree' `"\usepackage[margin=0.5cm, paperwidth=`width'mm,"'
file write `tree' `" paperheight=`height'mm,centering]{geometry}"' _n
file write `tree' `"\usepackage{colortbl}"' _n
file write `tree' `"\begin{document}"' _n
file write `tree' `"\centering"' _n
file write `tree' `"\begin{xy}"' _n
sort path
file write `tree' `" (0,-10)="0A" ,"' _n
file write `tree' `" "0A"+(`relabscissa',0)="0B","' _n
file write `tree' `" "0A";"0B"**@{-},"' _n
local countpaths=_N
```

(*Continued on next page*)

```
forvalues s=1/‘countpaths’{
        if parent[‘s’]!=""{
                local parent=parent[‘s’]
                local path=path[‘s’]
                local relordinate=relordinate[‘s’]
                file write ‘tree’ ‘" "‘parent’B"+(‘relabscissa’,‘relordinate’)+"’
                file write ‘tree’ ‘"(‘relabscissa’,0)="‘path’A" ,"’ _n
                file write ‘tree’ ‘" "‘path’A"+(‘relabscissa’,0)="‘path’B" *"’
                file write ‘tree’ ‘"{\bullet},"’ _n
                file write ‘tree’ ‘" "‘parent’B";"‘path’A"**@{-},"’ _n
                file write ‘tree’ ‘" "‘path’A";"‘path’B"**@{-},"’ _n
                }
        }
forvalues s=1/‘countpaths’{
        local path=path[‘s’]
        file write ‘tree’ ‘" "‘path’A"*[F]\txt{\input{"’
        file write ‘tree’ ‘"‘dir’tabelle/tab‘path’.tex}},"’ _n
        }
file write ‘tree’ ‘"\end{xy}"’ _n
file write ‘tree’ ‘"\end{document}"’ _n
file close ‘tree’
/*compile the tex file to generate the pdf or ps output*/
if "‘format’"=="pdf"{
        ! pdflatex  files/tree
        }
if "‘format’"=="ps"{
        ! latex  files/tree
        ! dvips  files/tree
        }
```

The resulting tree is presented in figure 1.

(*Continued on next page*)

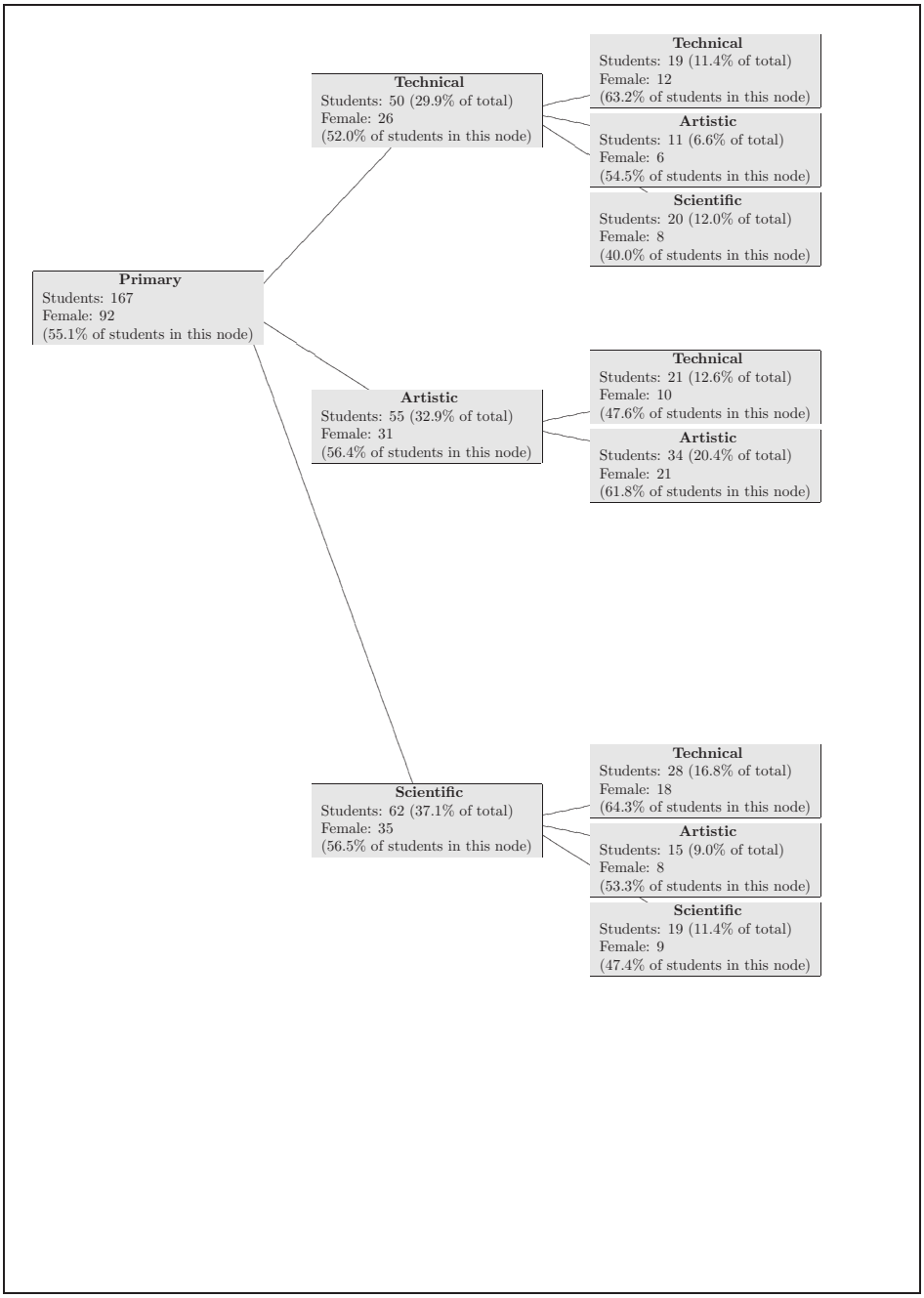

Figure 1: A tree automatically generated

To include a PDF or PS file generated by a sophisticated feature of LaTeX in an HTML document, we can use the technique shown in subsection 3.2.

4 Structure of a do-file automatically producing reports

We are left with the task of assembling tables, graphs, lists, trees, etc., into a coherent document, ready for presentation.

The task seems to be slightly different for the two languages:

- In LaTeX, one file is produced, with a table of contents outlining its navigation (the table of contents can be browsable via hyperlinks, if generated as a PDF file).
- In HTML, all the generated files remain distinct, but they can be browsed by means of a central file, usually named `index.html`.

In fact, the job for producing the two types of documents is similar, since what one does in LaTeX is also generating a "main" document, which during compilation calls all the other files and includes them in one PDF (or PS) file.

We show the construction of a short printable document, a short presentation, and a small web page, all collecting the same information from a small archive from a questionnaire on several aspects of infant care. The archive contains four categorical variables, and we want to give for any of them a simple table of frequencies and a histogram. Again we use `xcontract` and `listtex`.

▷ Example

The following code generates the tables (with the modal values evidenced), captions in LaTeX format, and graphs that are exported to PDF (via the `epstopdf` command of Ghostscript) shown in figure 1.

```
local acroread=cond(c(os)=="Unix","acroread","AcroRd32")
capture mkdir tex
use brfeed
/* for each variable a frequency table and a histogram are generated*/
foreach var of varlist pos natio edu brfeed{
        preserve
        local title:variable label `var'
        xcontract `var', nores
        gen perc=string(_perc,"%3.1f")+"\%"
        gen freq=string(_freq)
        decode `var', gen(`var'_label)
        qui sum _freq
        qui gen modal=(_freq==r(max))
        replace `var'_label="\rowcolor[gray]{.9} "+`var'_label if modal==1
```

```
        listtex ‘var’_label freq perc using tex/tab_‘var’.tex,             ///
             rstyle(tabular) headlines("\begin{tabular}{p{5cm}rr}"         ///
             "\hline \multicolumn{3}{c}{\bf ‘title’} \\\\ "                ///
             "& Frequency&  Percentage \\\\ \cline{2-3} ")                 ///
             footlines("\hline \end{tabular}") replace
        /* a file containing a caption is generated*/
        tempname capt
        file open ‘capt’ using tex/capt_‘var’.tex, write replace
        file write ‘capt’ ‘"\caption{‘title’}"’ _n
        file close ‘capt’
        restore
        /* the graphic is temporarily exported to eps and then translated
           into pdf via ghostscript*/
        qui sum ‘var’
        local min=r(min)
        local max=r(max)
        hist ‘var’, discrete width(.5)                                     ///
             ylabel(‘min’(1)‘max’, valuelabel angle(hor)) xtitle("") freq  ///
             ytitle("") graphregion(color(blue*.4)) bcolor(blue) hor
        tempfile graph
        graph export ‘graph’, as(eps) replace
        winexec epstopdf ‘graph’ --outfile=tex/graph_‘var’.pdf
        }
/* the main document is generated*/
cd tex
tempname main
file open ‘main’ using main_brfeed.tex, write replace
file write ‘main’ ‘"\documentclass{article}"’ _n
file write ‘main’ ‘"\usepackage{graphicx}"’ _n
file write ‘main’ ‘"\usepackage{color}"’ _n
file write ‘main’ ‘"\usepackage{colortbl}"’ _n
file write ‘main’ ‘"\usepackage[format=hang, labelsep=period,"’
file write ‘main’ ‘" labelfont={bf,small},"’
file write ‘main’ ‘" textfont=small,width=0.8\textwidth]{caption}"’ _n
file write ‘main’ ‘"\usepackage{hyperref}"’ _n
file write ‘main’ ‘"\title{Survey on child care behaviours}"’ _n
file write ‘main’ ‘"\begin{document}"’ _n
file write ‘main’ ‘"\maketitle"’ _n
file write ‘main’ ‘"\listoftables"’ _n
foreach var of varlist pos natio edu brfeed{
        file write ‘main’ ‘"\clearpage"’ _n
        file write ‘main’ ‘"\begin{table}\input{capt_‘var’.tex}"’ _n
        file write ‘main’ ‘"\centering"’ _n
        file write ‘main’ ‘"\includegraphics[width=.8\textwidth]"’
        file write ‘main’ ‘"{graph_‘var’.pdf}"’ _n _n
        file write ‘main’ ‘"\bigskip"’ _n
        file write ‘main’ ‘"\input{tab_‘var’.tex}"’ _n
        file write ‘main’ ‘"\end{table}"’ _n
        }
file write ‘main’ ‘"\end{document}"’ _n
file close ‘main’
/* the main document is compiled via pdflatex */
! pdflatex main_brfeed.tex
! pdflatex main_brfeed.tex
/* the main document is opened via acrobat reader*/
winexec ‘acroread’ main_brfeed.pdf
```

Example

To obtain a screen presentation, we need to modify only the last part of the code of the previous example and to add the commands necessary for the Beamer package of LaTeX to produce the desired output. Figure 2 is a snapshot of the presentation.

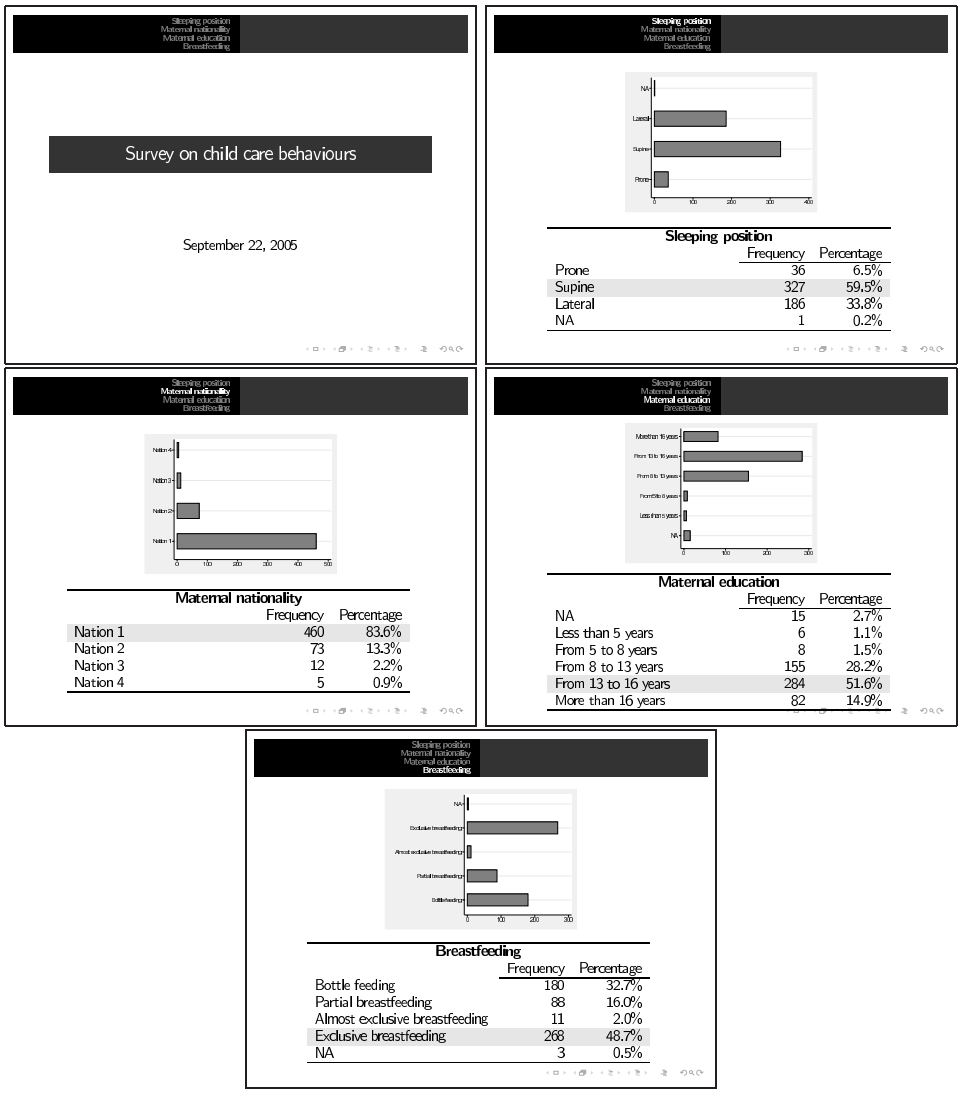

Figure 2: A snapshot of the automatically generated presentation

```
local acroread=cond(c(os)=="Unix","acroread","AcroRd32")
capture mkdir tex
use brfeed
/* the main document is generated */
cd tex
tempname main
file open ‘main’ using main_pres_brfeed.tex, write replace
file write ‘main’ ‘"\documentclass{beamer}"’ _n
file write ‘main’ ‘"\usepackage{beamerthemesplit}"’ _n
file write ‘main’ ‘"\beamertemplatefootempty"’ _n
file write ‘main’ ‘"\beamertemplatenumberedballsectiontoc"’ _n
file write ‘main’ ‘"\usepackage{colortbl}"’ _n
*file write ‘main’ ‘"\usepackage[format=hang, labelsep=period,
*     labelfont={bf,small}, textfont=small,width=0.8\textwidth]{caption}"’ _n
file write ‘main’ ‘""’ _n
file write ‘main’ ‘"\title{Survey on child care behaviours}"’ _n
file write ‘main’ ‘"\begin{document}"’ _n
file write ‘main’ ‘"\frame{\titlepage}"’ _n
foreach var of varlist pos natio edu brfeed{
        local title:variable label ‘var’
        file write ‘main’ ‘"\section{‘title’}"’ _n
        file write ‘main’ ‘"\frame{"’ _n
        file write ‘main’ ‘"\begin{table}"’ _n
        file write ‘main’ ‘"\centering"’ _n
        file write ‘main’ ‘"\includegraphics[width=.5\textwidth]"’
        file write ‘main’ ‘"{graph_‘var’.pdf}"’ _n _n
        file write ‘main’ ‘"\bigskip"’ _n
        file write ‘main’ ‘"\input{tab_‘var’.tex}"’ _n
        file write ‘main’ ‘"\end{table}"’ _n
        file write ‘main’ ‘"}"’ _n
        }
file write ‘main’ ‘"\end{document}"’ _n
file close ‘main’
/* the main document is compiled via pdflatex */
! pdflatex main_pres_brfeed.tex
! pdflatex main_pres_brfeed.tex
/* the main document is opened via acrobat reader*/
winexec ‘acroread’ main_pres_brfeed.pdf
```

◁

▷ Example

Finally, if we want to present the same data on a web page, some modifications must be made to the call to `listtex`, the graphs must be exported in PNG format, and the code of the main page must be written in HTML. The modified code is as follows:

```
local browser=cond(c(os)=="Unix","mozilla","explorer")
local gho=cond(c(os)=="Unix","gs","gswin32")
capture mkdir tex
use brfeed
```

```
/* for each variable a frequency table and a histogram are generated*/
foreach var of varlist pos natio edu brfeed{
        preserve
        local title:variable label `var'
        xcontract `var', nores
        qui sum _freq
        qui gen byte modal=(_freq==r(max))
        gen perc=string(_perc,"%3.1f")+"%"
        gen freq=string(_freq)
        /* in the table the modal values are put in boldface*/
        foreach desc in freq perc{
                replace `desc'="<b>"+ `desc'+" </b>" if modal==1
                }
        listtex `var' freq perc using tex/tab_`var'.htm,rstyle(html)          ///
             headlines("<table align=center width=50%>"                       ///
             "<tr><th colspan="3" align=center>`title'</th></tr> "            ///
             "<tr><td> </td><td> Frequency </td><td>Percentage</td></tr>")    ///
             footlines(" </table>") replace
        restore
        /* the graphic is temporarily exported to eps and then translated into
           high-resolution png via ghostscript*/
        qui sum `var'
        local min=r(min)
        local max=r(max)
        hist `var', discrete width(.5) ylabel(`min'(1)`max', valuelabel       ///
             angle(hor)) xtitle("") freq ytitle("")                           ///
             graphregion(color(blue*.4)) bcolor(blue) hor
        tempfile graph
        graph export `graph'.eps, as(eps) replace
        winexec `gho' -q -dEPSCrop -dNOPROMPT -dBATCH -dNOPAUSE              ///
             -sDEVICE=png16m -r200 -dGraphicsAlphaBits=4                     ///
             -sOutputFile=tex/graph_`var'.png `graph'.eps
        /* the htm file calling the image generated*/
        tempname tabfig
        file open `tabfig' using tex/tabfig_`var'.htm, write replace
        file write `tabfig' `"<table align=center width=90%>"' _n
        file write `tabfig' `"<tr><th>`title'</th></tr>"' _n
        file write `tabfig' `"<tr>"' _n
        file write `tabfig' `"<td valign="top">"' _n
        file write `tabfig' `"<img src="graph_`var'.png" width="90%" "'
        file write `tabfig' `"border="0" align="left" valign="top">"' _n

        file write `tabfig' `"</td>"' _n
        file write `tabfig' `"</tr>"' _n
        file write `tabfig' `"</table>"' _n
        file close `tabfig'
        }
/* the main document is generated */
cd tex
tempname main
file open `main' using main_brfeed.htm, write replace
file write `main' `"<!--File automatically generated by Stata-->"' _n
file write `main' `"<h2 align=center>Survey on child care behaviours<h2>"' _n
file write `main' `"<br><br>"' _n
file write `main' `"<table align=center width=30%>"' _n
file write `main' `"<tr><td><b>Variable</b></td><td><b>Table</b></td>"'
file write `main' `"<td><b>Graph</b></td><tr>"' _n
```

```
foreach var of varlist pos natio edu brfeed{
     local title:variable label ‘var’
     file write ‘main’ ‘"<tr><td>‘title’</td>"’
     file write ‘main’ ‘"<td><a Href=’tab_‘var’.htm’>[X]<a></td>"’
     file write ‘main’ ‘"<td><a Href=’tabfig_‘var’.htm’>[X]<a></td><tr>"’ _n
                                          }
file write ‘main’ ‘"</table>"’ _n
/* the main document is opened via a browser*/
winexec ‘browser’ main_brfeed.htm
```

5 A practical web example

The web page “Indicatori sulla Salute e l’Assistenza agli Anziani” of the Regional Agency for Public Health of Tuscany (http://www.arsanita.toscana.it/) is constructed by means of a Stata procedure.

The source of data is a dataset created by a series of SQL queries performed on the regional archive of health information. These data permit computation of 15 different quality indicators, whose values can be aggregated in several geographical and/or temporal levels.

Computing such indicators and organizing the output in a useful, understandable, and profitable way for different professional profiles (e.g., epidemiologists, politicians, medical doctors) is a difficult and complex task. For this reason, it was considered worth investing energy in building the report’s structure the first time and then using the same procedure yearly, whenever the regional archive is updated.

A do-file does the following:

1. Computes indicators and confidence intervals for every aggregation and saves them in DTA archives.
2. Writes a collection of HTML main pages, linked to each other.
3. Generates 125 graphs (eight for each indicator and five summary graphs) and exports them in `png` format in the same location (path) referred by the links in the corresponding HTML main pages.
4. Generates 12 HTML lists and saves them in the same location (path) referred by the links in the corresponding HTML main pages.
5. Generates 50 HTML tables and saves them in the same location (path) referred by the links in the corresponding HTML main pages.

6. Generates the same 50 tables and 12 lists in TEX format, exports the same 125 graphs in PDF, writes some summary TEX documents, compiles these TEX documents, and saves the resulting PDF documents in the same location (path) referred by the links in the corresponding HTML main pages—where they can be downloaded for printing by the user.

This program takes about 5 minutes to execute.

6 Problems and conclusion

Using Stata as illustrated above is sometimes complicated because Stata programs do not systematically store as saved results (in the form of scalars, matrices, or datasets) all the information needed to reconstruct the screen output—which requires modifying the original code of the command. For instance, in building the web site http://www.arsanita.toscana.it/ described in section 5, it was necessary to manipulate the code of `dstdize`, which at the time (2003) did not leave behind the saved results for calculating confidence intervals. We think that the Stata support and development community should be aware of the possibilities offered by this modified method, and we hope that leaving the essential computations of each command as saved results, at least as an option, becomes a habit. It would be even better if the interaction of the command with such universal tools as `statsby` (see [D] **statsby**) and `parmby` (by Roger Newson and available on SSC) was kept in mind when writing any estimation command. For instance, this interaction is still not possible with `dstdize`, because `statsby` accepts only scalar results as an input, whereas `dstdize` saves results in matrices. Therefore such an innocent task as producing a dataset with confidence intervals of both crude and standardized rates (with `ci`/`cii` and `dstdize`) requires performing a surprisingly long list of operations.

To take advantage of the suggested method, one must learn the basics of layout formatting in a markup language: the more one gets familiar with a formatting language, the more one can create complex, deep, and beautiful documents, with no virtual limit. LATEX and HTML, with their development and support communities, allow users to freely find complete documentation and resources on the web.

Stata's `file` commands can be combined with any other programming language to build not only reports but also programs.

7 Acknowledgments

We thank Una-Louise Bell, Nicholas Cox, Bianca De Stavola, and Marcello Pagano for useful discussions on the topics of this paper. Moreover, we thank the editor, Joe Newton, for pointing out the excellent paper by Roger Newson. Finally, we thank the referee for useful remarks.

8 References

Lamport, L. 1994. *LaTeX: A Document Preparation System*. 2nd ed. Reading, MA: Addison–Wesley.

Newson, R. 2003. Confidence intervals and p-values for delivery to the end user. *Stata Journal* 3: 245–269.

About the authors

Rosa Gini has a PhD in mathematics and works mainly on monitoring tools for the quality of public health systems. She is a member of the Italian TeX User Group (GUIT).

Jacopo Pasquini is a statistician studying for his PhD in methods for social research, and his interests are dedicated mainly to social epidemiology.

They both work as senior researchers at the Regional Agency for Public Health of Tuscany, in Florence, Italy.