



The World's Largest Open Access Agricultural & Applied Economics Digital Library

This document is discoverable and free to researchers across the globe due to the work of AgEcon Search.

Help ensure our sustainability.

Give to AgEcon Search

AgEcon Search

<http://ageconsearch.umn.edu>

aesearch@umn.edu

*Papers downloaded from **AgEcon Search** may be used for non-commercial purposes and personal study only. No other use, including posting to another Internet site, is permitted without permission from the copyright owner (not AgEcon Search), or as allowed under the provisions of Fair Use, U.S. Copyright Act, Title 17 U.S.C.*

No endorsement of AgEcon Search or its fundraising activities by the author(s) of the following work or their employer(s) is intended or implied.

The Stata Journal (2016)
16, Number 4, pp. 938–963

Rethinking literate programming in statistics

E. F. Haghish
Center for Medical Biometry and Medical Informatics
University of Freiburg
Freiburg, Germany
and
Department of Mathematics and Computer Science
University of Odense
Odense, Denmark
haghish@imada.sdu.dk

Abstract. Literate programming is becoming increasingly trendy for data analysis because it allows the generation of dynamic-analysis reports for communicating data analysis and eliminates untraceable human errors in analysis reports. Traditionally, literate programming includes separate processes for compiling the code and preparing the documentation. While this workflow might be satisfactory for software documentation, it is not ideal for writing statistical analysis reports. Instead, these processes should run in parallel. In this article, I introduce the **weaver** package, which examines this idea by creating a new log system in HTML or \LaTeX that can be used simultaneously with the Stata log system. The new log system provides many features that the Stata log system lacks; for example, it can render mathematical notations, insert figures, create publication-ready dynamic tables, and style text, and it includes a built-in syntax highlighter. The **weaver** package also produces dynamic PDF documents by converting the HTML log to PDF or by typesetting the \LaTeX log and thus provides a real-time preview of the document without recompiling the code. I also discuss potential applications of the **weaver** package.

Keywords: pr0063, weave, div, img, txt, tbl, weaver, weaving, literate programming, dynamic documents, reproducible research, log, HTML, \LaTeX

1 Introduction

Statistical analyses are carried out to draw conclusions from data, gain knowledge, and communicate claims with the scientific community. However, the communication aspect of statistics is often ignored in statistical software.

Traditionally, the analysis report is written manually using a word processor such as Microsoft Word by dragging and dropping figures, creating tables, and copying and pasting output from the statistical software into the document. This workflow is unreproducible and becomes more prone to human errors as the complexity of the report increases. More importantly, when figures and output are copied and pasted into the analysis report, they become disconnected from the code that produced them, potentially leading to more human errors that are not traceable. Automating the writing of analysis documents can save a lot of time and human labor, reduce the chance of human error in writing the report, and make the report reproducible.

The common approach for creating automated analysis reports in statistics is “literate programming”, a programming paradigm introduced by Knuth (1983, 1984, 1992). In literate programming, the documentation is written within the source files and is separated from the programming code with a special notation. Next, a literate-programming software (for which the special notation is written) parses the string notations to separate the programming code from the documentation and continues to either compile the software or prepare the documentation. Knuth names the process of compiling the software “tangle” and the process of preparing the documentation “weave”.

Software engineers advocate literate programming because it eases the effort needed to update the software documentation and it makes the source more comprehensible for maintenance programmers. In addition, it preserves the connection between the code and the documentation (Sametinger and Pomberger 1992; Ramsey and Marceau 1991; Cordes and Brown 1991), although its applications are not merely limited to software documentation (Shum and Cook 1994; Ruys and Brinksma 1998).

Literate programming is becoming a popular way to document data analysis and generate analysis reports, with literate-programming software that is independent of the target programming language, such as noweb (Johnson and Johnson 1997; Ramsey 1994) or any literate-programming software developed especially for producing data-analysis reports (Murdoch and Carey 2001; Xie 2016, 2014; Leisch 2002; Allaire et al. 2016; Lenth 2012). For Stata users, `markdoc`¹ (Haghish 2014a) is a convenient general-purpose literate-programming package that supports a variety of markup languages and document formats. This software package is inspired by Knuth’s (1983) literate-programming software called WEB and thus applies a special notation for separating the documentation text from the data-analysis code in the source files.

However, there is a major difference between software development and statistics when it comes to code communication. Computer programs are primarily written to be compiled by computers, and literate programming is the preferred method for software documentation. In contrast, the cornerstone of statistics lies in communicating the statistical analysis and interpreting the results. For example, while software documentation is presumably optional, interpreting the statistical results is the natural step after any data analysis. Moreover, data analysis is often carried out interactively by executing consequent commands, and as a result, a dynamic report should also be developed gradually and updated automatically with every new executed command. In this way, a user can obtain a live preview of his or her analysis report as the data analysis develops.

In contrast to software development, literate-programming packages in statistics should compile the analysis code either way because the analysis report must include output and figures generated from the statistics software. In other words, the code must be compiled whether the user wishes to create an analysis report or just carry out the data analysis without producing documentation. Thus, the question is why statisticians insist on executing tangle and weave independently when these are better

1. `markdoc` is hosted on <https://github.com/haghish/markdoc>.

run in parallel to save on execution time and provide support for interactive analysis report development.

To examine this idea, I developed the **weaver** package for Stata. It can be imagined as a new kind of analysis log system with distinctive behavior—that I refer to as **weaver** log—that can run in parallel with Stata Markup and Control Language (SMCL) or text log system. Because the **weaver** log system can produce HTML and \LaTeX documents, I will refer to a particular format as HTML log or \LaTeX log, and to avoid confusion with the native log system of Stata, I will refer to the latter as Stata log or SMCL log. The **weaver** log system includes a variety of features one could expect from a software that generates analysis reports. I present these features in the current article as well as discuss potential applications of **weaver** and how it can be better implemented in Stata.

2 The weaver package

2.1 The main idea

The **weaver** log system can be used simultaneously with SMCL or text log system, but for a very different purpose. From the time the SMCL log is opened, it documents every action that takes place during the analysis session, including commands, text-based output, and also errors. Therefore, it is a reliable way of saving the history of the data-analysis session. In contrast, the **weaver** log is not autonomous and remains indifferent to all Stata commands and outputs. It does not automatically register anything unless the user commands it otherwise. Such behavior allows the user to select what text, figures, commands, and output should be included in the **weaver** log file while the SMCL log documents the analysis in parallel.

2.2 Features

The **weaver** package supports HTML and \LaTeX for creating dynamic-analysis reports, without requiring any knowledge of these markup languages. **weaver** includes a simplified JavaScript-based markup language called **weaver** markup that allows writing text headings and paragraphs, styling text, changing text color, highlighting text, and adding hypertext links in the HTML log (see section 4.1). Moreover, **weaver** also comes with a syntax highlighter for Stata code in the HTML and \LaTeX dynamic document. For conveniently writing dynamic text, adding figures, and creating dynamic tables in both the HTML and the \LaTeX log files, **weaver** includes three specialized commands. Additionally, **weaver** supports rendering mathematical notations in HTML and \LaTeX log files. Finally, both the HTML or \LaTeX log and the rendered PDF document can be previewed in real time while developing the dynamic report.

2.3 Installation

weaver requires the **statax** (Haghish 2015) package, a syntax highlighter for HTML and \LaTeX dynamic documents. Both packages are hosted on GitHub website. The **github**² (Haghish 2016a) command can install **weaver** and its dependency as shown below.

```
. github install haghish/weaver
```

In addition, the HTML log requires some third-party software: wkhtmltopdf and MathJax. The wkhtmltopdf freeware provides command-line tools for converting HTML to PDF and accurately renders CSS and JavaScript to PDF. MathJax is a JavaScript engine for rendering mathematical notations in the HTML log file. Both MathJax and wkhtmltopdf are open-source freeware, available for Microsoft Windows, Mac OS X, and Linux operating systems. Naturally, for typesetting a PDF document from the \LaTeX log file, you should install a \LaTeX distribution on your machine. To use **weaver** without any third-party software installation, include the **nopdf** option to cancel the process of rendering a PDF document. This option will avoid errors regarding MathJax, wkhtmltopdf, and pdfLaTeX but will still produce an HTML or a \LaTeX dynamic document.

The GitHub repository hosts the ado-file and help files. The required third-party software should be installed manually. To make the installation process more convenient, **weaver** provides an optional automatic installation for MathJax and wkhtmltopdf. Both manual and automatic installation of the required software are described below.

Manual installation

Manual installation of wkhtmltopdf is straightforward. The software can be downloaded from <http://wkhtmltopdf.org> and installed in any location on your machine. The path to the executable wkhtmltopdf should be given to **weaver** by using the **printer(string)** option. Similarly, to typeset the \LaTeX log to PDF, the proper \LaTeX distribution should be downloaded from <https://latex-project.org> and the path to the executable pdfLaTeX should be provided using the **printer(string)** option. The manual installation of MathJax is easier. The **MathJax-master.zip** file can be downloaded from <http://mathjax.org> and unzipped in the **ado\plus\Weaver** directory (see the technical note below).

As an alternative to specifying the **printer(string)** option, users can set up the default file paths for PDF drivers or install MathJax in the **Weaver** directory permanently. The **weave setup** command opens **weaversetup.ado** in the Do-file Editor. It defines global macros for memorizing the full paths to the executable wkhtmltopdf, the executable pdfLaTeX, and the MathJax master directory for future use. Note that **weaversetup.ado** is not included in the package and is created the first time the **weave setup** command is executed.

2. To install the **github** command, type

```
net install github, from("https://haghish.github.io/github/").
```

Automatic installation

When opening a **weaver** log, the **install** option can be used to download wkhtmltopdf and MathJax automatically and install them in the **Weaver** directory (see the technical note below). The automatic installation of the required software was successfully tested on Mac OS X (10.9, 10.10, 10.11); 32-bit and 64-bit versions of Microsoft Windows (XP, 7, 8); and 64-bit versions of Linux Ubuntu 14.04.4 LTS, Mint 17, and CentOS 7. However, manual installation is generally recommended, because it ensures the installation of the latest version of the software.

□ Technical note

weaver creates a directory named **Weaver** in the **plus** directory to store the required software. The path to the **ado\plus** directory can be found using the **sysdir** command, which returns all the system directories. In Stata 13 and 14, the likely paths of the **Weaver** directory on Microsoft Windows, Mac OS X, and Linux operating systems are shown below.

Windows: C:\ado\plus\Weaver

Mac OS X: ~/Library/Application Support/Stata/ado/plus/Weaver

Linux: /home/username/ado/plus/Weaver

□

3 Syntax and examples

weaver includes two sets of commands. The commands that begin with **weave** are related to handling the **weaver** log, for example, opening a new log file, merging existing HTML or L^AT_EX documents into the current log, exporting a live preview PDF document, closing the log, or temporarily turning the log on or off. The other commands are used for annotating the document, that is, inserting a figure, creating a dynamic table, echoing Stata commands and outputs, and writing and styling text in the log file.

3.1 The weave commands

The commands related to managing the **weaver** log file begin with the **weave** command and are summarized in table 1. The syntax of these commands is similar to the Stata **log** command syntax.

Table 1. The **weave** commands

| Command | Description |
|--------------------------------------------------------|--------------------------------------------|
| weave using <i>filename</i> [, <i>options</i>] | create a new log |
| weave merge using <i>filename</i> | merge a file into the log |
| weave pdf [, <i>replace</i>] | create a PDF preview of the log file |
| weave close | close the log and create the PDF |
| weave query | report the status of the log file |
| weave { <i>off</i> <i>on</i> } | turn the log off or on |
| weave { <i>preserve</i> <i>restore</i> } | preserve and restore the log status |
| weave setup | set the file path to the required software |

Opening a new log

The dynamic report begins with the command **weave using** *filename*, which creates a new log file. This log stays on until it is closed, turned off, or Stata is shut down. The overall syntax for opening a log file is shown below.

```
weave using filename [, install nopdf printer(string) markup(name)
  {replace | append} papersize(name) landscape margin(int,int,int,int)
  style(name) template(filename) toc font(string) title(string)
  author(string) affiliation(string) address(string) summary(string) date
  synoff noisily]
```

The **weave using** *filename* command accepts several options that determine the general format of the log and the PDF dynamic report, such as paper size, margins, and styles that can change the overall appearance of the document. These options are further described below.

install automatically downloads and installs the wkhtmltopdf and MathJax software if they are not located in the **Weaver** directory or cannot be accessed by **weaver**.

nopdf avoids rendering PDF and mathematical notations (in HTML log). This option allows the user to use **weaver** HTML or L^AT_EX dynamic reports without installing any third-party software.

printer(*string*) defines the file path of the executable wkhtmltopdf software or of pdfLaTeX on your machine.

markup(*name*) defines the log type as **html** or **latex**. If the *filename* already includes a file extension (that is, **.html** or **.tex**), then **weaver** will recognize the markup language and this option is unnecessary. The default is **markup(html)**.

`replace` overwrites an existing `weaver` log and PDF files with the identical name.

`append` specifies that results be appended to an existing HTML or \LaTeX log file.

`papersize(name)` changes the default page size of the PDF document. The default is `papersize(A4)`, but `weaver` supports most of the widely used page sizes for generating printer-ready PDF documents from both the HTML and the \LaTeX log files. The supported paper sizes are A0, A1, A2, A3, A4, A5, A6, A7, A8, A9, B0, B1, B2, B3, B4, B5, B6, B7, B8, B9, B10, C5E, Comm10E, DLE, Executive, Folio, Ledger, Legal, Letter, and Tabloid. These names may also be specified in lowercase. In addition, the `weave setup` command may be used to permanently change the default paper size.

`landscape` flips the page to landscape mode.

`margin(int,int,int,int)` changes the default margins of the PDF document and can range between 3 to 50 millimeters. Each *int* indicates a page margin, beginning with the top margin and moving clockwise; from left to right, the integers indicate the top margin, right margin, bottom margin, and left margin.

`style(name)` changes the theme style of the `weaver` log. The available styles are `stata` (the default), `classic`, `modern`, `elegant`, `minimal`, and `empty`. The `empty` style creates an empty log for HTML or \LaTeX to allow advanced users to manually customize the dynamic document.

`template(filename)` imports an external CSS file for the HTML log or the heading of a \LaTeX document for the \LaTeX log to overrule the existing style sheet of the dynamic report.

`toc` adds an automatic table of contents and creates a reference link to the corresponding headings. The table of contents will only appear in the PDF documents.

`font(string)` specifies the font for all headings, subheadings, and paragraphs.

`title(string)` displays the title of the document on the title page.

`author(string)` displays the author's name on the title page.

`affiliation(string)` displays the author's affiliation (or any preferred relevant information) on the title page.

`address(string)` displays the author's contact information (or any preferred relevant information) on the title page.

`summary(string)` displays a summary or abstract on the title page.

`date` displays the current date on the title page.

`synoff` turns the `statax` syntax highlighter off in the dynamic document.

`noisily` displays the command's output in the Results window.

As noted, `weaver` can create HTML or \LaTeX dynamic reports. The syntax for creating a \LaTeX log is similar to that for HTML, however, the simplified `weaver` markup

(see section 4.1) is not supported in \LaTeX because it is written in JavaScript. Because writing with `weaver` markup is simpler and more intuitive than \LaTeX , most of the examples in this article are written for an HTML log.

► Example

```
set more off
weave using example, install markup(html) replace date      ///
title("Opening HTML Log") author("E. F. Haghish")          ///
affiliation("University of Freiburg")                        ///
summary("This is an example of opening an HTML log in the " ///
"[mono]weaver[#] package. The following examples should be " ///
"run after opening the HTML log. Note that [mono]weaver[#] " ///
"markup codes may be used wherever to annotate the document. " ///
"However, the examples cover only the basic features of the " ///
"[mono]weaver[#] package. All the examples in the current " ///
"section may be downloaded from "                           ///
"[-- https://github.com/haghish/weaver --][-GitHub-].")
```

Opening HTML Log

E. F. Haghish
University of Freiburg
20 Nov 2016

This is an example of opening an HTML log in the `weaver` package. The following examples should be run after opening the HTML log. Note that `weaver` markup code may be used wherever to annotate the document. However, the examples cover only the basic features of the `weaver` package. All the examples in the current section may be downloaded from [GitHub](https://github.com).

Figure 1. HTML preview of the `weave using example`

◀

Creating a PDF preview of the dynamic report

The `weaver` log file is the building block of the dynamic report in the `weaver` package. However, the HTML log file is only interesting for sharing analysis on the web, and most users are probably interested in PDF documents, especially if the log is written in \LaTeX . The PDF document is more practical than HTML or \LaTeX because it is a

single file that includes the report and the graphical figures; thus, it is highly portable and easily sharable. Therefore, it is important to see how the document looks in PDF format during the data analysis, without having to rerun whole codes to compile a PDF document. The command below may be used anytime during the data-analysis session to view the PDF document.

```
weave pdf [ , replace ]
```

This command will generate a dynamic PDF document by converting the HTML log to PDF format using `wkhtmltopdf` or by converting the \LaTeX log to PDF format using `pdfLaTeX`. The final dynamic report is produced by `weave close`, which closes the log file and renders the dynamic report.

Merging a file to the weaver log

Plenty of Stata packages can produce HTML and \LaTeX documents, such as `texdoc` (Jann 2016), `markdoc` (Haghighi 2014a), `synlight` (Haghighi 2014b), `estout` (Jann 2004), and `tabout` (Watson 2004). `weaver` can merge any web-based file or \LaTeX document to the HTML or \LaTeX log, respectively. The syntax for merging a document into the log is as follows:

```
weave merge using filename
```

Both HTML and \LaTeX documents have a straightforward structure, beginning with a heading followed by the body of the document, which includes the main content of the file. Therefore, merging two documents is not as simple as appending one file to another because the template of the merged file can interfere with `weaver`'s template (in HTML) or corrupt the log (in \LaTeX).

`weaver` processes the given file and only merges the main body of the document to avoid any damage to the log file. Content that is omitted in the merging process will be included in the log file as comments (which only can be seen in the source of the HTML and \LaTeX logs), and the user will be notified about the ignored content.

Restoring the weaver log

When writing a dynamic document interactively, you might wish to undo an analysis or remove a Stata outcome from the document. By using `weave preserve` and `weave restore`, you can preserve the status of an HTML or a \LaTeX log and later restore the log to undo changes. The commands do not, however, preserve or restore the data.

3.2 Annotating commands

The annotating commands, as the name suggests, are used for writing the dynamic report, adding mathematical notation, inserting Stata code and output, importing graphs,

and creating dynamic tables. Table 2 lists the annotating commands with a brief description of each.

Table 2. The annotating commands

| Command | Description |
|---------------------------------------------------------------|----------------------------------------|
| <code>div [<u>c</u>ode <u>r</u>esult] <i>command</i></code> | print code, output, or both in the log |
| <code>txt [<u>c</u>ode] <i>display_directive</i></code> | print text, append code |
| <code>img [<u>u</u>sing <i>filename</i>]</code> | insert a graph |
| <code>tbl (*[,*...] [\ *[,*...] [\ [...]]])</code> | create a dynamic table |

3.3 The div command

By default, **weaver** does not include any of the Stata commands and output run from the do-file or typed in the Command window. This allows you to perform exploratory analysis without worrying that the dynamic report will contain output that is not essential or interesting. However, both Stata code and output can be echoed in the **weaver** log by using the **div** command placed before any Stata command. The syntax of the **div** command is shown below.

```
div [code | result] command
```

The **code** or **result** subcommand may be added to **div** to echo only the Stata command or the output, respectively. These subcommands would be used when you intend to include a chunk of code or only the output of a command in the log, making the document less technical and more formal.

► Example

To demonstrate the **div** command, we load **auto.dta** and list the first five observations. Because the **div** command is not specified, **weaver** will not include them in the document. Nevertheless, the output will be shown in the Results window and SMCL log file, if it has been opened. The next three commands, which begin with the **div** command, allow Stata to communicate with the **weaver** log. Figure 2 shows the contents of the **weaver** log after all five commands.

```
sysuse auto, clear
list in 1/5
div misstable summarize
div summarize price mpg
div tabulate foreign
```

```
misstable summarize
```

| | | | | Obs<. | | |
|----------|-------|-------|-------|---------------|-----|-----|
| Variable | Obs=. | Obs>. | Obs<. | Unique values | Min | Max |
| rep78 | 5 | | 69 | 5 | 1 | 5 |

```
summarize price mpg
```

| Variable | Obs | Mean | Std. Dev. | Min | Max |
|----------|-----|----------|-----------|------|-------|
| price | 74 | 6165.257 | 2949.496 | 3291 | 15906 |
| mpg | 74 | 21.2973 | 5.785563 | 12 | 41 |

```
tabulate foreign
```

| Car type | Freq. | Percent | Cum. |
|----------|-------|---------|--------|
| Domestic | 52 | 70.27 | 70.27 |
| Foreign | 22 | 29.73 | 100.00 |
| Total | 74 | 100.00 | |

Figure 2. Preview of the **weaver** log

The `code` subcommand modifies the `div` command to echo only the command and not any resulting output. In the following example, the `code` subcommand is used successively to create a code block as shown in figure 3.

```
div code misstable summarize
div code summarize price mpg
div code tabulate foreign
```

```
misstable summarize
```

```
summarize price mpg
```

```
tabulate foreign
```

Figure 3. The **weaver** log after `div code`

To echo only the Stata output in the **weaver** log, use the **result** subcommand, as shown below; see figure 4.

```
div result regress price mpg
```

| Source | SS | df | MS | Number of obs | = | 74 |
|----------|-----------|----|------------|---------------|---|--------|
| Model | 139449474 | 1 | 139449474 | F(1, 72) | = | 20.26 |
| Residual | 495615923 | 72 | 6883554.48 | Prob > F | = | 0.0000 |
| Total | 635065396 | 73 | 8699525.97 | R-squared | = | 0.2196 |
| | | | | Adj R-squared | = | 0.2087 |
| | | | | Root MSE | = | 2623.7 |

| price | Coeff. | Std. Err. | t | P> t | [95% Conf. Interval] | |
|-------|-----------|-----------|-------|-------|----------------------|-----------|
| mpg | -238.8943 | 53.07669 | -4.50 | 0.000 | -344.7008 | -133.0879 |
| _cons | 11253.06 | 1170.813 | 9.61 | 0.000 | 8919.088 | 13587.03 |

Figure 4. The **weaver** log after **div result**

◀

3.4 The **img** command

Software that generates dynamic reports should handle inserting images and graphs in the document with great care. With **weaver**, you can resize and style graphs in the document, and you can also add a text description by using the **title(string)** option. The **img** command has two options for inserting figures in the **weaver** log. The syntax of the **img** command is shown below.

```
img [using filename] [, title(string) width(int) height(int)
    {left|center}]
```

In the first option, if **img using filename** is specified, **img** adds the markup code required for rendering the graphical file in the HTML or \LaTeX log. In this procedure, the Stata figures should be first exported and then imported in the **weaver** log. For an HTML log, the graph should be in a format recognizable by web browsers (for example, PNG, JPEG, or GIF). In addition to graphical formats supported in HTML, the \LaTeX log also supports PDF figures without requiring the loading of an additional package.

The second option for importing a figure from Stata requires less coding. As shown in the syntax, specifying **using filename** is optional. If the *filename* is not specified, the **img** command automatically exports the current graph from Stata to PNG format in a directory named **Weaver-figure**, located in the same directory as the **weaver** log. Therefore, the **img** command can be executed right after generating a graph in Stata to automatically export it and import it into the dynamic document.

The **width(int)** and **height(int)** options allow the user to resize an image in the document without changing the actual size of the image. The **title(string)** option can be used to write a description about the figure, and the **left** and **center** options align

the image to the left side or to the center of the document, respectively. The example below demonstrates how to insert a graph with the `img` command; see figure 5.

► Example

```
histogram price, title("Price histogram")
img
```

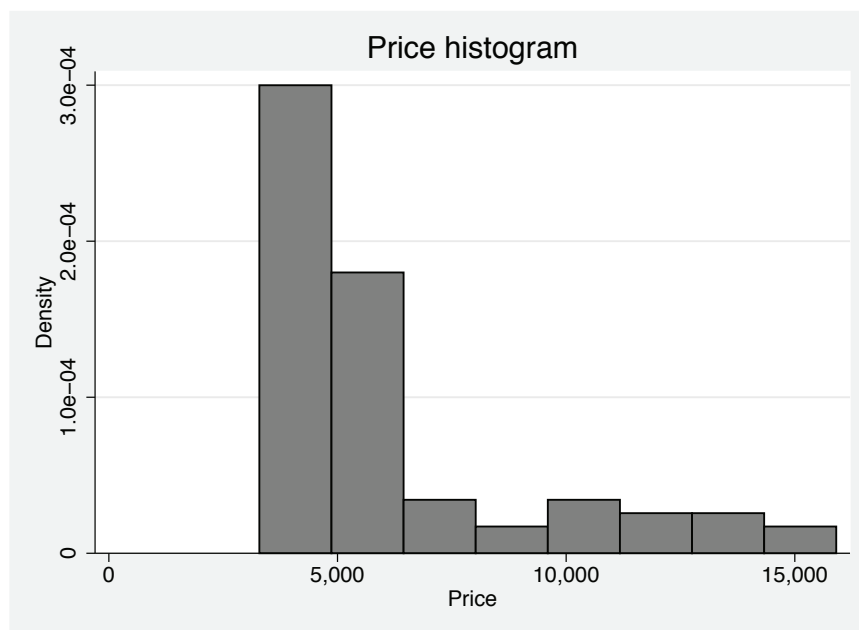


Figure 5. Result of the `img` command

◀

3.5 The `txt` command

The `txt` command acts similarly to the `display` command, and it can be used to display dynamic text in `weaver`. Dynamic text includes scalars and macros, and thus its content can vary. The syntax of this command is as follows:

```
txt [code] display_directive
```

display_directive may be a double-quoted or compound double-quoted string, a display format (for example, `%5.2f`), `_skip(#)`, `_column(#)`, `_newline(#)`, or `_dup(#)`. By default, the `txt` command prints a text paragraph if no markup language is used to alter the documentation. The optional `code` suffix allows writing unformatted text on

the `weaver` log, which is only useful for advanced users who wish to write HTML code or paste a chunk of HTML code or a script in the HTML log. `code` does not change the function of the `txt` command in the L^AT_EX log.

As shown in the example below (see figure 6), in the dynamic document the `txt` command automatically displays scalars with two decimal digits, which is common in scientific writing. A display format value can be used to alter the way the scalar value is displayed.

► Example

```
quietly summarize price

txt "-- Writing dynamic text with the txt command --"          ///
"The [mono]txt[#] command can be used for writing dynamic text, " ///
"(that is, displaying macros in the text paragraph). For example, " ///
"the [mono]price[#] variable in [mono]auto.dta[#] includes "    ///
"#_`r(N)`_# observations with a mean of #_" r(mean) "_# and a "  ///
"standard deviation of #_" r(sd) "_#."

txt [box] Note that [mono]weaver[#] markup codes can be applied  ///
in the [mono]txt[#] command to style text. For example, the     ///
text color can be changed to [blue] blue[#], [red]red[#], or any  ///
of the other supported colors. Similarly, text can be           ///
[-yellow]highlighted[#] in variety of colors.[#]
```

Writing dynamic text with the `txt` command

The `txt` command can be used for writing dynamic text, (that is, displaying macros in the text paragraph). For example, the `price` variable in `auto.dta` includes 74 observations with a mean of 6165.26 and a standard deviation of 2949.50.

Note that `weaver` markup codes can be applied in the `txt` command to style text. For example, the text color can be changed to blue, red, or any of the other supported colors. Similarly, text can be highlighted in variety of colors.

Figure 6. Dynamic text styled using `weaver` markup

◀

The `txt` command can also be used to write mathematical notations, which is explained in section 5.3.

3.6 The `tbl` command

The `tbl` command can be used to create publication-ready dynamic tables in the dynamic report. The syntax is similar to that of Stata's `matrix input` command, which is used for manually defining a matrix. When creating the dynamic table, the table can

be thought of as a matrix with its content defined row by row, from top to bottom. The options of the `tbl` command are identical to the options of the `img` command. The syntax is as follows:

```
tbl (*[, *... ] [\ *[,*... ] [\ [ ... ] ] ) [ , title(string) width(int)
      height(int) {left|center} ]
```

The `*` represents a display directive, which may be a double-quoted string, compound double-quoted string, macro, or scalar. This command also allows merging cells horizontally with `{col #}` directives, where `#` represents the number of columns to be merged. By default, `tbl` left-aligns the first column and centers the other columns. However, the content of each column may be left-aligned, centered, or right-aligned by using the `{l}`, `{c}`, or `{r}` directive, respectively. Examples of column alignment can be found in the help file.

In the example below, the `summarize` command is used twice to obtain the number of observations, means, and standard deviations of two variables from `auto.dta`. To demonstrate storing and displaying numbers in the dynamic table (see figure 6), the content of the first row was saved in local macros and the content of the second row was saved in scalars.

► Example

```
quietly summarize price
local priceobs `r(N)`
local pricemean : display %7.2f `r(mean)`
local pricesd   : display %7.2f `r(sd)`
quietly summarize mpg
tbl ("Variable", "Observations", "Mean", "SD" \           ///
    "__price__#", `priceobs`, `pricemean`, `pricesd` \     ///
    "__mpg__#", r(N), r(mean), r(sd) )
```

| Variable | Observations | Mean | SD |
|----------|--------------|---------|---------|
| price | 74 | 6165.26 | 2949.50 |
| mpg | 74 | 21.30 | 5.79 |

Figure 7. A dynamic table

□ Technical note

Like the `txt` command, the `tbl` command renders numeric scalars with two decimal digits, which is commonly practiced. The `tbl` command deals with numeric macros and scalars differently, however, and the content of the macros is not reformatted automatically. Nevertheless, the content of any numeric macro or scalar can be reformatted (see the example in section 8).

□

4 Supported markup languages

In addition to HTML and \LaTeX , which are the native markup languages for annotating the content of the log file, the `weaver` package introduces a simplified JavaScript markup language for the HTML log called `weaver` markup. Furthermore, `weaver` supports rendering \LaTeX mathematical notations in both the HTML and the \LaTeX log files.

4.1 `weaver` markup

Inspired by Markdown ([Haghish 2016b](#)), which is a simple and intuitive markup language, a new markup language was developed for the `weaver` package to provide all the text styling options that a basic text editor can provide, such as bolding or italicizing text, writing headings and subheadings, changing text color, highlighting text, and adding hypertext links. The `weaver` markup language was written in JavaScript and thus is only supported in the HTML log. It can be used along with HTML code to style text anywhere in the document, including within the `txt` command and all options that allow the display of descriptive text for graphs, tables, and even the title page of the document. Wherever text appears in the HTML log, `weaver` markup can be used to style it. Table 3 describes the codes used in the `weaver` markup language.

Table 3. The `weaver` markup language

| Markup code | Result |
|---------------------------------------------------|-------------------------------|
| Headings | |
| <code>*- Heading 1 -*</code> | Heading 1 |
| <code>*-- Heading 2 --*</code> | Heading 2 |
| <code>*--- Heading 3 ---*</code> | Heading 3 |
| <code>*---- Heading 4 ----*</code> | Heading 4 |
| Page and line breaks | |
| <code>line-break</code> | add an empty line |
| <code>page-break</code> | begin a new page (in PDF) |
| Text styling | |
| <code>** Underscored **</code> | <u>Underscored</u> |
| <code>#_ Italic _#</code> | <i>Italic</i> |
| <code>#__ Bold __#</code> | Bold |
| <code>#___ Italic and Bold ___#</code> | <i>Italic and Bold</i> |
| <code>[center] text [#]</code> | align text to center |
| <code>[right] text [#]</code> | align text to right |
| <code>[box] text [#]</code> | print text in a colored box |
| <code>[mono] text [#]</code> | display monospace text |
| Text color | |
| <code>[blue] blue color text [#]</code> | blue color text |
| <code>[green] green color text [#]</code> | green color text |
| <code>[red] red color text [#]</code> | red color text |
| <code>[purple] purple color text [#]</code> | purple color text |
| <code>[pink] pink color text [#]</code> | pink color text |
| <code>[orange] blue color text [#]</code> | orange color text |
| Text highlight | |
| <code>[-blue] blue color highlight [#]</code> | blue color highlight |
| <code>[-green] green color highlight [#]</code> | green color highlight |
| <code>[-purple] purple color highlight [#]</code> | purple color highlight |
| <code>[-pink] pink color highlight [#]</code> | pink color highlight |
| <code>[-yellow] yellow color highlight [#]</code> | yellow color highlight |
| <code>[-gray] gray color highlight [#]</code> | gray color highlight |
| Link | |
| <code>[-- "link url" --] [- title -]</code> | <u>title</u> |

4.2 Writing mathematical notations in the HTML log

The ability to write mathematical notations can be essential when the **weaver** package is used to develop educational materials for students or to develop technical documents related to new statistical commands. Using the MathJax (Cervone 2012; Cervone et al. 2014; D'Ambrosia and Spitznagel 2013) software, **weaver** allows rendering \LaTeX mathematical notations in a paragraph or a separate line within the HTML log and, consequently, in the PDF document rendered from the HTML log.

In addition to displaying text, the **txt** command can also be used for printing mathematical notations in the log. Simply place the mathematical notations between $\backslash(\dots \backslash)$ or $\backslash[\dots \backslash]$ signs, which render the notations in a text paragraph or on a separate line, respectively. Moreover, a single or double dollar sign can also be applied in the \LaTeX log. Examples of these notations are provided in the example in section 8.

5 Examples

The previous examples demonstrated how to communicate Stata commands and output to the **weaver** log, write and style text using **weaver** markup, write mathematical notations, add graphs, and create dynamic tables. The following examples briefly review several features in both the HTML and the \LaTeX log files.

5.1 HTML example

In this example, a summary of the **weaver** commands is provided with more styling details to demonstrate how **weaver** can be used to produce appealing educational materials, without any knowledge of HTML language. To make the dynamic report look more formal, **style(minimal)** is specified. To keep the document short, the title page is omitted by excluding the **title()**, **author()**, **affiliation()**, **address()**, **summary()**, and **date()** options from the **weave using filename** command.

► Example

```
weave using final.html, replace style(minimal) install

sysuse auto, clear
summarize mpg
scalar mpgN    = r(N)
scalar mpgMean = r(mean)
scalar mpgSD   = r(sd)

txt "-- Basic linear regression --"                                     ///
"[mono]auto.dta[#] includes the [mono]mpg[#] and [mono]weight[#] "      ///
"variables, which include data about mileage and weight of " r(N)        ///
" observations. The variables are summarized in table 1."

summarize weight
```

```

tbl ({1}"Variable", "Observations", {1}"Mean", {1}"SD" \           ///
"[mono]mpg[#]", mpgN, mpgMean, mpgSD \                             ///
"[mono]weight[#]", r(N), r(mean), r(sd)),                           ///
title("Table 1. Summary of the [mono]mpg[#] and [mono]weight[#] variables") ///
center

correlate mpg weight

txt "There is a high and negative correlation of " r(rho)           ///
" between the [mono]mpg[#] and [mono]weight[#] variables. We can impose " ///
"a regression line on the plotted data to examine the quality of fit " ///
"of a classic linear model. To do so, type the following command in " ///
"Stata. Figure 1 shows the fit regression line in the scatterplot."

div code graph twoway (scatter mpg weight) (lfit mpg weight)
graph export scatter.png, replace
img using scatter.png, center w(250) h(180)                       ///
title("Figure 1. Regression line fit on the [mono]mpg[#] and"      ///
"[mono]weight[#] variables")

txt The fit regression line can be modeled as follows:           ///
\[ \hat{\mu}(x) = \beta_{_1} x + \beta_{_0} \]

txt The \(\hat{\mu}(x)\) indicates the expected car mileage      ///
([mono]mpg[#]) for a given weight of \(\ x\) (lbs). Using the    ///
[mono]mpg[#] and [mono]weight[#] variables, the model can be updated ///
as follows:                                                      ///
\[ \texttt{mpg} = \beta_{_1}\texttt{weight} + \beta_{_0} \]

txt We can fit a regression model in Stata by using the          ///
[mono][blue]regress[#][#] command as shown below. Table 2 presents ///
a summary of the regression analysis.

div c regress mpg weight
mat reg = r(table)

scalar Wcof = reg[1,1]
scalar Wsr = reg[2,1]
scalar Wp = reg[4,1]
scalar Wll = reg[5,1]
scalar Wul = reg[6,1]
scalar Ccof = reg[1,2]
scalar Csr = reg[2,2]
scalar Cp = reg[4,2]
scalar Cl1 = reg[5,2]
scalar Cul = reg[6,2]

tbl ("Predictor", {1}"(\beta)", {1}"SE",                          ///
{1}{col 2} "95% Conf. Interval", {1}"p-value" \                 ///
"Weight", Wcof, Wsr, Wll, Wul, %7.3f Wp \                       ///
"Intercept", Ccof, Csr, Cl1, Cul, %7.3f Cp ), center           ///
title("Table 2. Summary of the regression analysis")

```

```

txt "The two-sided p-value was found to be less than " %7.3f Wp      ///
". Based on the estimated coefficients, the fit regression "         ///
"equation for estimating the [mono]mpg[#] based on given "          ///
"[mono]weight[#] would be "                                         ///
"\[ \texttt{mpg} = " %7.3f Wcof "\texttt{weight} + " Ccof " \]"
weave close

```

◀

5.2 L^AT_EX example

Here, the previous example is repeated for L^AT_EX to allow comparison of L^AT_EX markup with the more simplified `weaver` markup. Because the output of this example is similar to the previous example, it is omitted from the article.

► Example

```

weave using example.tex, install replace margin(5,20,5,20)

sysuse auto, clear
summarize mpg
scalar mpgN = r(N)
scalar mpgMean = r(mean)
scalar mpgSD = r(sd)

txt "\section{Basic linear regression}"

txt "\texttt{auto.dta} includes the \texttt{mpg} and "           ///
"\texttt{weight} variables, which include data about mileage and" ///
"weight of " r(N) " observations. The variables are summarized in table 1."

summarize weight

tbl ({1}"Variable", "Observations", {1}"Mean", {1}"SD" \      ///
"mpg", mpgN, mpgMean, mpgSD \                                  ///
"weight", r(N), r(mean), r(sd)),                               ///
title("Summary of the \texttt{mpg} and \texttt{weight} variables") center

correlate mpg weight

txt "There is a high and negative correlation of " r(rho)       ///
" between the \texttt{mpg} and \texttt{weight} variables. "    ///
"We can impose a regression line on the plotted data to examine the " ///
"quality of fit of a classic linear model. To do so, type the following " ///
"command in Stata. Figure 1 shows the fit regression "         ///
"line in the scatterplot."

div code graph twoway (scatter mpg weight) (lfit mpg weight)
graph export scatter.png, replace
img using scatter.png, center w(250) h(180)                   ///
title("Regression line fit on the \texttt{mpg} and \texttt{weight} variables")

```

```

txt The fit regression line can be modeled as follows:          ///

$$\widehat{\mu}(x) = \beta_1 x + \beta_0$$
          ///

txt "The  $\widehat{\mu}(x)$  indicates the expected car mileage "      ///
"for a given weight of  $x$  (lbs). Using the \texttt{mpg} "      ///
"and \texttt{weight} variables, the model can be updated as follows:"  ///

$$\text{mpg} = \beta_1 \text{weight} + \beta_0$$
          ///

txt "We can fit a regression model in Stata by using the \texttt{"      ///
"\color{blue}regress command as shown below. Table 2 presents "      ///
"a summary of the regression analysis."                          ///

div c regress mpg weight
mat reg = r(table)

scalar Wcof = reg[1,1]
scalar Wsr = reg[2,1]
scalar Wp = reg[4,1]
scalar Wll = reg[5,1]
scalar Wul = reg[6,1]
scalar Ccof = reg[1,2]
scalar Csr = reg[2,2]
scalar Cp = reg[4,2]
scalar Cl1 = reg[5,2]
scalar Cul = reg[6,2]

tbl ("Predictor", {1}"  $\beta$  ", {1}"SE",          ///
{1}{col 2} "95\% Conf. Interval", {1}"p-value" \      ///
"Weight", Wcof, Wsr, Wll, Wul, %7.3f Wp \           ///
"Intercept", Ccof, Csr, Cl1, Cul, %7.3f Cp ), center  ///
title("Summary of the regression analysis")

txt "The two-sided p-value was found to be less than " %7.3f Wp      ///
". Based on the estimated coefficients, the fit regression "          ///
"equation for estimating the \texttt{mpg} based on given "      ///
"\texttt{weight} would be "                                     ///

$$\text{mpg} = \text{\%7.3f Wcof} \text{\texttt{weight}} + \text{Ccof}$$
          ///

weave close

```

1 Basic linear regression

`auto.dta` includes the `mpg` and `weight` variables, which include data about mileage and weight of 74 observations. The variables are summarized in table 1.

Table 1: Summary of the `mpg` and `weight` variables

| Variable | Observations | Mean | SD |
|---------------------|--------------|---------|--------|
| <code>mpg</code> | 74 | 21.30 | 5.79 |
| <code>weight</code> | 74 | 3019.46 | 777.19 |

There is a high and negative correlation of -0.81 between the `mpg` and `weight` variables. We can impose a regression line on the plotted data to examine the quality of fit of a classic linear model. To do so, type the following command in Stata. Figure 1 shows the fit regression line in the scatterplot.

```
. graph twoway (scatter mpg weight) (lfit mpg weight)
```

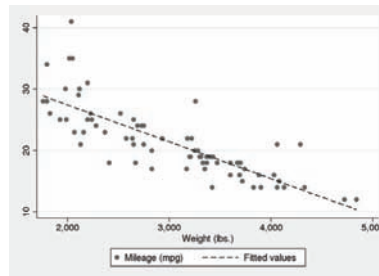


Figure 1: Regression line fit on `mpg` and `weight` variables

The fit regression line can be modeled as follows:

$$\hat{\mu}(x) = \beta_1 x + \beta_0$$

The $\hat{\mu}(x)$ indicates the expected car mileage for a given weight of x (lbs). Using the `mpg` and `weight` variables, the model can be updated as follows:

$$\text{mpg} = \beta_1 \text{weight} + \beta_0$$

We can fit a regression model in Stata by using the `regress` command as shown below. Table 2 presents a summary of the regression analysis.

```
. regress mpg weight
```

Table 2: Summary of the regression analysis

| Predictor | β | SE | 95% Conf. Interval | | p-value |
|-----------|---------|------|--------------------|-------|---------|
| Weight | -0.01 | 0.00 | -0.01 | -0.00 | 0.000 |
| Intercept | 39.44 | 1.61 | 36.22 | 42.66 | 0.000 |

The two-sided p-value was found to be less than 0.000. Based on the estimated coefficients, the fit regression equation for estimating the `mpg` based on given `weight` would be

$$\text{mpg} = -0.006 \text{weight} + 39.44$$

Figure 8. Preview of the PDF example

□ Technical note

Note that the `tbl` command parses the rows with the backslash symbol. Therefore, to include in a dynamic table any \LaTeX notations that begin with a backslash, such as `\beta` or `95\%`, a double backslash must be used to avoid conflict with the parsing syntax (for example, `\\beta` and `95\\%`). □

6 Applications and conclusion

weaver is a dynamic document package for Stata that creates an HTML or \LaTeX log file and a PDF dynamic document. Similarly to a Stata log, the **weaver** log is updated interactively as the analysis commands are executed, and the dynamic document can be previewed in real time. It also provides interesting features that are lacking in a SMCL log because HTML and \LaTeX markups can render a variety of graphical features and provide numerous styling options. The **weaver** log can run parallel to the SMCL log but will not register any input or output autonomously. While all commands and output are registered in the SMCL log, the **weaver** log instead allows the user to select the content of interest for the documentation.

The main advantage of this approach to dynamic reports is that users do not have to create a separate script file for writing the report. When the user simply tells **weaver** what to include in the HTML or \LaTeX log file, the document can be generated interactively while the analysis is developed or based on the same code files that are used for the data analysis. Moreover, in contrast to literate-programming packages that require “weaving” the document from the source code, **weaver** catches output from Stata as it is executed and then updates both the SMCL and the **weaver** log. **weaver** also allows merging other files into the **weaver** log, and therefore, it can include outputs from all other user-written packages that export \LaTeX or HTML files or tables, such as the **markdoc** package. Thus, a time-consuming and long-term analysis can be written in separate files using **weaver** or alternative packages, and then can be put together dynamically in an HTML or \LaTeX log file.

The **weaver** package has many applications for Stata users. In addition to creating a dynamic report, which is the main purpose of the package, it can also be used to write tutorials for students. Anything from a mathematical equation to the way it can be applied or programmed in Stata can be documented using the **weaver** package, making it a useful tool for developing educational documents. It can also be used as an active learning tool for statistics students. For example, [Baumer et al. \(2014\)](#) argue that integrating reproducible analysis tools in teaching statistics, as early as introductory statistics courses, improves students’ ability to express statistical results.

weaver also has several limitations. The package is mainly developed with the Stata language and partially supports the Mata language. In other words, Mata commands cannot be directly included in the log using the `div` command unless they are executed with the `mata:` prefix, which is the usual way of calling a Mata function in Stata without entering the Mata environment. However, the **markdoc** literate-programming package supports the Mata environment; thus, HTML or \LaTeX output generated by **markdoc**

can be merged into the **weaver** log to include sections of data analysis carried out by Mata.

Creating a dynamic-analysis document requires executing additional text commands or programming preparations for creating dynamic tables (for example, saving estimations into local macros or scalars). This can potentially result in increasing the complexity of the code due to mixing two languages in one place. This is a problem with any literate-programming software (not just the **weaver** package) that creates a trilingual source code including a programming language, a human language for interpretation, and a markup language for formatting and typesetting the documentation. If the markup language is complex, such as L^AT_EX or HTML, then the documentation becomes less distinguishable from the computer language, resulting in a file that is harder for humans to read. **weaver** attempts to address this issue by creating the **weaver** markup language, an easy-to-read and easy-to write markup language for annotating the content of the dynamic document.

The **weaver** log is not necessarily limited to HTML or L^AT_EX. The package can be updated to support other markup languages such as markdown to further simplify working with **weaver** package. Such a feature would allow Stata to store the output and documentation in a markdown file—that is running in parallel to the SMCL log—and use the markdown log to generate a dynamic document in various formats, an approach used by **markdoc** package.

Nonetheless, no tool or software can guarantee the clarity and cleanness of the code file, because writing a dirty code file is essentially a human problem. But keep in mind that the aim of writing code for running the analysis and compiling the dynamic document simultaneously is not mainly sharing the HTML or PDF document. Instead, it is commenting the procedure, interpreting the results, and sharing the code file and data that compile the analysis and reproduce the dynamic document. Readability of the source code is essential: the document can be misleading by excluding part of the analysis, giving a false description of the analysis procedure, or falsely interpreting the results. Therefore, code should be written with great care about its appearance and simplicity to facilitate others' reading and comprehension of the code.

7 References

- Allaire, J. J., J. Cheng, Y. Xie, J. McPherson, W. Chang, J. Allen, H. Wickham, A. Atkins, and R. Hyndman. 2016. *rmarkdown: Dynamic Documents for R*. R package version 0.5. <https://cran.r-project.org/web/packages/rmarkdown/index.html>.
- Baumer, B., M. Cetinkaya-Rundel, A. Bray, L. Loi, and N. J. Horton. 2014. R markdown: Integrating a reproducible analysis tool into introductory statistics. *Technology Innovations in Statistics Education* 8(1): 1–29. <http://escholarship.org/uc/item/90b2f5xh>.
- Cervone, D. 2012. MathJax: A platform for mathematics on the web. *Notices of the AMS* 59: 312–316.

- Cervone, D., C. Stark, R. Miner, and P. Topping. 2014. MathJax documentation: Release 2.3. <https://media.readthedocs.org/pdf/mathjax/v2.3-latest/mathjax.pdf>.
- Cordes, D., and M. Brown. 1991. The literate-programming paradigm. *IEEE Computer* 24: 52–61.
- D'Ambrosia, B. K., and C. R. Spitznagel. 2013. MathJax: Seamless mathematics on the web. In *Proceedings of the 25th International Conference on Technology in Collegiate Mathematics*. Norfolk, VA: ICTCM.
- Haghighi, E. F. 2014a. MarkDoc: Literate programming in Stata. <http://www.haghighi.com/statistics/stata-blog/reproducible-research/markdoc.php>.
- . 2014b. synlight: Stata module to highlight syntax in SMCL and translate to HTML format. Statistical Software Components S457894, Department of Economics, Boston College. <https://ideas.repec.org/c/boc/bocode/s457894.html>.
- . 2015. Statax: JavaScript syntax highlighter for Stata. <http://www.haghighi.com/statax/statax.php>.
- . 2016a. github: Stata module for searching and installing Stata packages from GitHub. GitHub. <https://github.com/haghighi.github>.
- . 2016b. markdown: A Stata module to convert Markdown to SMCL language. <http://github.com/haghighi/markdown>.
- Jann, B. 2004. estout: Stata module to make regression tables. Statistical Software Components S439301, Department of Economics, Boston College. <https://ideas.repec.org/c/boc/bocode/s439301.html>.
- . 2016. Creating LaTeX documents from within Stata using texdoc. Working Paper 14, University of Bern Social Sciences. <http://repec.sowi.unibe.ch/files/wp14/jann-2015-texdoc.pdf>.
- Johnson, A. L., and B. C. Johnson. 1997. Literate programming using noweb. *Linux Journal* 42: 64–69.
- Knuth, D. E. 1983. The WEB system of structured documentation. Technical Report STAN-CS-83-980, Department of Computer Science, Stanford University. <http://infolab.stanford.edu/pub/ctr/reports/cs/tr/83/980/CS-TR-83-980.pdf>.
- . 1984. Literate programming. *Computer Journal* 27: 97–111.
- . 1992. *Literate Programming*. Stanford, CA: CSLI Lecture Notes.
- Leisch, F. 2002. Sweave: Dynamic generation of statistical reports using literate data analysis. In *COMPSTAT 2002*, ed. W. Härdle and B. Rönz, 575–580. Physica Verlag: Heidelberg.
- Lenth, R. V. 2012. *StatWeave Users' Manual*. <http://homepage.stat.uiowa.edu/~rlenth/StatWeave/StatWeave-manual.pdf>.

- Murdoch, D. J., and V. J. Carey. 2001. On the edge: Statistics & computing: Literate statistical programming: Concepts and tools. *Chance* 14: 46–50.
- Ramsey, N. 1994. Literate programming simplified. *IEEE Software* 11: 97–105.
- Ramsey, N., and C. Marceau. 1991. Literate programming on a team project. *Software—Practice and Experience* 21: 677–683.
- Ruys, T. C., and E. Brinksmas. 1998. Experience with literate programming in the modelling and validation of systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, ed. W. R. Cleaveland, 393–408. Stony Brook, NY: Springer.
- Sametinger, J., and G. Pomberger. 1992. A hypertext system for literate C++ programming. *Journal of Object-Oriented Programming* 4: 24–29.
- Shum, S., and C. Cook. 1994. Using literate programming to teach good programming practices. *ACM SIGCSE Bulletin* 26(1): 66–70.
- Watson, I. 2004. tabout: Stata module to export publication quality cross-tabulations. Statistical Software Components S447101, Department of Economics, Boston College. <https://ideas.repec.org/c/boc/bocode/s447101.html>.
- Xie, Y. 2014. *Dynamic Documents with R and knitr*. Boca Raton, FL: Chapman & Hall/CRC.
- . 2016. *knitr: A General-Purpose Package for Dynamic Report Generation in R*. R package version 1.13. <https://cran.r-project.org/web/packages/knitr/index.html>.

About the author

E. F. Haghish is a PhD student in applied statistics at the Center for Medical Biometry and Medical Informatics at the University of Freiburg in Germany. He is also affiliated with the University of Southern Denmark in Denmark.