The Stata Journal (2016)
16, Number 4, pp. 964–988

markdoc: Literate programming in Stata

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. Rigorous documentation of the analysis plan, procedure, and computer codes enhances the comprehensibility and transparency of data analysis. Documentation is particularly critical when the codes and data are meant to be publicly shared and examined by the scientific community to evaluate the analysis or adapt the results. The popular approach for documenting computer codes is known as literate programming, which requires preparing a trilingual script file that includes a programming language for running the data analysis, a human language for documentation, and a markup language for typesetting the document. In this article, I introduce `markdoc`, a software package for interactive literate programming and generating dynamic-analysis documents in Stata. `markdoc` recognizes Markdown, LaTeX, and HTML markup languages and can export documents in several formats, such as PDF, Microsoft Office `.docx`, OpenOffice and LibreOffice `.odt`, LaTeX, HTML, ePub, and Markdown.

Keywords: pr0064, markdoc, Markdown, HTML, LaTeX, literate programming, dynamic documents, reproducible research, log file, translator

1 Introduction

In recent years, there have been concerns about replicability of study findings as well as reproducibility of data analysis results in scientific publications. Replication requires reimplementing experiments to validate the findings. Reproducibility is when an independent researcher can replicate the data analysis or the computation using the same data, procedure, and methodology (Baggerly and Berry 2011). Reproducing the analysis is a minimum standard for evaluating the quantitative results because reproducibility does not necessarily certify quality, sound methodology, correctness of data collection, or validity of the findings (Peng 2011; Stodden, Leisch, and Peng 2014). However, it does provide partial transparency for other researchers to validate the analysis procedure and examine or adapt the claims (Gentleman and Lang 2007).

To support reproducibility of the analysis, the researcher should prepare detailed documentation of the methodology and analysis plan, data, and analytic codes and output. The documentation can be included in the analysis code as comments; however, reading comments requires navigating through script files that get more convoluted as the size and complexity of the program increases. Furthermore, comments are scattered across the script files and do not fulfill the demand for a coherent document. Alternatively, the documentation can be written using a word processor, such as Microsoft Word. While this approach might suffice for documenting the variables and the analysis plan, it would be boring, time consuming, and prone to human error to reference the code or output and to update the documentation after making a change in the code.

Another solution to this problem is to include the documentation within the script files along with a special notation that separates the documentation from the source code. Next, a software package is used to parse the special notations and then typeset the documentation or compile the code, procedures that are called "weave" and "tangle", respectively. This solution was proposed by Knuth (1984) and is called literate programming. Knuth (1983) developed the WEB literate-programming software, which provided the means to write structured documentation within the source code and to generate dynamic documentation.

Many have attempted to adapt the literate-programming paradigm for statistical data analysis (Leisch 2002; Rossini and Leisch 2003; Rossini 2001; Xie 2016, 2014; Lenth 2012). Yet the only software allowing literate programming for Stata is StatWeave (Lenth 2012), which is written in Java, and its Stata package alternative, `texdoc` (Jann 2016). The main drawback of StatWeave and `texdoc` is that they only support LaTeX, excluding all Stata users who are not familiar with this markup language. Although LaTeX is very enabling, it is a complex markup language and can hinder the readability of the source file (see section 5.2).

Another shortcoming of StatWeave and `texdoc` is that they do not support a real-time preview of the document; that is, they do not support writing and viewing the dynamic document interactively without reexecuting the analysis code. In practice, data analysis is developed gradually in a script file, and an ideal literate-programming software would allow interactive documentation and execution of the data analysis. Furthermore, StatWeave forces the user to create a new script file to generate the dynamic-analysis document, which leads to additional problems in updating and version control of the script files. For example, the source file that StatWeave executes cannot be executed directly in a do-file because the source file includes special notations that separate the LaTeX documentation from the analytic code. These notations are not meaningful for Stata. In addition, `texdoc` includes several commands to initialize and close the document, which makes working with it inconvenient, especially for those just learning statistics.

Literate programming in data analysis should be easy to implement in the source code, support a real-time preview of the document, provide some options to adjust the commands and output in the dynamic document, and above all, keep the source file simple and easy to read. Furthermore, the ability to generate dynamic documents in various formats from the same source code would be an advantage and would extend the applications of the literate-programming software. For example, some users might wish to produce HTML documentation that can be uploaded on websites, while others might wish for PDF or editable documentation formats, such as LaTeX, Microsoft Word `.docx`, or OpenOffice `.odt`. In this article, I introduce the `markdoc` package, which provides a convenient solution for literate programming in Stata, and discuss its potential applications.

2 Workflow

Because the log file registers every entry in Stata, including the markup annotations and the text required to typeset the dynamic document, the `markdoc` command was designed to produce the dynamic document as a by-product of the log file. In other words, `markdoc` interprets the log file—which is updated in real-time during the analysis session—and typesets the dynamic-analysis document based on the content of the log file. Once the user opens a Stata Markup and Control Language (SMCL) log file and begins the data analysis, `markdoc` can quickly and interactively compile a dynamic document, as shown in figure 1.

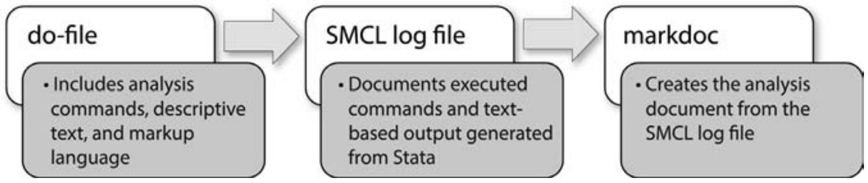

do-file
• Includes analysis commands, descriptive text, and markup language
SMCL log file
• Documents executed commands and text-based output generated from Stata
markdoc
• Creates the analysis document from the SMCL log file

Figure 1. The process of producing dynamic documents with `markdoc`

This workflow has several advantages when compared with other common literate-programming packages used for data analysis. First, the log file can be used to compile a dynamic document in various formats without rerunning the analysis code. Second, dynamic documents can be produced quickly without reexecuting the analysis code because the log file contains the history of commands and output and is updated automatically. Finally, this workflow eliminates the need for creating an additional script file to produce the dynamic document because it follows the natural data analysis practice in Stata without introducing numerous commands for creating the analysis document.

Although using a single command to convert a SMCL log file into various document formats is convenient, it does not ensure the reproducibility of the source code, even if the do-file begins with opening a log and ends by closing the log. For example, the user might have made changes to the data that are not included in the do-file. Therefore, the do-file must be examined in a clean workspace, where no data are loaded in Stata.

`markdoc` supports both procedures. If a SMCL log file is given, it converts the log to a document without evaluating the reproducibility of the script file that generated the log. In the example below, `markdoc` takes the SMCL log file as input and generates a PDF document. Opening the log `quietly` avoids including the log description in the document. The `quietly log close` command (which can be abbreviated as `qui log c`) is automatically removed by `markdoc` to keep the document clean.

▷ **Example**

```
quietly log using example, replace smcl
display "Hello world"
quietly log close
markdoc example.smcl, export(pdf)
```

◁

In addition, `markdoc` can also take a do-file as an input and actively execute it in a new workspace—where no data is loaded in that workspace—to examine its reproducibility and generate a dynamic document from the source code[1] as shown in figure 2.

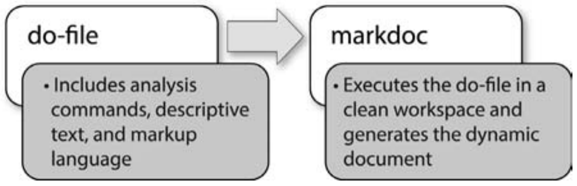

do-file
• Includes analysis commands, descriptive text, and markup language
markdoc
• Executes the do-file in a clean workspace and generates the dynamic document

Figure 2. The process of producing dynamic documents from do-files

In the example above, if the code is saved as a do-file—for example, in a file named `example.do`—then `markdoc` can produce an identical document using the script file as shown in the example below.

▷ **Example**

```
markdoc example.do, export(html) master
```

◁

❏ **Technical note**

HTML and LaTeX documents have a strict layout format to represent the documentation properly. Literate-programming software such as `texdoc` and StatWeave require the user to create the layout manually, which reduces the readability of the source code. The `master` option in `markdoc` automatically produces a standard HTML or LaTeX layout to keep the source code as clean and simple as possible.

❏

3 The markdoc package

3.1 Features

`markdoc` recognizes Markdown, HTML, and LaTeX markup languages and can export the dynamic document into several formats: PDF, Microsoft Office `.docx`, Open Office

1. `markdoc` can take a Mata file or an ado-file as input and generate Stata help files (`.sthlp`) or a package vignette from Markdown documentation. This feature is not discussed in this article.

and LibreOffice `.odt`, LaTeX, HTML, ePub, and Markdown,[2] as shown in figure 3. By applying any of these markup languages, users can style the dynamic document, insert figures, create dynamic tables, write dynamic text, and style the text.

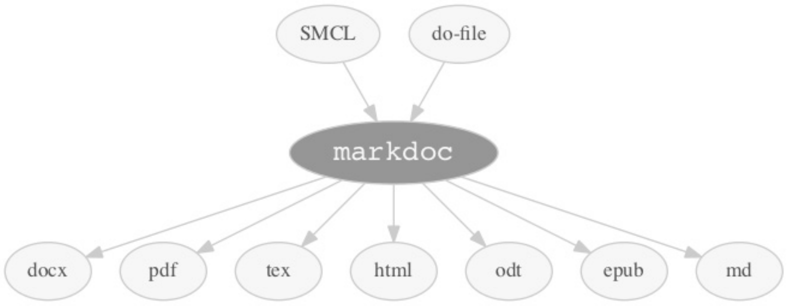


Figure 3. Document formats supported by `markdoc`

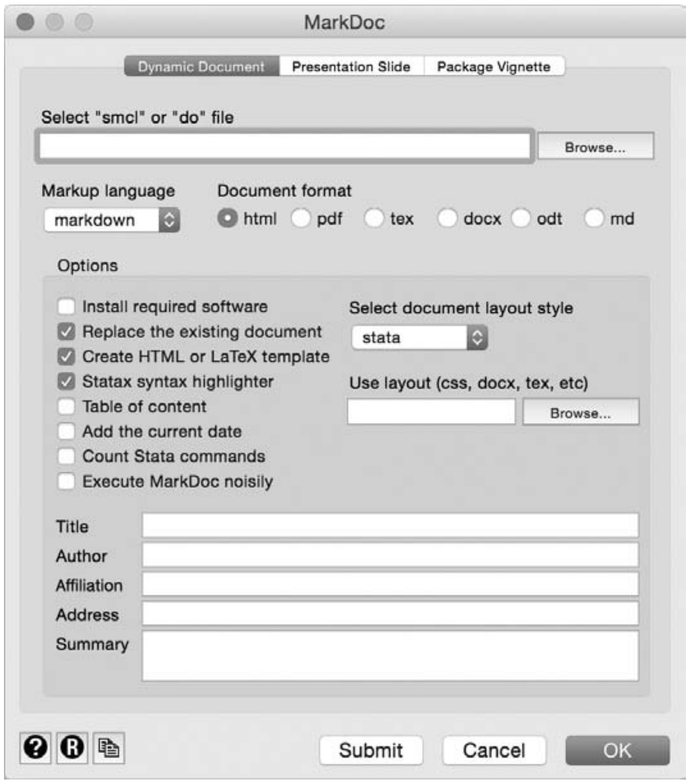
In addition, `markdoc` includes several document styles, and it also allows users to create a dynamic document by using external template files, for example, a LaTeX header file, a CSS file, or a Microsoft Word template file. Furthermore, `markdoc` uses a syntax highlighter for Stata commands, which makes the code more appealing, distinguishable from the documentation text, and easier to read and comprehend. Finally, the package allows rendering LaTeX mathematical notations when exporting not only to a LaTeX document, but also to HTML, PDF, Microsoft Word, and OpenOffice documents.

3.2 Dialog box

`markdoc` was designed to be a user-friendly package—consisting of one command—that can export a variety of documents from the same source code. Particularly, it was intended to be used in classrooms for teaching statistics. Students can use the package to document their analysis using Markdown, include mathematical notations, and actively practice data analysis and interpretation. To further facilitate learning `markdoc`, a dialog box was programmed to visualize the main options and functionalities of the package. To launch the dialog box, type

2. `markdoc` can also produce PDF and HTML dynamic presentation slides. This feature goes beyond the purpose of this article and thus is not discussed here.

```
. db markdoc
```



Select "smcl" or "do" file

Markup language: markdown

Document format: [✓ html] pdf tex docx odt md

Options

- [ ] Install required software
- [✓] Replace the existing document
- [✓] Create HTML or LaTeX template
- [✓] Statax syntax highlighter
- [ ] Table of content
- [ ] Add the current date
- [ ] Count Stata commands
- [ ] Execute MarkDoc noisily

Select document layout style: stata

Use layout (css, docx, tex, etc)

Title

Author

Affiliation

Address

Summary

Figure 4. `markdoc` dialog box

The dialog box includes three tabs, each specializing in a particular document format, that are independent of one another (see figure 4). In this article, I only focus on the *Dynamic Document* tab, which is used for generating dynamic-analysis reports in several formats. The other tabs are used for generating dynamic PDF and HTML presentation slides and Stata help files and package vignettes from Markdown. The documentation for the additional functionalities can be found on GitHub Wiki, that is, http://github.com/haghish/markdoc/wiki.

In the *Dynamic Document* tab, you can select a do-file or SMCL file and select the format in which the dynamic document will be executed. The dialog box provides all the options supported by `markdoc`. You can specify the markup language that you use for writing the documentation, select the layout style of the document, or use your own layout template. Moreover, you can turn on the syntax highlighter or create a table of contents with a mouse click. The dialog box helps you to become familiar with the package and examine all of its features.

3.3 Installation

markdoc requires the weaver, statax, and markdown packages. weaver (Haghish 2014b) is a dynamic-document generator, which creates LaTeX, HTML, and PDF dynamic documents in Stata. It also supports markdoc with commands to create dynamic tables, write dynamic text, and insert the current graph from Stata into the dynamic document. statax (Haghish 2015) is a syntax highlighter engine that highlights Stata commands in HTML, PDF, and LaTeX documents. The markdown (Haghish 2016b) command is a Markdown-to-SMCL translator that markdoc uses to generate Stata help files from Markdown (Gruber 2004). The mentioned packages are hosted on http://www.github.com. The github[3] (Haghish 2016a) command can install markdoc and its dependencies as shown below.

```
. github install haghish/markdoc
```

In addition, markdoc requires the third-party software Pandoc (MacFarlane 2006), for converting Markdown to other file formats, and wkhtmltopdf (Kulkarni and Truelsen 2015), for creating PDF documents from source written with Markdown or HTML. Users who wish to write with LaTeX can also compile a PDF document with the markdoc command if a LaTeX distribution is installed on your machine. All mentioned software are open-source freeware, available for Microsoft Windows, Mac, and Linux operating systems. The packages hosted on the Statistical Software Components server only include the ado-files and help files. markdoc provides optional automatic installation for Pandoc and wkhtmltopdf, which might be more convenient for many users than downloading and installing the third-party software manually. The manual and automatic installation procedures are both described below.

Manual installation of third-party software

The Pandoc software can be downloaded from http://pandoc.org and installed manually. Once Pandoc is installed, the path to the executable Pandoc should be provided to markdoc by using the pandoc(*str*) option. The wkhtmltopdf software can be downloaded from http://wkhtmltopdf.org and installed anywhere on your machine. The path to the executable wkhtmltopdf file should be provided to markdoc by using the printer(*str*) option. Similarly, for compiling LaTeX to PDF, the proper LaTeX distribution should be downloaded from https://latex-project.org and the path to the executable pdfLaTeX compiler should be provided using the printer(*str*) option (see the master option in section 4 in this regard).

The paths to Pandoc, wkhtmltopdf, and pdfLaTeX can be permanently defined using the weave setup command. This command opens a script file that memorizes the path to each software package within a particular global macro. The file includes instructions and examples of how the file paths should be defined.

3. To install the github command, type
net install github, from("https://haghish.github.io/github/").

Automatic installation of third-party software

The `markdoc` command includes the `install` option, which downloads the Pandoc and wkhtmltopdf software automatically if they are not already installed or cannot be accessed by `markdoc`. The automatic installation was successfully tested on Mac OS X (10.9 and 10.10); 32-bit and 64-bit versions of Microsoft Windows (XP, 7, and 8); Microsoft Windows 10 (64-bit); and Linux Ubuntu 14.04 (64-bit), Mint 17 (32-bit and 64-bit), and CentOS 7 (64-bit). However, manual installation is generally recommended because it ensures the installation of the latest version of the software.

❏ Technical note

`markdoc` installs the required software in a directory named `Weaver` inside the `plus` directory, where Stata expects to find user-written ado-files. The path to the `\ado\plus\` directory can be found using the `sysdir` command, which lists Stata's system directories. For Stata 13 and 14, the default `Weaver` directory paths are shown below based on the operating system.

Windows: `C:\ado\plus\Weaver`

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

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

❏

3.4 Testing markdoc

After installing the required packages, `markdoc` can be tested to ensure that the software is running properly, as shown below. (Note that when the `test` option is used, there is no need to specify the SMCL log filename.) If the software was installed manually, the `pandoc()` and `printer()` options should be specified to tell `markdoc` where it can access Pandoc and wkhtmltopdf. If the software was installed automatically, then only the `test` option is needed to carry out the test.

```
markdoc, test pandoc("path/pandoc") printer("path/printer")
```

4 Syntax

The `markdoc` command only requires the name of the SMCL log file or a do-file to produce the dynamic document, as shown below. If the file suffix (`.smcl` or `.do`) is not specified, then SMCL log is assumed.

`markdoc` *filename* [, `pandoc(`*str*`)` `printer(`*str*`)` `install` `test` `replace`
`export(`*name*`)` `markup(`*name*`)` `numbered` `style(`*name*`)` `toc` `title(`*str*`)`
`author(`*str*`)` `affiliation(`*str*`)` `address(`*str*`)` `summary(`*str*`)` `date` `master`
`statax` `template(`*filename*`)` `noisily`]

Options

`pandoc(`*str*`)` specifies the path to the executable Pandoc on your machine. This option is only required if Pandoc is installed manually and the path is not permanently defined using the `weave setup` command.

`printer(`*str*`)` specifies the path to the executable wkhtmltopdf or pdfLaTeX software on your machine. wkhtmltopdf generates a PDF document from the Markdown and HTML markup languages, and pdfLaTeX typesets LaTeX to a PDF document.

`install` downloads the required third-party software automatically if they are not already installed or accessible (see section 3.3).

`test` runs an example do-file and generates HTML and PDF dynamic documents to ensure that the required software is running properly.

`replace` rewrites the exported document if it already exists.

`export(`*name*`)` specifies the format of the exported document. The supported document formats are `html`, `pdf`, `epub`, `tex` (LaTeX), `docx` (Microsoft Office), and `odt` (OpenOffice and LibreOffice). If this option and the `markup(`*name*`)` option are not specified, then `markdoc` exports a Markdown (`export(md)`) file by default.

`markup(`*name*`)` specifies the markup language used to annotate the document. *name* can be `markdown` (the default), `html`, or `latex`.

`numbered` turns on numbering of the commands in the dynamic document.

`style(`*name*`)` specifies the theme of the HTML, LaTeX, Microsoft Word `.docx`, and PDF documents. *name* can be `simple` or `stata`. `style(stata)` can be used to export LaTeX documents in *Stata Journal* style, even if the document is written with Markdown.

`toc` automatically creates a table of contents in the PDF, LaTeX, and Microsoft Word dynamic documents.

`title(`*str*`)` displays the title of the dynamic document on the title page.

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

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

`address(`*str*`)` displays the author's contact information (or any preferred relevant information) on the title page. For example, it can be used to add a telephone number, email address, or mailing address.

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

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

`master` creates a layout for HTML and LaTeX documents. For example, it includes MathJax and CSS codes in HTML documents and includes the most common packages for rendering graphs and figures in LaTeX documents. Both HTML and LaTeX documents have a restricted layout. `master` automates this process and creates a complete layout for HTML and LaTeX documents, allowing the user to focus on the content of the document and keep the source code clean. Users who are not familiar with HTML or LaTeX will find this option handy.

`statax` uses the `statax` package to highlight the syntax of Stata codes in the HTML and PDF documents.

`template(`*filename*`)` applies an external style sheet. For example, if the document is written in Markdown or HTML and exported to HTML or PDF, then a CSS style sheet can be specified to alter the appearance of the dynamic document. Similarly, when the document is exported to Microsoft Word `.docx`, a reference document with a similar format can be used to alter the style of the Word document (for example, create a `.docx` document, change the styles and themes, and use it as a template file). Finally, if the document is written in LaTeX, this option can be used to append the LaTeX header (that is, required packages, user-defined commands, etc.) to the exported document.

`noisily` displays the extended log.

4.1 Version declaration

As noted, the `github install haghish/markdoc` command installs the latest version of the `markdoc` package and its dependencies. However, `markdoc`—like any other software—may change over time, making older projects that were documented using `markdoc` irreproducible. To tackle this problem, all releases of `markdoc` are made available via GitHub and can be installed via the `github` (Haghish 2016a) command. To ensure that your current projects remain executable over time, you should document the version of `markdoc` that you are using to write the analysis documents. For example, if you are using `markdoc` version 3.9.4, write the following note in your analysis codes:

```
/***
Version declaration
-------------------

This analysis was developed using Stata 14.2 and markdoc 3.9.4
on Mac OS X 10.10.5. To install markdoc 3.9.4 and the dependencies
required by this version, type

    github install haghish/markdoc, version(3.9.4)
***/
```

The `version()` option not only installs the specified version of `markdoc`, but also `markdoc`'s dependencies for version 3.9.4, ensuring that you install the same version of `markdoc` and its dependencies that was used to produce a dynamic-analysis report. Naturally, reinstalling the `markdoc` package without specifying the `version()` option will update the package and its dependencies to the current version. A similar declaration is recommended in scientific publication, by specifying the version of Stata and `markdoc` used for executing and documenting the data analysis.

5 Supported markup languages

To automatically typeset a document from source files, a markup language is needed. A markup language is a computer language that annotates the content and styles in the document. Markup languages can be divided into three categories: descriptive, procedural, and presentational (Reid 2015). Descriptive languages, such as XML, describe the content of the document. Procedural languages, such as PostScript, define how the document should be processed. And presentational languages, such as HTML and LaTeX, frame how the document should be rendered and presented. To write a literate program, we need a presentational markup language to define the template of the document and render headings and images in the document. `markdoc` supports the Markdown, HTML, and LaTeX markup languages. These markup languages are briefly compared with examples in the following sections.

Regardless of the markup language used to write a dynamic-analysis document, there is a demand for a special notation to separate the documentation text from the analytic code and results. As shown in the examples of the following sections, `markdoc` requires the markup syntax and the documentation text to be written as a comment in the do-file, starting with `/***` and ending with `***/`, each on a separate line. This notation allows the do-file to remain executable by keeping the comments within from interfering with the code execution in Stata.

5.1 Markdown

Markdown (Gruber 2004) is a minimalistic markup language and has an intuitive syntax that makes it preferable to HTML and LaTeX. Table 1 presents the most common Markdown syntax.[4]

4. Complete documentation of the Markdown commands can be found at http://daringfireball.net/projects/markdown/dingus.

Table 1. Markdown syntax references

Markdown syntax	Result
Heading 1	**Heading 1**
=========	
Heading 2	**Heading 2**

###Heading 3	**Heading 3**
####Heading 4	**Heading 4**
plain text paragraph	plain text paragraph
> text	block quote
Bold or __Bold__ text	**Bold** or **Bold** text
Italic or _Italic_ text	*Italic* or *Italic* text
`monospace` text	monospace text
superscript^2^	superscript^2
---	horizontal rule
1. Ordered item1	1. Ordered item1
A. Sublist 1	A. Sublist 1
a. Subsublist 1	a. Subsublist 1
2. Ordered item2	2. Ordered item2
* Unordered item1	● Unordered item1
* Sublist 1	- Sublist 1
* Subsublist 1	* Subsublist 1
* Unordered item2	● Unordered item2
![Text](*filename*)	Insert an image with description
[Text](http://url)	Insert a hyperlink

Dynamic documents written in Markdown can include LaTeX mathematical notations (see section 5.3). They can also be exported to HTML, LaTeX, or any of the other supported document formats, thereby providing greater flexibility compared with HTML and LaTeX. The example below demonstrates using Markdown to create headings and subheadings and to style text in a do-file. The do-file is then compiled into a Microsoft Word document.

▷ **Example**

```
quietly log using example, replace
/***
This is a heading
=================

This is a subheading
--------------------

Text can also appear as _Italic_ or __Bold__.
***/
quietly log close
markdoc example, export(docx)
```

◁

❏ **Technical note**

The default markup language is Markdown. If the document is annotated with HTML or LaTeX, then the markup language should be specified using the `markup()` option.

❏

5.2 HTML and LaTeX

Although it is simple, convenient, and flexible, Markdown only provides commands for basic styling, such as writing headings, making text bold or italic, adding a hyperlink, and inserting a graph or image. In contrast, HTML and LaTeX provide more options for styling and annotating the dynamic document; however, dynamic documents written in these markup languages will not be as readable as documents written with Markdown. While HTML and LaTeX are supported by `markdoc`, the reader is encouraged to practice Markdown documentation whenever the document does not require detailed styling. The following example creates an HTML document using HTML markup.

▷ **Example**

```
quietly log using example, replace
/***
<h1>This is a heading </h1>
<h2>This is a subheading </h2>
<p>Text can also appear as <i>Italic</i> or <b>Bold</b>.</p>
***/
quietly log close
markdoc example, export(html) markup(html)
```

◁

Now, the previous example is repeated using LaTeX markup and with the addition of the `master` option to create the document's layout automatically and allow compilation of the document. Because we are compiling LaTeX to PDF, the path to pdfLaTeX should also be specified in the `printer` option. To demonstrate the alternative procedure to produce dynamic documents with `markdoc`, assume that the following code is saved in a do-file named `example.do`. Note that the `statax` option was also added to highlight the syntax of Stata commands in the LaTeX document.

▷ **Example**

```
/***
\section{This is a heading}
\subsection{This is a subheading}
Text can also appear as \textit{Italic} or \textbf{Bold}.
***/
display "writing with LaTeX increases the complexity of the code"
```

Then the following command will generate the dynamic document:

```
markdoc example.do, master export(pdf) markup(latex) statax       ///
        printer("/path/to/pdflatex")
```

◁

5.3 Mathematical notations

`markdoc` supports LaTeX markup language, and naturally, it can also render LaTeX notations. In addition, when the document is written in Markdown, `markdoc` can render LaTeX notation in all the supported formats. To write an inline notation, the notation should begin and end with a single dollar sign. To place the notation on a separate line, the notation should begin and end with double dollar signs.

▷ **Example**

```
quietly log using example, replace
/***
Writing mathematical notations
==============================

A text paragraph can include mathematical notations. For example, the
formula $Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$ will be displayed within
the text paragraph, whereas this next formula will be placed on a separate
line: $$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$$
***/
quietly log close
markdoc example, export(docx)
```

◁

6 Additional notation markers

The `markdoc` package can include a subset of the SMCL log file's code and output to create less-detailed documentation or to concentrate on the main result of the analysis. For example, the user may not want to include the process of data preparation and exploration in the dynamic document, because all the executed commands and output are already documented in the log file. However, turning the SMCL log on and off is not a favorable approach because it reduces the transparency of the analysis, especially if the data analysis is carried out interactively. Instead, `markdoc` includes a list of notation markers that allow the user to exclude a command, output, or part of the log file from the dynamic document. These markers are listed in table 2.

Table 2. Additional notation markers

Marker	Description
`/**/`	exclude the Stata command but keep the output
`/***/`	exclude the Stata output but include the command
`//OFF`	exclude everything in the log that follows
`//ON`	deactivate the `//OFF` marker
`<!*!>`	interprets macros and scalars in documentation
`//IMPORT` *filename*	include an external file (Markdown, HTML, LaTeX)

To exclude the Stata command or output, the code line should begin with `/**/` or `/***/` markers, respectively. To exclude a section of code and output from the SMCL log file, place the `//OFF` and `//ON` markers on a separate line in the do-file; `markdoc` will ignore anything that appears between these markers. The `<!*!>` marker allows writing dynamic text and is discussed in section 7.

Several user-written packages export Stata output to LaTeX and HTML files, including `tabout` (Watson 2004), `weaver` (Haghish 2014b), and `synlight` (Haghish 2014a). `markdoc` provides the `//IMPORT` *filename* command to include external files in the dynamic document. For example, using the `outreg2` package (Wada 2005), a regression table can be exported to a LaTeX file and then imported into the dynamic document. To keep the source code clean, the `//IMPORT` command can also be used to include a descriptive text file, such as a project description or methodology, that is not supposed to change based on the analysis results.

7 Writing dynamic documentation

`markdoc` borrows three additional commands from the `weaver` package (Haghish 2014b): `txt`, `tbl`, and `img`, which are used to write dynamic text, to create dynamic tables, and to capture and include graphs in the dynamic document, respectively. These commands are discussed with examples in this section.

7.1 Dynamic text

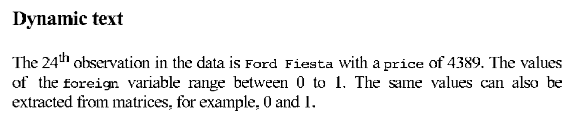
Dynamic text—that is, including macros and scalars within text—is a useful feature in literate programming. When the data are changed and the code reexecuted, dynamic text will automatically update the values mentioned in the text. `markdoc` allows including macros and scalars inside the documentation text using `<!` **`!>` marker. The ** can be a numeric or string macro or a scalar as well as a single scalar subsetted from a matrix or a dataset. Furthermore, Stata's *display_directive* (see [P] **display**) can be used to format the value of ***. This marker requires an active execution, that is, when `markdoc` is given a do-file to produce the dynamic document (see figure 5):

Example

```
//OFF
sysuse auto, clear
summarize foreign
scalar c = "Dynamic"
local min = `r(min)´
scalar max = r(max)
matrix Mat = (`r(min)´, `r(max)´)
//ON
/***
###<!c!> text
The 24^th^ observation in the data is `<!make[24]!>` with a `price` of
<!price[24]!>. The values of the `foreign` variable range between
<!`min´!> to <!max!>. The same values can also be extracted from matrices,
for example, <!Mat[1,1]!> and <!Mat[1,2]!>.
***/
```

If the example is saved in `example.do`, the document can be produced by typing the following:

```
. markdoc example.do, export(docx)
```



Dynamic text

The 24th observation in the data is `Ford Fiesta` with a `price` of 4389. The values of the `foreign` variable range between 0 to 1. The same values can also be extracted from matrices, for example, 0 and 1.

Figure 5. A dynamic text example

◁

Alternatively, to provide the ability to write dynamic text within loops and user-written program, the `txt` command—which is used for a similar purpose in the `weaver` package (Haghish 2014b)—is borrowed and updated to support `markdoc`. The syntax of the `txt` command is as follows:

`txt` [`code`] [*display_directive* [*display_directive* [...]]]

display_directive (similar to the *display_directive* of Stata's `display` command) can be a double-quoted string, a compound double-quoted string, a mathematical operation, a scalar, or a formatting directive (*%fmt*). By default, the `txt` command creates a text paragraph. However, if the `code` subcommand is added, it produces a monospace-font text line that can be used to discuss a line of code in the analysis document. In the example below, the `r(N)` scalar, which is returned from the `summarize` command, is used to print the number of observations in the `price` variable in a dynamic-text paragraph.

▷ Example

```
quietly log using example, replace
sysuse auto, clear
quietly summarize price
txt r(N) " observations are included in the dataset."
quietly log close
markdoc example, export(odt)
```

◁

❏ Technical note

Using the `<!*!>` marker is convenient and follows `markdoc`'s usual procedure for writing the documentation. However, this marker can only be used when the dynamic document is actively executed by `markdoc`, that is, when the do-file is given to `markdoc`. Therefore, to generate a dynamic document from the SMCL log file, the dynamic text should be written using the `txt` command to display the markup code in the SMCL log. Moreover, the `<!*!>` marker cannot be used in loops or within ado-files. For executing markup and documentation within a loop or user-written program, the `txt` command should be applied.

❏

7.2 Dynamic tables

The `tbl` command is also borrowed from the `weaver` package (Haghish 2014b) to create simple Markdown tables that can be exported to any of the supported document formats. This command creates a Markdown table and thus should only be used if the document is written in Markdown. The syntax of `tbl` is similar to that of Stata's `matrix input` command, creating the table by defining subsequent rows.

```
tbl (*[,*...] [\*[,*...] [\[...]]]) [, title(str)]
```

The asterisk symbol represents a display directive, which can be a double-quoted string, compound double-quoted string, mathematical operation, scalar, or formatting directive (*%fmt*). Using `auto.dta`, I demonstrate the creation of a simple table that includes the number of observations, mean, and standard deviation of the `price` variable; see figure 6.

Example

```
quietly summarize price
tbl ("Variable", "Observations", "Mean", "SD" \  ///
     "__price__", r(N), %9.2f r(mean), %9.2f r(sd))
```

Variable	Observations	Mean	SD
price	74	6165.26	2949.50

Figure 6. A dynamic table

◁

7.3 Dynamic figures

Markdown, HTML, and LaTeX have specific syntaxes to include figures in the document. To automate the process of including a Stata graph in the document, the graph should first be exported to an image file and then included in the document. The example below demonstrates how to include a figure in a Microsoft Word document.

Example

```
quietly log using example, replace
sysuse auto, clear
histogram price
quietly graph export graph.png, width(300) replace
/***
Adding a figure in the document
===============================

![Histogram of the `price` variable](./graph.png)
***/
quietly log close
markdoc example, export(docx)
```

◁

To include figures in the analysis document, they should be exported to any of the common graphical formats: `.png`, `.gif`, `.jpeg`, etc. The `.png` format is recommended because it is a lossless and popular graphical format that can be included in any of the supported document formats and is also supported by Stata.

Alternatively, we can again borrow from the `weaver` package. This time, we borrow the `img` command, which has the following syntax:

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

The `img` command can be used in two different ways. It can include an image file that is already stored on your machine: `img using` *filename*. Or, it can be used without `using` *filename*, which will automatically capture the current graph from Stata, store it in a directory named `Weaver-figure` in the working directory, and import it into the dynamic document.

▷ **Example**

```
quietly log using example, replace
/***
Using the `img` command
----------------------

***/
sysuse auto, clear
histogram price
img
quietly log close
markdoc example, export(pdf) statax
```

◁

By default, the `img` command prints Markdown code in the log; the `markup()` option can be used to define `html` or `latex` markup. The `width()` and `height()` options, which are used to specify the figure size in the dynamic document, only work in HTML and LaTeX because Markdown cannot resize images.

8 Example and notes

In the following example, I use Markdown syntax to write and style text, to insert a graph, and to export the document in PDF format (see figure 7). The example also includes the `tbl` command for creating a dynamic table as well as the notation markers for hiding parts of the log file in the dynamic document.

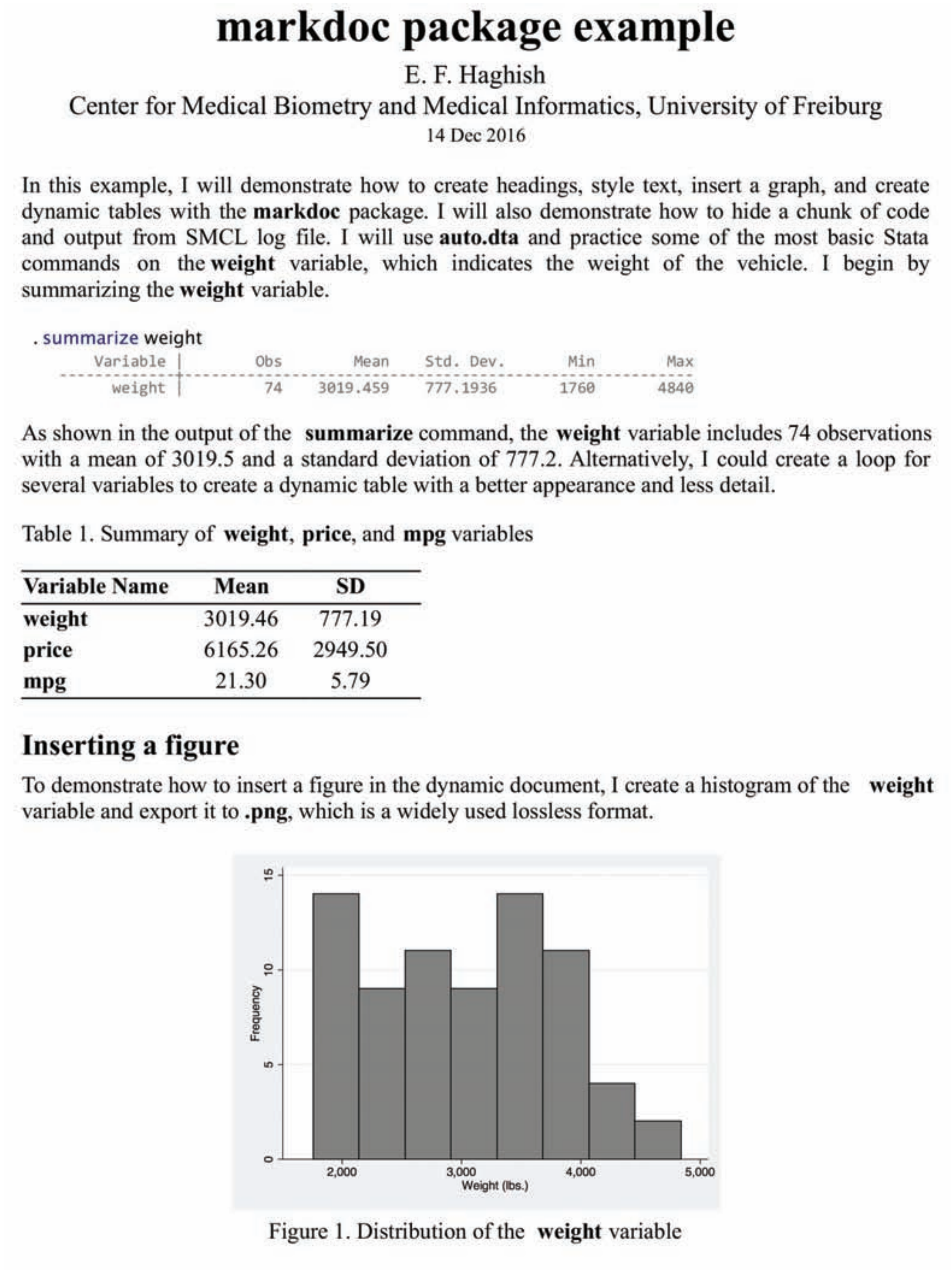
▷ **Example**

```
/***
In this example, I will demonstrate how to create headings, style text, insert
a graph, and create dynamic tables with the __markdoc__ package. I will also
demonstrate how to hide a chunk of code and output from the SMCL log file.
I will use __auto.dta__ and practice some of the most basic Stata commands
on the __weight__ variable, which indicates the weight of the vehicle. I
begin by summarizing the __weight__ variable.
***/
//OFF
sysuse auto, clear
histogram weight, frequency scheme(sj)
quietly graph export graph.png, width(150) replace
//ON
summarize weight
/***
As shown in the output of the __summarize__ command, the __weight__ variable
includes <!r(N)!> observations with a mean of <!%9.1f r(mean)!> and a standard
deviation of <!%9.1f r(sd)!>. Alternatively, I could create a loop for several
variables to create a dynamic table with a better appearance and less detail.
***/
//OFF
foreach var of varlist weight price mpg {
      summarize `var´
      local `var´_mean : display %9.2f r(mean)
      local `var´_sd   : display %9.2f r(sd)
}
//ON
tbl ("Variable name", "Mean", "SD" \                           ///
     "__weight__", `weight_mean´, `weight_sd´ \                ///
     "__price__", `price_mean´, `price_sd´ \                   ///
     "__mpg__", `mpg_mean´, `mpg_sd´),                         ///
     title("Table 1. Summary of the __weight__, __price__, and" ///
       "__mpg__ variables")
/***
Inserting a figure
------------------

To demonstrate how to insert a figure in the dynamic document, I create
a histogram of the __weight__ variable and export it to __.png__,
which is a widely used lossless format.

![Figure 1. Distribution of the __weight__ variable](graph.png)
***/
```

If the example is saved in `example.do`, the following command will test the reproducibility of the do-file and produce the dynamic document:

```
markdoc example.do, replace export(pdf) title("markdoc package example")    ///
        author("E. F. Haghish")                                             ///
        affiliation("Center for Medical Biometry and Medical Informatics,"  ///
        "University of Freiburg") date statax style(formal)
```

◁



markdoc package example

E. F. Haghish

Center for Medical Biometry and Medical Informatics, University of Freiburg

14 Dec 2016

In this example, I will demonstrate how to create headings, style text, insert a graph, and create dynamic tables with the **markdoc** package. I will also demonstrate how to hide a chunk of code and output from SMCL log file. I will use **auto.dta** and practice some of the most basic Stata commands on the **weight** variable, which indicates the weight of the vehicle. I begin by summarizing the **weight** variable.

```
. summarize weight
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
      weight |         74    3019.459    777.1936       1760       4840
```

As shown in the output of the **summarize** command, the **weight** variable includes 74 observations with a mean of 3019.5 and a standard deviation of 777.2. Alternatively, I could create a loop for several variables to create a dynamic table with a better appearance and less detail.

Table 1. Summary of **weight**, **price**, and **mpg** variables

Variable Name	Mean	SD
weight	3019.46	777.19
price	6165.26	2949.50
mpg	21.30	5.79

Inserting a figure

To demonstrate how to insert a figure in the dynamic document, I create a histogram of the **weight** variable and export it to **.png**, which is a widely used lossless format.

Figure 1. Distribution of the **weight** variable

Figure 7. Preview of the document written with Markdown

9 Conclusion

Peer reviewers of quantitative research need transparent documentation of data analysis to review the procedure and reproduce the results. For the same reason, scientific journals are becoming stricter in requiring authors to publish their data and script files as well as make potentially reusable data publicly available for further research (Loder and Groves 2015; Sturges et al. 2015; Nature Publishing Group 2015; Nature Neuroscience Editors 2007; Piwowar, Day, and Fridsma 2007; Sturges et al. 2014; Piwowar and Chapman 2008; McCain 1995). Furthermore, research is becoming more collaborative (Wray 2002; Petre 1994; Subramanyam 1983), which consequently demands within-group data, code sharing, and collaborative programming. In this article, I discussed some of the common challenges in documenting data analysis, and I introduced `markdoc`, a new literate-programming package for Stata.

Literate programming in statistics—that is, explaining and documenting data analysis and statistical codes—can facilitate learning statistics and creating analysis documents (Baumer et al. 2014; Rossini 2001). Literate-programming values clean, well-written, and well-documented code and encourages users to read and comprehend the code (Knuth 1974, 1984). Toward this purpose, `markdoc` facilitates clean code documentation. `markdoc` supports a minimalistic markup language that helps keep documentation simple, easy to read, and appealing; provides several options and styles to simplify LaTeX and HTML documentation; and allows use of the same source to compile documents in various formats. These are just a few of the features that make the package distinctive in the field of statistics.

The simplicity of `markdoc` not only encourages Stata users to practice literate programming but also makes teaching literate programming feasible. Literate programming is advertised as a way to improve students' statistical comprehension and results interpretation (Baumer et al. 2014). Students can use literate-programming tools to write their notes within the statistics software and document the code within their script file. Because `markdoc` supports LaTeX mathematical notations and compiles them to PDF, HTML, and Microsoft Word as well as LaTeX, it is a powerful documentation tool that can be used within the Stata Do-file Editor.

Statistics teachers can also benefit from `markdoc` by creating dynamic PDF slides that include figures, mathematical notations, and Stata commands and output, allowing them to reuse parts of the slides in other lectures. There is also a syntax highlighter that can be used in HTML, PDF, and LaTeX documents and slides. Thus, the `markdoc` package is a complete tool for developing appealing educational materials in addition to writing dynamic-analysis documents.

10 References

Baggerly, K. A., and D. A. Berry. 2011. Reproducible research. *Amstat News*. http://magazine.amstat.org/blog/2011/01/01/scipolicyjan11/.

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.

Gentleman, R., and D. T. Lang. 2007. Statistical analyses and reproducible research. *Journal of Computational and Graphical Statistics* 16: 1–23.

Gruber, J. 2004. Markdown: Syntax. http://daringfireball.net/projects/markdown/syntax.

Haghish, E. F. 2014a. 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.

———. 2014b. Weaver package 3.3.5. http://www.haghish.com/statistics/stata-blog/reproducible-research/weaver.php.

———. 2015. Statax: JavaScript syntax highlighter for Stata. http://www.haghish.com/statax/statax.php.

———. 2016a. github: Stata module for searching and installing Stata packages from GitHub. GitHub. https://github.com/haghish.github.

———. 2016b. markdown: A Stata module to convert Markdown to SMCL language. http://github.com/haghish/markdown.

Jann, B. 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.

Knuth, D. E. 1974. Computer programming as an art. *Communications of the ACM* 17: 667–673.

———. 1983. The WEB system of structured documentation. Technical Report STAN-CS-83-980, Department of Computer Science, Stanford University. http://infolab.stanford.edu/pub/cstr/reports/cs/tr/83/980/CS-TR-83-980.pdf.

———. 1984. Literate programming. *Computer Journal* 27: 97–111.

Kulkarni, A., and J. Truelsen. 2015. WK<html> to pdf. http://wkhtmltopdf.org/.

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.

Loder, E., and T. Groves. 2015. The BMJ requires data sharing on request for all trials. *British Medical Journal* 350: h2373.

MacFarlane, J. 2006. Pandoc: A universal document converter. http://johnmacfarlane.net/pandoc/index.html.

McCain, K. W. 1995. Mandating sharing: Journal policies in the natural sciences. *Science Communication* 16: 403–431.

Nature Neuroscience Editors. 2007. Got data? *Nature Neuroscience* 10: 931.

Nature Publishing Group. 2015. Availability of data, material and methods. http://www.nature.com/authors/policies/availability.html.

Peng, R. D. 2011. Reproducible research in computational science. *Science* 334: 1226–1227.

Petre, M. 1994. A Paradigm, Please–and Heavy on the Culture. In *User-Centred Requirements for Software Engineering Environments*, ed. D. J. Gilmore, R. L. Winder, and F. Détienne, 273–284. Berlin: Springer.

Piwowar, H. A., and W. W. Chapman. 2008. A review of journal policies for sharing research data. In *Proceedings ELPUB 2008 Conference on Electronic Publishing*. Toronto, Canada: ELPUB.

Piwowar, H. A., R. S. Day, and D. B. Fridsma. 2007. Sharing detailed research data is associated with increased citation rate. *PLOS ONE* 2: e308.

Reid, J. 2015. *HTML5 Programmer's Reference*. Berkeley, CA: Apress.

Rossini, A., and F. Leisch. 2003. Literate statistical practice. Working Paper 194, Department of Biostatistics, University of Washington. http://biostats.bepress.com/uwbiostat/paper194/.

Rossini, A. J. 2001. Literate statistical practice. In *Proceedings of the 2nd International Workshop on Distributed Statistical Computing*. Vienna, Austria: DSC.

Stodden, V., F. Leisch, and R. D. Peng, eds. 2014. *Implementing Reproducible Research*. Boca Raton, FL: Chapman & Hall/CRC.

Sturges, P., M. Bamkin, J. Anders, and A. Hussain. 2014. Journals and their policies on research data sharing. https://jordproject.wordpress.com/reports-and-article/journals-and-their-policies-on-research-data-sharing/.

Sturges, P., M. Bamkin, J. H. S. Anders, B. Hubbard, A. Hussain, and M. Heeley. 2015. Research data sharing: Developing a stakeholder-driven model for journal policies. *Journal of the Association for Information Science and Technology* 66: 2445–2455.

Subramanyam, K. 1983. Bibliometric studies of research collaboration: A review. *Journal of Information Science* 6: 33–38.

Wada, R. 2005. outreg2: Stata module to arrange regression outputs into an illustrative table. Statistical Software Components S456416, Department of Economics, Boston College. https://ideas.repec.org/c/boc/bocode/s456416.html.

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.

Wray, K. B. 2002. The epistemic significance of collaborative research. *Philosophy of Science* 69: 150–168.

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.