MarkDoc: Literate Programming in Stata

Abstract.

Rigorous documentation of analysis plan, procedure, and computer codes enhances comprehensibility and transparency of the 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. This article introduces 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, Open Office and LibreOffice ODT, LaTeX, HTML, ePub, and Markdown.

Keywords: markdown, HTML, LaTeX, literate programming, dynamic documents, reproducible research, log file, translator

1 Introduction

In the recent years, concerns about replicability of the study findings as well as reproducibility of the data analysis results in scientific publication have proliferated. In contrast to replication, which requires reimplementing experiments to validate the findings, reproducibility means replicating the data analysis or the computation by an independent researcher using the same data, procedure, and methodology (Baggerly and Berry 2009). Reproducing the analysis can be considered as a minimum standard for evaluating the quantitative results since reproducibility does not necessarily certify quality, sound methodology, correctness of data collection, or validity of the findings (Peng 2011; Stodden et al. 2014). Although, it provides partial transparency for other researchers to validate the analysis procedure and examine or adapt the claims in the scientific publication (Gentleman and Lang 2012).

To support reproducibility of the analysis, a detailed documentation of the methodology and analysis plan, data, and analytic codes and output should be prepared. The documentation can be included in the analysis code as comments to shed light about a particular function or code chunk. However, reading comments requires navigating through script files which can be convoluted, as the size and complexity of the structure 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, making reference about the

code or output and also, updating the documentation after making a change in the code would be very boring, time consuming, and prone to human errors.

Another solution to this problem is to write the documentation within the script files by applying a special notation to separate the documentation from the source code. Next, a software is used to parse the special notations and whether typeset the documentation or compile the code, procedures which are called "weave" and "tangle" respectively. This solution was proposed by Donald Knuth (1984) and is called "Literate Programming." Knuth 1983 developed the `WEB` literate programming software, which provided the means for writing structured documentation within the source code and generating dynamic documentation.

There have been several attempts to adapt literate programming paradigm for statistical data analysis (Leisch 2002; Rossini and Leisch 2003; Rossini 2001; Xie 2013b,a; Lenth 2012). Yet, the only software allowing literate programming for Stata have been *StatWeave* (Lenth 2012), which is written in Java and its Stata package alternative, `texdoc` (Jann 2015). The 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, yet, it is a complex markup language and can harm the readability of the source file (see section 4).

Another shortcoming of *StatWeave* and `texdoc` is that they do not support real-time preview of the document i.e. they do not support writing and viewing the dynamic document interactively without reexecuting the analysis code. Furthermore, they force the users to create a new script file for generating the dynamic analysis document, which consequently 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 it includes special notations for separating LaTeX documentation from the analytic code that are not meaningful for Stata. In practice, the data analysis is developed gradually in a script files and an ideal literate programming software must allow documenting the data analysis and executing it interactively, which is a more natural workflow in the field of statistics, as compared to software development. Besides, `texdoc` includes several commands for initializing and closing the document and including Stata log-files which makes working with it inconvenient, especially for statistics learners.

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

2 Workflow

One of the main features and also a potential problem with dynamic analysis documents is that users select the code and output that they wish to include in the document and dismiss the details that are not essential or necessary for reporting. Hence, it is indispensable to bare in mind that dynamic analysis document is not a replacement for the analysis log-file, which provides maximum transparency about the process of data analysis.

Since the log-file registers every entry in Stata including the markup annotations and text required for typesetting 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. Therefore, as soon as the user opens a Stata smcl log-file and begins the data analysis, `markdoc` can quickly compile a dynamic document interactively as shown below.

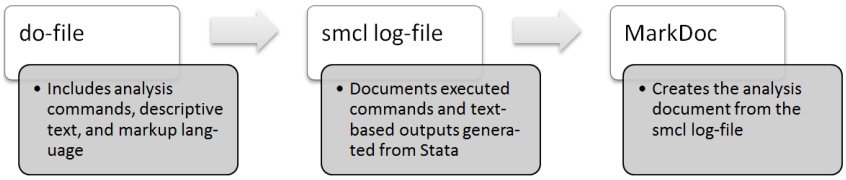

Figure 1: The process of producing dynamic documents with `markdoc`

This workflow has several advantages compared to other common literate programming packages used for data analysis. First of all, the log-file can be used to compile a dynamic document in various formats without rerunning the analysis code. Moreover, dynamic documents can be produced quickly without re-executing the analysis code , since the log-file contains the history of commands and outputs and is updated automatically. Finally, this workflow eliminated the need for creating an additional script file for producing 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 to 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, users 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 is 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 and the `qui log c` which is the abbreviation of `quietly log close` is automatically removed by `markdoc` to

keep the document clean.

▷ **Example**

```
. quietly log using example, replace smcl
. ...
. qui log c

. 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]. In the example above, if the code is saved as a do-file – for example in a file named example.do – `markdoc` can produce an identical document using the script file:

▷ **Example**

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

◁

Both procedures have the same workflow. However, the latter creates a temporary log while executing the do-file and translates the temporary log to a dynamic document.

3 MarkDoc Package

3.1 Features

`markdoc` recognizes Markdown, HTML, and LaTeX markup languages and can export dynamic document to several formats which are PDF, Microsoft Office Docx, Open Office and LibreOffice ODT, LaTeX, HTML, ePub, and Markdown [2]. By applying any of these markup languages users can style the document, insert figures in the dynamic document, create dynamic tables, write dynamic text for interpreting the data analysis, and style the text.

In addition, `markdoc` includes several document styles and it also allows the users to create a dynamic document using external template files e.g. a LaTeX header file, a CSS file, or a Microsoft Word template file (see section 3.3). 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 not only when exporting to a LaTeX document, but also HTML, PDF, Microsoft Word, and OpenOffice.

[1] `markdoc` can take a Mata or an ado-file as an input and generate Stata help files or package vignette. This feature is still under revision and is not discussed in this article

[2] Since the version 3.5, `markdoc` can also produce PDF and HTML dynamic presentation slides. However, this feature goes beyond the purpose of current article and is not discussed

3.2 Installation

`markdoc` requires two other packages which are `weaver` and `statax`, both are hosted on SSC server. `weaver` (Haghish 2014a) is a dynamic document generator which creates LaTeX, HTML, and PDF dynamic documents in Stata. It also supports `markdoc` with three commands for creating dynamic tables, writing dynamic text, and capturing the current graph from Stata and inserting it in the dynamic document. `statax` (Haghish 2015) is a syntax highlighter engine which highlights Stata commands in HTML, PDF, and LaTeX documents. The following commands install `markdoc`, `weaver`, and `statax` packages:

```
. ssc install markdoc
. ssc install weaver
. ssc install statax
```

In addition, `markdoc` requires two third-party software which are Pandoc (MacFarlane 2012) for converting Markdown to other file formats and wkhtmltopdf (Ashish 2015) for creating PDF documents from source written with Markdown or HTML. Furthermore, users who wish to write with LaTeX can also compile a PDF document with `markdoc` command, if a LaTeX distribution is installed on the machine. All of these software are open-source freeware, available for Microsoft Windows, Mac, and Linux operating systems. The packages hosted on SSC server only include the ado and help files and the third-party software can be downloaded and installed manually. Moreover, `markdoc` provides optional automatic installation for Pandoc and wkhtmltopdf which might be more convenient for many users. The manual and automatic installation procedures are further explained below.

3.2.1 Manual installation of third-party software

Pandoc software can be downloaded from *pandoc.org* website and installed manually. Once Pandoc is installed, the path to executable Pandoc on the operating system can be provided to `markdoc` using the `pandoc(`*str*`)` option. Similarly, wkhtmltopdf software can be downloaded from *wkhtmltopdf.org* and be installed anywhere on the machine. Next, the path to the executable wkhtmltopdf file should be provided to `markdoc` using the `printer(`*str*`)` option. For compiling LaTeX to PDF, the proper LaTeX distribution based on the operating system should be downloaded from *latex-project.org* and the path to executable pdfLaTeX compiler should be given to `printer(`*str*`)` option (see the `texmaster` option in section 3.3 in this regard).

The path 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 within a particular global macro. The file includes instructions and examples that how the file paths should be defined.

3.2.2 Automatic installation of third-party software

The `markdoc` command includes the `install` option which downloads 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 OSX (10.9 and 10.10), 32bit and 64bit versions of Microsoft Windows (XP, 7, and 8), Microsoft Windows 10 (64bit), and several Linux systems such as Ubuntu 14.04 (64bit), Mint 17 (32bit and 64bit), and CentOS 7 (64bit). However, manual installation is recommended because it ensures the users install the latest versions of the software and update them frequently.

❑ **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 obtained by using the `sysdir` command which lists Stata's system directories. The default `Weaver` directory paths are shown below based on the operating system for Stata 13 and 14.

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

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

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

❑

3.3 Syntax

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

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

3.3.1 Options

`markdoc` options are as follows:

`pandoc(`*str*`)` specifies the path to the executable Pandoc on the 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 the machine. wkhtmltopdf generates a PDF document from

Markdown and HTML markup languages and pdfLaTeX typesets LaTeX to PDF.

`install` downloads the required third-party software automatically if they are not already installed or accessible (see section 3.2.2).

`linesize(#)` specifies the line size of the document and can range from 80 to 255. `markdoc` also evaluates the linesize of the document and applies the actual linesize automatically, if the linesize is not specified. In general, it is a good practice to keep the line size of the documentation within 80 characters or less and avoid writing long lines. breaking long line in the script file does not influence the dynamic document.

`test` runs an example do-file and generate an HTML and PDF dynamic documents to ensure that the required software are 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`, LaTeX `tex`, Microsoft Office `docx`, and OpenOffice/LibreOffice `odt`. If this option as well as the `markup(`*name*`)` option are not specified, `markdoc` exports a Markdown `md` file by default.

`markup(`*name*`)` specifies the markup language used for annotating the document and it can be `markdown`, `html`, or `latex`. Markdown is the default markup language.

`numbered` turns the numbering system on for Stata commands to number the order of the commands in the dynamic document.

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

`linesize(`*int*`)` specifies the log-file's width and it can range between 80 to 255. If the specified width is less than the actual width of the log-file, some words might appear disjoint in the exported dynamic document.

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

`title(`*str*`)` specifies the title of the dynamic document.

`author(`*str*`)` specifies the author's name.

`affiliation(`*str*`)` specifies the authors' affiliation or any relevant information.

address(*str*) specifies a contact information or any relevant information in the document. For example, it can be used to add a telephone number, e-mail, or mailing address.

date adds the current date to the document.

summary(*str*) adds a text paragraph for the abstract or summary of document.

texmaster creates a template for LaTeX documents and also includes the most common packages for rendering graphs and figures in the document. Without this option, LaTeX document must begin with defining the document class, loading the necessary packages, as well as beginning and ending the document. However, this option automates the process and creates an executable LaTeX document, allowing the users to only focus on the content of the document and keep the source code clean. Users who are not familiar with LaTeX will find this option very handy.

statax highlights the syntax of Stata codes in the HTML and PDF documents, using statax package which is a syntax highlighter engine for Stata.

template(*filename*) applies an external style sheet file. For example, if the document is written in Markdown or HTML and exported to HTML or PDF, a CSS style sheet file can be specified to alter the appearance of dynamic document. Similarly, when the document is exported to Microsoft Word Docx, a reference document with the similar format can be used to alter the style of the word document (e.g. 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 also be used to append the LaTeX header (i.e. required packages, user-defined commands, etc.) to the exported document.

noisily prints extended log for debugging markdoc.

3.4 Testing MarkDoc

After installing the required packages, markdoc can be tested to ensure 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-file name). If the software were installed manually, the pandoc(*str*) and printer(*str*) options should be specified to tell markdoc where it can access Pandoc and wkhtmltopdf. In contrast, if the software were installed automatically, only the test option is enough to carry out the test.

```
. markdoc, test pandoc(str) printer(str)
```

4 Supported Markup languages

In order 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 which are *descriptive* which describe the content of the document such as XML, *procedural* which define how the document should be processed such as PostScript, or *presentational* which frame how the document should be rendered and presented such as HTML and LaTeX (Reid 2015). For writing a literate program, we are in need of a presentational markup language to define the template of the document and render headings and images in the document. `markdoc` supports annotating text using three markup languages which are Markdown, HTML, and LaTeX. These markup languages are briefly compared with examples in the following sections.

Similar to other literate programming software and regardless of the markup language used for writing 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 the "`/***`" sign and ending with a "`***/`" sign, each on a separate line. This notation allows the do-file to remain executable since the notation does not interfere with the code execution in Stata.

❏ **Technical note**

In general, markup languages should not be mixed with one another. If the document is written in HTML or LaTeX, the `markup(`*name*`)` option should be specified because the engine of `markdoc` works differently based on the used markup language.

❏

4.1 Markdown

Markdown (Gruber 2004) is a minimalistic markup language and has an intuitive syntax which makes it preferable to HTML and LaTeX. Table 1 presents the most common Markdown syntax. The complete documentation of the Markdown commands can be found on *daringfireball.net* website.

Dynamic documents written in Markdown can include LaTeX mathematical notations and also be exported to HTML and LaTeX as well as any of the other supported document formats. Therefore it also provides a greater flexibility compared to HTML and LaTeX. The example below demonstrates using Markdown for creating heading, subheading, and styling text in a do-file and compiles it to a Microsoft Word document.

Table 1: Markdown syntax references

Markdown syntax	Results
`Heading 1` `=========`	# Heading 1
`Heading 2` `---------`	## Heading 2
`### Heading 3`	### Heading 3
`#### Heading 4`	#### Heading 4
`plain text paragraph`	plain text paragraph
`**Bold** or __Bold__ text`	**Bold** or **Bold** text
`*Italic* or _Italic_ text`	*Italic* or *Italic* text
`` `inline` code ``	`inline` code
`superscript^2^`	superscript^2

`1. Ordered item1`
`  1. Sublist 1`
`    1. Subsublist 1`
`2. Ordered item2`
	1. Ordered item1
	  1. Sublist 1
	  1. Subsublist 1
	2. Ordered item2

`* Unordered item1`
`  * Sublist 1`
`  * Subsublist 2`
`* Unordered item2`
	• Unordered item1
	  - Sublist 1
	  * Subsublist 2
	• Unordered item2

`![Text](`*filename*`)`	Inserting an image with description
`[Text](http://url)`	Inserting a hyperlink

Example

```
. qui log using example, replace

/***
```

```
This is a heading
=================

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

Text can also appear as _Italic_ or __Bold__.
***/

. qui log c
. markdoc example, export(docx)
```

◁

4.2 HTML and LaTeX

Despite its simplicity, convenience, and flexibility, Markdown only provides the basic styling commands such as writing headings, making text bold or italic, adding a hyper-link and inserting a graph or image in the document. In contrast, HTML and LaTeX provide much more options for styling and annotating the dynamic document. But dynamic documents written in these markup languages will not be as readable as documents written with Markdown and can only produce PDF or HTML and LaTeX respectively. Yet, HTML and LaTeX are both supported by `markdoc` package for writing dynamic documents, although 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**

```
. qui 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>
***/

. qui log c
. markdoc example, export(html) markup(html)
```

◁

The previous example is repeated using LaTeX. In addition, the `texmaster` option was added to `markdoc` command in order to create the document's template automatically and allow compiling the document. Since `markdoc` is asked for compiling LaTeX to PDF, the path to pdfLaTeX should also be specified which is different for each operating system and LaTeX distribution.

▷ **Example**

```
. qui log using example, replace
```

```
/***
\section{This is a heading}

\subsection{This is a subheading}

Text can also appear as \textit{Italic} or \textbf{Bold}.
***/

. qui log c
. markdoc example, texmaster export(pdf) markup(latex)           ///
  printer("/path/pdflatex")
```

◁

❑ **Technical note**

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

❑

4.3 Additional Notation Markers

As noted earlier, `markdoc` package can include a subset of the smcl log-file's code and output to create a less-detailed documentation or concentrate on the main result of the analysis. For example, the user might not be interested to include the process of data preparation and exploration in the dynamic document, since all the executed commands and outputs are already documented in the log-file. Nevertheless, 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 a 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
`//IMPORT` *filename*	Include an external file (Markdown, HTML, LaTeX)

For excluding the Stata command or output, the code line should begin with `/**/` or `/***/` markers respectively. In order to exclude a section of code and

output from the smcl log-file the `//OFF` and `//ON` markers should be placed on a separate lines in the do-file, which makes `markdoc` ignore anything that appears in between.

There are a number of user-written packages for exporting Stata outputs to LaTeX and HTML files such as `tabout` (Watson 2007), `weaver` (Haghish 2014a), and `synlight` (Haghish 2014b). `markdoc` provides the "`//IMPORT` *filename*" command , which can be used to include external files in the dynamic document. For example, using the `outreg2` (Wada 2014) package, a regression table can be exported to a LaTeX file and imported in 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 descriptin or methodology that is not supposed to changed based on the analysis results.

Since these markers are in the form of comments, they do not influence the execution of the code in Stata. However, Stata imposes some limits for working with comment markers, similar to any other comment signs. Namely, comments cannot be executed from user-written programs and also, cannot be re-executed in loops. To tackle this problem, dynamic text should be written and executed by Stata which is discussed in the following section.

4.4 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 of the supported formats. For writing inline notations, the notations should begin and end with a single dollar sign and for placing the notations on a separate line, double dollar signs can be used, as shown in the example below:

▷ Example

```
. qui log using example, replace

/***
Writing mathematical notations
==============================

The text paragraph can include mathematical notations. For example, this
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: $$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$$
***/

. qui log c
. markdoc example, export(docx)
```

◁

5 Writing dynamic documentation

`markdoc` borrows three additional commands from `weaver` package which are `img`, `txt`, and `tbl`, used for capturing and including graphs in the dynamic

document, writing dynamic text, and creating Markdown dynamic tables respectively. These commands are discussed with examples in this section.

5.1 Dynamic text

Writing dynamic text – including macros and scalars within text – is a useful feature in literate programming. By making a change in the data and re-executing the code, dynamic text will automatically update the values mentioned in the text. To provide writing dynamic text, the `txt` command – which is used for a similar purpose in the `weaver` package – is borrowed and updated to support `markdoc`. The syntax of the `txt` command is as follows:

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

Similar to the display directive of the `display` command in Stata, the `display_directive` can be a double-quoted string, compound double-quoted string, mathematical operation, scalar, as well as 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 which can be used for discussing a line of code in the analysis document. In the next example, 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

```
. qui log using example, replace
. sysuse auto, clear
. quietly summarize price
. txt r(N) " observations are included in the data set."
. qui log c
```

◁

Note that for executing markup and documentation within a loop or user-written programs, the `txt` command should be applied.

5.2 Dynamic table

Similar to the `txt`, the `tbl` command is also borrowed from the `weaver` package to create simple markdown tables which 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 the `tbl` is similar to `matrix input` command in Stata, 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 a formatting directive (`%fmt`). Using the auto.dta data set, I demonstrate creating a simple table that includes the number of observations, mean, and standard deviation of the *Price* variable.

▷ **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 2: Preview of the dynamic table

◁

5.3 Dynamic figures

Markdown, HTML, and LaTeX have specific syntax for including figures in the document. In order to automatize the process of including a Stata graph in the document, first it should 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, using the *auto* data set:

▷ **Example**

```
. qui 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)
***/

. qui log c
. markdoc example, export(docx) statax
```

◁

It is noteworthy that for including figures in the analysis document, they should be exported to any of the common graphical formats such as PNG, GIF, JPEG, etc. PNG (Portable Network Graphics) 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.

An alternative procedure is provided by borrowing the `img` command from the `weaver` package. The syntax of the `img` command is as follows:

`img` [`using` *filename*] [`, title(`*str*`) width(`*int*`) height(`*int*`) markup(`*name*`) left center`]

As indicated by the syntax, the `img` command can be used in two different ways. First, to include an image file that is already stored on the machine i.e. "`img using` *filename*". The `img` command can also 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 in the dynamic document.

▷ Example

```
. qui log using example, replace
. sysuse auto, clear
. histogram price
. img
. markdoc example, export(html) statax
```

◁

By default, the `img` command prints Markdown code in the log, unless the `markup(`*str*`)` option is used to define `html` or `latex` markup. The `width(`*int*`)` and `height(`*int*`)` commands, which are used to specify the figure size in the dynamic document, only work in HTML and LaTeXbecause Markdown cannot resize images.

6 Examples and notes

In the following example, I will use Markdown syntax for writing and styling text and inserting a graph to the document and export it in PDF format. The example also includes the `txt` and `tbl` commands for writing dynamic text and creating a dynamic table as well as the notation markers for hiding parts of the log-file.

▷ Example

```
. qui log using example, replace

  /***
  In this example, I will demonstrate how to create headings, style text, insert
  a graph, and create dynamic tables using MarkDoc package. I will also
  demonstrate how to hide a chunk of code and output from the smcl log file. I
  will use the _auto.dta_ data set and practice some of the most basic Stata
  commands on the _weight_ variable which indicates the Weight of the vehicle. I
  will begin with summarizing the _Weight_ variable.
  ***/

  //OFF
. sysuse auto, clear
. histogram weight, frequency scheme(sj)
. quietly graph export graph.png, width(300) replace
  //ON

. summarize weight

. txt "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 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 _Weight_, _Price_, and _MPG_ variables")

  /***
  Inserting a figure
  ------------------

  In order to demonstrate how to insert a figure in the dynamic document, I create
  a histogram of the _Weight_ variable and export it to PNG (Portable Network
  Graphics), which is a widely used lossless format.

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

. qui log c
. markdoc example, replace export(pdf) title("MarkDoc Package Example")          ///
  author("E. F. Haghish") affiliation("University of Freiburg (IMBI)") date
```

MarkDoc Package Example

E. F. Haghish
University of Freiburg (IMBI)
30 Dec 2015

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

```
1 . 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 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	2949.5
MPG	21.3	5.79

Inserting a figure

In order to demonstrate how to insert a figure in the dynamic document, I creat e a histogram of the *Weight* variable and export it to PNG (Portable Network Graphics), which is a widely used lossless format.

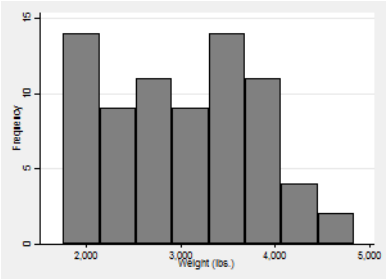

Figure 1. Distribution of the *Weight* variable

7 Conclusion

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

Literate programming in statistics – explaining and documenting the data analysis and statistical codes – can facilitate learning statistics and creating analysis documents (Baumer et al. 2014; Rossini 2001). Nevertheless, the dynamic document is only a product of literate programming but not its the main purpose. In other words, literate programming values a clean, well-written, and well-documented code and assumes it encourages users to read and comprehend the code easier (Knuth 2007, 1984). Toward this purpose, `markdoc` was programmed in a way to facilitate clean code documentation. Supporting a minimalistic markup language that helps to keep the documentation simple, easy to read, and appealing such as Markdown, providing several options and styles for simplifying LaTeX and HTML documentation, and allowing to use the same source to compile documents in various formats are a few features that makes the package distinctive in the field of statistics. Perhaps the biggest advantage of `markdoc` is that it makes data analysis a joyful activity by encouraging the user to communicate the analysis more actively.

The simplicity of `markdoc` not only encourages Stata users to practice literate programming, but also it makes teaching literate programming in introductory courses feasible, which is advertised as a way of improving 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. Supporting LaTeX mathematical notations and compiling them to PDF, HTML, Microsoft Word, as well as LaTeX turn `markdoc` to a powerful documentation tool that can be used within Stata Do-file Editor.

Similarly, statistics teachers can get benefit from `markdoc` to create dynamic PDF slides that include figures, mathematical notations, Stata commands and output, which also allows them to reuse parts of their slides in other lectures. Incorporating a syntax highlighter that can be used in HTML, PDF, and LaTeX documents and slides, makes the package a complete tool for developing appealing educational material as well as writing dynamic analysis documents.

8 References

2007. Got Data? *Nature Neuroscience* 10(8). URL http://www.nature.com/neuro/journal/v10/n8/full/nn0807-931.html.

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

Ashish, T., Kulkarni; Jakob. 2015. WK HTML to PDF. URL http://wkhtmltopdf.org/.

Baggerly, K. A., and D. A. Berry. 2009. Reproducible research.

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).

Gentleman, R., and D. T. Lang. 2012. Statistical analyses and reproducible research. *Journal of Computational and Graphical Statistics* .

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

Haghish, E. F. 2014a. Rethinking Literate Programming in Statistics. URL http://www.haghish.com/statistics/stata-blog/reproducible-research/weaver.php.

———. 2014b. synlight: Stata module to highlight syntax in SMCL and translate to HTML format. *Statistical Software Components* . URL https://ideas.repec.org/c/boc/bocode/s457894.html.

———. 2015. Statax: JavaScript Syntax Highlighter for Stata. URL http://www.haghish.com/statax/statax.php.

Jann, B. 2015. Creating LaTeX documents from within Stata using texdoc. Technical report, University of Bern, Department of Social Sciences.

Knuth, D. E. 1983. *The WEB system of structured documentation*. Department of Computer Science, Stanford University.

———. 1984. Literate programming. *The Computer Journal* 27(2): 97–111.

———. 2007. Computer programming as an art. In *ACM Turing award lectures*, 1974. ACM.

Leisch, F. 2002. Sweave: Dynamic generation of statistical reports using literate data analysis. In *Compstat*, 575–580. Springer.

Lenth, R. V. 2012. StatWeave Users' Manual . URL 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. *BMJ* 350: h2373.

MacFarlane, J. 2012. About Pandoc. URL http://johnmacfarlane.net/pandoc/index.html.

McCain, K. W. 1995. Mandating sharing journal policies in the natural sciences. *Science Communication* 16(4): 403–431.

Peng, R. D. 2011. Reproducible research in computational science. *Science (New York, Ny)* 334(6060): 1226.

Petre, M. 1994. A Paradigm, Please-and Heavy on the Culture. In *User-Centred Requirements for Software Engineering Environments*, ed. D. Gilmore, R. Winder, and F. Détienne, vol. 123 of *NATO ASI Series*, 273–284. Springer Berlin Heidelberg. URL http://dx.doi.org/10.1007/978-3-662-03035-6_21.

Piwowar, H. A., and W. W. Chapman. 2008. A review of journal policies for sharing research data. In *ELPUB2008*.

Piwowar, H. A., R. S. Day, and D. B. Fridsma. 2007. Sharing detailed research data is associated with increased citation rate. *PloS one* 2(3): e308.

Reid, J. 2015. *HTML5 Programmer's Reference*. Appress.

Rossini, A., and F. Leisch. 2003. Literate statistical practice .

Rossini, A. J. 2001. Literate Statistical Practice. In *Proceedings of the 2nd International Workshop on Distributed Statistical Computing (DSC 2001)*.

Stodden, V., F. Leisch, and R. D. Peng. 2014. *Implementing reproducible research*. CRC Press.

Sturges, P., M. Bamkin, J. Anders, and A. Hussain. 2014. Access to Research Data: Addressing the Problem through Journal Data Sharing Policies .

Sturges, P., M. Bamkin, J. H. 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* n/a–n/a. URL http://dx.doi.org/10.1002/asi.23336.

Subramanyam, K. 1983. Bibliometric studies of research collaboration: A review. *Journal of information Science* 6(1): 33–38.

Wada, R. 2014. OUTREG2: Stata module to arrange regression outputs into an illustrative table. *Statistical Software Components* .

Watson, I. 2007. *Publication quality tables in Stata: a tutorial for the tabout program*. URL http://www.ianwatson.com.au/stata/tabout_tutorial.pdf.

Wray, K. B. 2002. The epistemic significance of collaborative research. *Philosophy of Science* 69(1): 150–168.

Xie, Y. 2013a. *Dynamic Documents with R and knitr*. CRC Press.

———. 2013b. knitr: A general-purpose package for dynamic report generation in R. *R package version* 1(7).