

STATA REPORTING REFERENCE MANUAL

RELEASE 19



A Stata Press Publication
StataCorp LLC
College Station, Texas



® Copyright © 1985–2025 StataCorp LLC
All rights reserved
Version 19

Published by Stata Press, 4905 Lakeway Drive, College Station, Texas 77845

ISBN-10: 1-59718-442-X

ISBN-13: 978-1-59718-442-7

This manual is protected by copyright. All rights are reserved. No part of this manual may be reproduced, stored in a retrieval system, or transcribed, in any form or by any means—electronic, mechanical, photocopy, recording, or otherwise—without the prior written permission of StataCorp LLC unless permitted subject to the terms and conditions of a license granted to you by StataCorp LLC to use the software and documentation. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document.

StataCorp provides this manual “as is” without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. StataCorp may make improvements and/or changes in the product(s) and the program(s) described in this manual at any time and without notice.

The software described in this manual is furnished under a license agreement or nondisclosure agreement. The software may be copied only in accordance with the terms of the agreement. It is against the law to copy the software onto DVD, CD, disk, diskette, tape, or any other medium for any purpose other than backup or archival purposes.

The automobile dataset appearing on the accompanying media is Copyright © 1979 by Consumers Union of U.S., Inc., Yonkers, NY 10703-1057 and is reproduced by permission from CONSUMER REPORTS, April 1979.

Stata, **STATA**, Stata Press, Mata, **MATA**, NetCourse, and NetCourseNow are registered trademarks of StataCorp LLC.

Stata and Stata Press are registered trademarks with the World Intellectual Property Organization of the United Nations.

StataNow is a trademark of StataCorp LLC.

Other brand and product names are registered trademarks or trademarks of their respective companies.

For copyright information about the software, type `help copyright` within Stata.

The suggested citation for this software is

StataCorp. 2025. *Stata 19*. Statistical software. StataCorp LLC.

The suggested citation for this manual is

StataCorp. 2025. *Stata 19 Reporting Reference Manual*. College Station, TX: Stata Press.

Contents

Intro	Introduction to reporting manual	1
docx2pdf	Convert a Word (.docx) document to a PDF file	4
Dynamic documents intro	Introduction to dynamic documents	7
Dynamic tags	Dynamic tags for text files	12
dyndoc	Convert dynamic Markdown document to HTML or Word (.docx) document	19
dyntext	Process Stata dynamic tags in text file	33
html2docx	Convert an HTML file to a Word (.docx) document	38
markdown	Convert Markdown document to HTML file or Word (.docx) document	42
putdocx intro	Introduction to generating Office Open XML (.docx) files	49
putdocx begin	Create an Office Open XML (.docx) file	60
putdocx collect	Add a table from a collection to an Office Open XML (.docx) file	69
putdocx pagebreak	Add breaks to an Office Open XML (.docx) file	75
putdocx paragraph	Add text or images to an Office Open XML (.docx) file	79
putdocx table	Add tables to an Office Open XML (.docx) file	96
Appendix for putdocx	Appendix for putdocx entries	119
putexcel	Export results to an Excel file	123
putexcel advanced	Export results to an Excel file using advanced syntax	149
putpdf intro	Introduction to generating PDF files	165
putpdf begin	Create a PDF file	170
putpdf collect	Add a table from a collection to a PDF file	174
putpdf pagebreak	Add breaks to a PDF file	181
putpdf paragraph	Add text or images to a PDF file	184
putpdf table	Add tables to a PDF file	191
Appendix for putpdf	Appendix for putpdf entries	212
set docx	Format settings for blocks of text	214
Glossary		217
Subject and author index		218

Cross-referencing the documentation

When reading this manual, you will find references to other Stata manuals, for example, [\[U\] 27 Overview of Stata estimation commands](#); [\[R\] regress](#); and [\[D\] reshape](#). The first example is a reference to chapter 27, *Overview of Stata estimation commands*, in the *User's Guide*; the second is a reference to the `regress` entry in the *Base Reference Manual*; and the third is a reference to the `reshape` entry in the *Data Management Reference Manual*.

All the manuals in the Stata Documentation have a shorthand notation:

[GSM]	<i>Getting Started with Stata for Mac</i>
[GSU]	<i>Getting Started with Stata for Unix</i>
[GSW]	<i>Getting Started with Stata for Windows</i>
[U]	<i>Stata User's Guide</i>
[R]	<i>Stata Base Reference Manual</i>
[ADAPT]	<i>Stata Adaptive Designs: Group Sequential Trials Reference Manual</i>
[BAYES]	<i>Stata Bayesian Analysis Reference Manual</i>
[BMA]	<i>Stata Bayesian Model Averaging Reference Manual</i>
[CAUSAL]	<i>Stata Causal Inference and Treatment-Effects Estimation Reference Manual</i>
[CM]	<i>Stata Choice Models Reference Manual</i>
[D]	<i>Stata Data Management Reference Manual</i>
[DSGE]	<i>Stata Dynamic Stochastic General Equilibrium Models Reference Manual</i>
[ERM]	<i>Stata Extended Regression Models Reference Manual</i>
[FMM]	<i>Stata Finite Mixture Models Reference Manual</i>
[FN]	<i>Stata Functions Reference Manual</i>
[G]	<i>Stata Graphics Reference Manual</i>
[H2OML]	<i>Machine Learning in Stata Using H2O: Ensemble Decision Trees Reference Manual</i>
[IRT]	<i>Stata Item Response Theory Reference Manual</i>
[LASSO]	<i>Stata Lasso Reference Manual</i>
[XT]	<i>Stata Longitudinal-Data/Panel-Data Reference Manual</i>
[META]	<i>Stata Meta-Analysis Reference Manual</i>
[ME]	<i>Stata Multilevel Mixed-Effects Reference Manual</i>
[MI]	<i>Stata Multiple-Imputation Reference Manual</i>
[MV]	<i>Stata Multivariate Statistics Reference Manual</i>
[PSS]	<i>Stata Power, Precision, and Sample-Size Reference Manual</i>
[P]	<i>Stata Programming Reference Manual</i>
[RPT]	<i>Stata Reporting Reference Manual</i>
[SP]	<i>Stata Spatial Autoregressive Models Reference Manual</i>
[SEM]	<i>Stata Structural Equation Modeling Reference Manual</i>
[SVY]	<i>Stata Survey Data Reference Manual</i>
[ST]	<i>Stata Survival Analysis Reference Manual</i>
[TABLES]	<i>Stata Customizable Tables and Collected Results Reference Manual</i>
[TS]	<i>Stata Time-Series Reference Manual</i>
[I]	<i>Stata Index</i>
[M]	<i>Mata Reference Manual</i>

Description

This manual documents Stata's features for reporting. With the commands described here, you can create reproducible reports in Word, Excel, PDF, and HTML formats. These reports can be customized to include formatted text, tables of Stata results, and graphs.

Remarks and examples

Remarks are presented under the following headings:

Introduction
Exporting to a Word (.docx) file
Exporting to a PDF file
Exporting to an Excel file
Creating dynamic documents
Converting file types

Introduction

Stata's commands for exporting estimation results, summary statistics, and graphs deliver neatly formatted reports in Word, Excel, PDF, and HTML files.

There are two varieties of commands for creating reports. The first variety creates Word documents, Excel files, and PDF documents that incorporate stored results from Stata commands in formatted text and tables. The `putdocx`, `putpdf`, and `putexcel` suites of commands create documents in this manner. The second variety creates HTML and Word documents that include the full output from Stata commands and allows you to format the text using Markdown. The `dyndoc` and `dyntext` commands, which are discussed in *Creating dynamic documents*, incorporate Stata output in this manner.

Below, we briefly describe each of the report-generating commands. We also review the commands that allow you to convert files from one type to another.

Exporting to a Word (.docx) file

The `putdocx` suite of commands creates Word (.docx) documents with embedded Stata results. With single-line commands, you can export a whole estimation table, an image, or a matrix to a document. You can also build complex tables with custom layouts. The suite allows you to create a document complete with formatted text and Stata results without leaving Stata.

[RPT] putdocx intro	Introduction to generating Office Open XML (.docx) files
[RPT] putdocx begin	Create an Office Open XML (.docx) file
[RPT] putdocx pagebreak	Add breaks to an Office Open XML (.docx) file
[RPT] putdocx paragraph	Add text or images to an Office Open XML (.docx) file
[RPT] putdocx table	Add tables to an Office Open XML (.docx) file
[RPT] putdocx collect	Add a table from a collection to an Office Open XML (.docx) file

We recommend that you read [\[RPT\] putdocx intro](#) first for an overview of the putdocx commands and how you use them. Then you will want to review [\[RPT\] putdocx begin](#) to learn how to create a .docx file in memory. Once you have created a .docx file with putdocx begin, you can refer to [\[RPT\] putdocx paragraph](#) for exporting text and images to your file. You can also refer to [\[RPT\] putdocx table](#) and [\[RPT\] putdocx collect](#) for exporting tables of results to your file.

The putdocx suite allows you to interact Stata’s capabilities with Word’s additional formatting features. You can create a .docx file complete with Stata results from within Stata, but you might also append fragments created in both Stata and Word. See [Workflow options for report building](#) in [\[RPT\] putdocx intro](#) for different ways to create Word documents and how to determine which method is most appropriate for the report you want to create.

Word documents can also be created using the dynamic documents commands described below in [Creating dynamic documents](#).

Exporting to a PDF file

The putpdf suite of commands creates PDF files with Stata results. With these commands, you can incorporate formatted text, summary statistics, regression results, images, customized tables, and matrices in your document.

[RPT] putpdf intro	Introduction to generating PDF files
[RPT] putpdf begin	Create a PDF file
[RPT] putpdf pagebreak	Add breaks to a PDF file
[RPT] putpdf paragraph	Add text or images to a PDF file
[RPT] putpdf table	Add tables to a PDF file
[RPT] putpdf collect	Add a table from a collection to a PDF file

We recommend that you read [\[RPT\] putpdf intro](#) first for an overview of the putpdf commands and how you use them. In [\[RPT\] putpdf begin](#), we demonstrate how to create a file in memory. Once you have done so, you can refer to [\[RPT\] putpdf paragraph](#), [\[RPT\] putpdf table](#), and [\[RPT\] putpdf collect](#) for details on embedding text, images, and tables in a PDF file.

Exporting to an Excel file

With putexcel, you can export Stata results to an Excel workbook, including estimation results, matrices, and images. You can write Stata expressions as well as Excel formulas to a workbook and save portions of your work to separate sheets.

[RPT] putexcel	Export results to an Excel file
[RPT] putexcel advanced	Export results to an Excel file using advanced syntax

We recommend that you read [\[RPT\] putexcel](#) first to learn the basics of exporting Stata results to Excel. In [\[RPT\] putexcel advanced](#), we provide advanced syntax for exporting multiple types of results simultaneously and for formatting existing contents of cells.

Creating dynamic documents

Stata's dynamic document commands allow you to embed Stata output in text files and to create HTML files and Word documents from Markdown text and Stata output. Dynamic tags are used to process Stata commands in a text file; they run the code and export the output to the destination file. To create text files with Stata output, you simply enclose Stata commands within these [dynamic tags](#) throughout your source file and then use [dyntext](#) to create the output file. To create HTML files and Word documents, you can combine Stata dynamic tags and Markdown text in a file and then use [dyndoc](#) to convert it to an HTML file or Word document. [dyndoc](#) calls on [markdown](#) to process the Markdown text.

[RPT] Dynamic documents intro	Introduction to dynamic documents
[RPT] Dynamic tags	Dynamic tags for text files
[RPT] dyndoc	Convert dynamic Markdown document to HTML or Word (.docx) document
[RPT] dyntext	Process Stata dynamic tags in text file
[RPT] markdown	Convert Markdown document to HTML file or Word (.docx) document

We recommend reading [\[RPT\] Dynamic documents intro](#) first because it demonstrates the process of using dynamic tags in your text file, converting it to an output text file, and converting it to an output HTML file or Word document. After reading that entry, you can review [\[RPT\] Dynamic tags](#) for the list of tags that are available for including Stata output in your file. You will find the relevant tags for running Stata commands, including graphs in your file, and displaying Stata expressions.

Converting file types

Stata also has commands for converting files from HTML to Word and from Word to PDF. These commands may be used whether the original files were created using one of the Stata commands listed above or otherwise.

[RPT] html2docx	Convert an HTML file to a Word (.docx) document
[RPT] docx2pdf	Convert a Word (.docx) document to a PDF file

Description

docx2pdf converts a Word (.docx) document to a PDF file.

Quick start

Convert Word document `myfile.docx` to a PDF saved as `myfile.pdf`

```
docx2pdf myfile
```

Same as above, but save the PDF as `mypdf.pdf`

```
docx2pdf myfile, saving(mypdf)
```

Same as above, and overwrite the existing `mypdf.pdf`

```
docx2pdf myfile, saving(mypdf) replace
```

Syntax

```
docx2pdf srcfile [ , options ]
```

srcfile is a .docx file. If *srcfile* is specified without an extension, .docx is assumed. If *srcfile* contains embedded spaces or other special characters, enclose it in double quotes.

options

Description

saving(*targetfile*)

specify the target PDF file to be saved

replace

replace the target PDF file if it already exists

nomsg

suppress message with link to *targetfile*

Options

saving(*targetfile*) specifies the target PDF file to be saved. If *targetfile* is specified without an extension, .pdf is assumed. If *targetfile* contains embedded spaces or other special characters, enclose it in double quotes. If saving() is not specified, the target filename is constructed using the source filename (*srcfile*) with the .pdf extension.

replace specifies that the target PDF file be replaced if it already exists.

nomsg suppresses the message that contains a link to the target file.

Remarks and examples

docx2pdf converts Word (.docx) documents to PDF files. This command is most useful when you want to convert an HTML file to a PDF file; in this case, you would first use [html2docx](#) to convert the HTML file to a Word document and then use docx2pdf to convert it to a PDF file. If you would like to convert a dynamic Markdown document to a PDF file, you can use [dyndoc](#) to create a Word document and then use docx2pdf to convert that Word document to a PDF file.

If you are wanting to embed Stata results and graphs in a PDF file, you can also use the putpdf suite; see [\[RPT\] putpdf intro](#).

► Example 1: Convert Word document to a PDF file

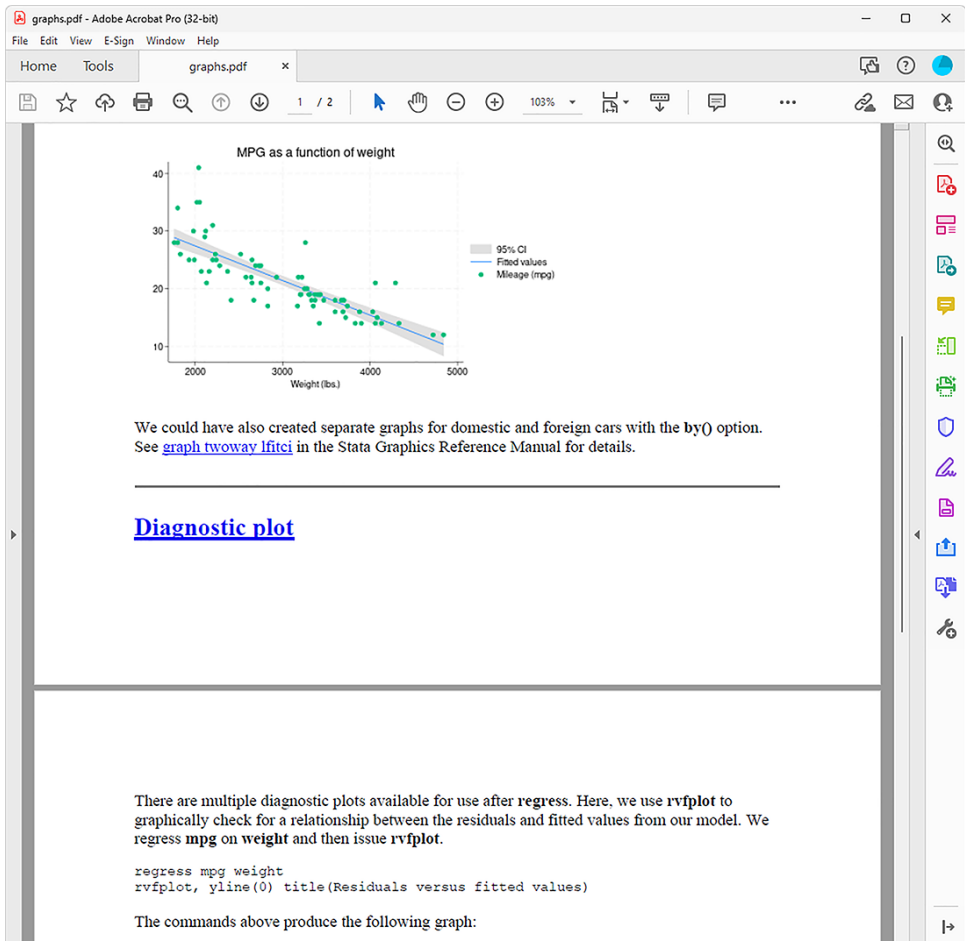
We have a Word document, `graphs.docx`, which you can download to your current working directory by typing

```
. copy https://www.stata-press.com/data/r19/reporting/graphs.docx .
```

To convert this file to a PDF file, we type

```
. docx2pdf graphs.docx
```

Because we did not specify a filename for our target file, the source filename was used with the `.pdf` extension. Thus, the file `graphs.pdf` was saved. Here is a portion of this file:



You can see the whole file at <https://www.stata-press.com/data/r19/reporting/graphs.pdf>.

Also see

[RPT] [html2docx](#) — Convert an HTML file to a Word (.docx) document

[RPT] [putpdf intro](#) — Introduction to generating PDF files

Description

Stata's dynamic document commands create text files, Word (.docx) documents, and HTML files that include Stata results. With these commands, you can create documents that combine text with summary statistics, regression results, graphs, and other Stata results. You can include the full output of Stata commands or incorporate individual values from the results of commands. Word documents and HTML files can easily be customized using the Markdown text-formatting language.

See the following manual entries for details on dynamic documents:

[RPT] Dynamic tags	Dynamic tags for text files
[RPT] dyndoc	Convert dynamic Markdown document to HTML or Word (.docx) document
[RPT] dyntext	Process Stata dynamic tags in text file
[RPT] markdown	Convert Markdown document to HTML file or Word (.docx) document

These documents are dynamic because, as your data change, you simply rerun the `dyndoc` or `dyntext` command that creates your dynamic document, and the HTML file, Word document, or text file is updated with the new results.

Remarks and examples

Creating files with Stata output is straightforward with `dyntext` and `dyndoc`. The former processes Stata commands within a plain text file to create a text file that includes Stata output. The latter converts a text file with Markdown text and Stata commands into a formatted HTML file or Word document file with Stata output. The Stata commands are processed according to the [dynamic tags](#) that indicate how commands, output, results of expressions, and graphs should be inserted in a document. Markdown, a simple markup language with a formatting syntax based on plain text, is processed by `dyndoc`. This allows you to include headings, subheadings, bold and italic font, text boxes, bulleted lists, and more in the HTML files and Word documents generated by this command.

Below, we briefly overview and demonstrate the dynamic document creation commands. See the individual entries for the syntax and additional examples.

► Example 1: Using dynamic tags

Whether you want to create a text file, Word document, or an HTML file, you will use dynamic tags in your source file to embed Stata output in your destination file. Different tags are available to include Stata output, expressions, and graphs in the destination file. With dynamic tags, you can control whether the Stata command, the output, or both are included in the final document. For example, in the following text file we load `auto.dta` and then summarize `mpg`:

```

-----begin example.txt-----
Using Stata dynamic tags
=====
I am going to examine fuel efficiency using -auto.dta-. First, I load the dataset:

<<dd_do>>
sysuse auto, clear
<</dd_do>>

Now I -summarize mpg-, but I only display the output, not the command:

<<dd_do: nocommands>>
summarize mpg
<</dd_do>>
-----end example.txt-----

```

You can type

```
. copy https://www.stata-press.com/data/r19/reporting/example.txt .
```

to copy example.txt to your current working directory.

We use the «dd_do» dynamic tag to run the sysuse command. However, when summarizing mpg, we specify the nocommands attribute to suppress the command in the output file. Attributes, which can be thought of as options, modify the tag's behavior. See [RPT] [Dynamic tags](#) for a full list of dynamic tags.



► Example 2: Create a text file with Stata output

Having enclosed the Stata commands within the dynamic tags in our text file, we can use dyntext to embed the output from those commands in a text file:

```
. dyntext example.txt, saving(output1.txt)
```

This command produces the following:

output1 - Notepad					
File Edit Format View Help					
Using Stata dynamic tags					
=====					
I am going to examine fuel efficiency using -auto.dta-. First, I load the dataset:					
. sysuse auto, clear					
(1978 automobile data)					
Now I -summarize mpg-, but I only display the output, not the command:					
Variable	Obs	Mean	Std. dev.	Min	Max

mpg	74	21.2973	5.785503	12	41

This output1.txt file is available at <https://www.stata-press.com/data/r19/reporting/>.

For a more detailed example of including Stata output in text files, see [RPT] [dyntext](#).



► Example 3: Create an HTML file with Stata output

Suppose that now we want to convert `example.txt` to an HTML file. We could use `dyndoc` instead of `dyntext`, but the output would not be formatted nicely; you can try it for yourself to see the difference. In fact, the beauty of `dyndoc` is that it can process Markdown-formatted text and embed Stata output in the destination file. Let's see `dyndoc` in its full potential by adding Markdown text to our previous file. We still need to use dynamic tags to process the Stata code, but rather than using “-” to indicate Stata command names, variable names, etc., we will use two asterisks around each command and variable name. We also enclose Stata commands and tags in sets of four tildes to display the content in plain text. Our modified text file is shown here:

```
-----begin example2.txt-----
Using Stata dynamic tags
=====

I am going to examine fuel efficiency using **auto.dta**. First, I load the
dataset:

~~~~
<<dd_do>>
sysuse auto, clear
<</dd_do>>
~~~~

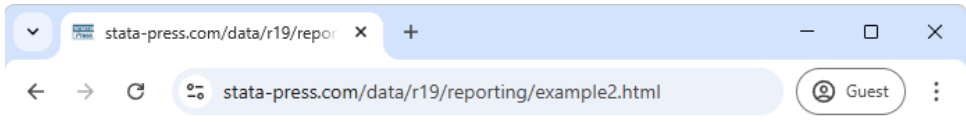
Now I **summarize mpg**, but I only display the output, not the command:

~~~~
<<dd_do: nocommands>>
summarize mpg
<</dd_do>>
~~~~
-----end example2.txt-----
```

We copy this file to our current working directory and then convert it to an HTML file with `dyndoc`:

```
. copy https://www.stata-press.com/data/r19/reporting/example2.txt .
. dyndoc example2.txt
```

This produces the following:



Using Stata dynamic tags

I am going to examine fuel efficiency using **auto.dta**. First, I load the dataset:

```
. sysuse auto, clear
(1978 Automobile Data)
```

Now I **summarize mpg**, but I only display the output, not the command:

Variable	Obs	Mean	Std. Dev.	Min	Max
mpg	74	21.2973	5.785503	12	41

We could have also included a graph or Stata expression in the HTML file; see [RPT] [dyndoc](#) for a more detailed example.

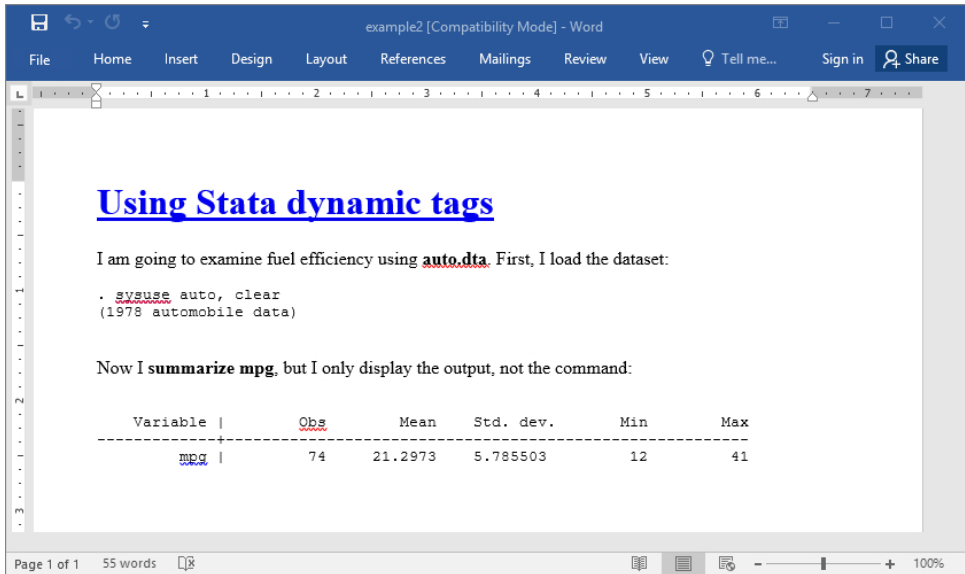


► Example 4: Create a Word document with Stata output

By default, [dyndoc](#) will create an HTML output file, but we could just as easily convert `example.txt` to a Word (`.docx`) document instead by specifying the `docx` option:

```
. dyndoc example2.txt, docx
```

This produces `example2.docx`, which looks like this:



Once you have created a Word document, you can also convert it to a PDF file; see [RPT] [docx2pdf](#).



The output files, `example2.html` and `example2.docx`, are available at <https://www.stata-press.com/data/r19/reporting/>.

Above, we used `dyndoc` to embed the output from commands and to process Markdown-formatted text when we created an HTML file and a Word document. `dyndoc` uses the `markdown` command to convert the Markdown-formatted text to the HTML format. `markdown` can also be used independently to create HTML files and Word documents when the source file does not contain Stata dynamic tags.

Also see

[RPT] [Dynamic tags](#) — Dynamic tags for text files

[RPT] [dyndoc](#) — Convert dynamic Markdown document to HTML or Word (.docx) document

[RPT] [dyntext](#) — Process Stata dynamic tags in text file

[RPT] [markdown](#) — Convert Markdown document to HTML file or Word (.docx) document

[Description](#)[Remarks](#)[Also see](#)

Description

Dynamic tags are instructions used by Stata's dynamic documents commands, [dyndoc](#) and [dyntext](#), to perform a certain action, such as run a block of Stata code, insert the result of a Stata expression in text, export a Stata graph to an image file, or include a link to the image file.

Remarks

Remarks are presented under the following headings:

[Descriptions of dynamic tags](#)

[Version control](#)

[Execute and include output from a block of Stata code](#)

[Include strings and values of scalar expressions in text](#)

[Include values of scalar expressions and formatted text in a .docx file](#)

[Export and include a Stata graph](#)

[Include a text file](#)

[Disable dynamic text processing](#)

[Process contents based on condition](#)

[Skip contents based on condition](#)

[Remove contents](#)

Descriptions of dynamic tags

Here is a list of available dynamic tags and a short description for each. The tag may be abbreviated; the minimum abbreviation is indicated by the underlined letters.

Dynamic tag	Description
<code>«dd_version»</code>	specify the minimum version required to convert the dynamic document
<code>«dd_do»</code>	execute a block of Stata code and optionally include its output
<code>«/dd_do»</code>	end <code>«dd_do»</code>
<code>«dd_display»</code>	include output of Stata expression as shown by Stata's display command
<code>«dd_docx_display»</code>	include output of Stata expression in a .docx file as shown by Stata's display command and format text within a block
<code>«dd_graph»</code>	export a Stata graph and include a link to the file
<code>«dd_ignore»</code>	disable processing of dynamic tags except <code>«dd_remove»</code>
<code>«/dd_ignore»</code>	end <code>«dd_ignore»</code>
<code>«dd_include»</code>	include the contents of a text file
<code>«dd_remove»</code>	remove the following text until <code>«/dd_remove»</code> is specified
<code>«/dd_remove»</code>	end <code>«dd_remove»</code>
<code>«dd_if»</code>	process text based on condition
<code>«dd_else»</code>	process text based on condition
<code>«dd_endif»</code>	end <code>«dd_if»</code> block
<code>«dd_skip_if»</code>	skip text based on condition
<code>«dd_skip_else»</code>	skip text based on condition
<code>«dd_skip_end»</code>	end <code>«dd_skip_if»</code> block

`«dd_docx_display»` is only for use with `putdocx` `textblock` commands in a do-file.

Some tags must start at the beginning of a line, and the text in the same line after the tag is simply ignored. Other tags can be written in the middle of a line. The following table lists the required position in text for all tags.

Dynamic tag	Description
«dd_version»	beginning of a line, recommended at the start of a file
«dd_do»	beginning of a line
«/dd_do»	beginning of a line
«dd_display»	within a line
«dd_docx_display»	within a line
«dd_graph»	within a line
«dd_ignore»	beginning of a line
«/dd_ignore»	beginning of a line
«dd_include»	beginning of a line
«dd_remove»	within a line
«/dd_remove»	within a line
«dd_if»	beginning of a line
«dd_else»	beginning of a line
«dd_endif»	beginning of a line
«dd_skip_if»	beginning of a line
«dd_skip_else»	beginning of a line
«dd_skip_end»	beginning of a line

Tags can have attributes. Attributes are modifiers of a tag’s behavior. Attributes can be repeated, and the last one will take effect. For example, if you specify «dd_do: *commands nocommands*», the commands will not be displayed because the attribute *nocommands* supersedes the previously specified attribute *commands*. This is useful when you experiment with the behavior of attributes for the best output. Some attributes have values; for example, *graphname()* requires the name of the graph to be exported. If a tag has only one attribute and that attribute requires a value, then the attribute name is omitted and only the value is required; for example, the *dd_version* tag is used as «dd_version: *an integer number*».

Version control

```
<<dd_version: version_number>>
```

The «dd_version» tag specifies the minimum version required to convert the source file. The version number is independent of Stata’s *version* command. The tag must be at the beginning of a new line. We recommend that the tag be placed at the beginning of the *srcfile*.

The current version, and the default, is 2, and it is introduced as of the release of Stata 16. The current version number is also stored in *c(dyndoc_version)*.

Execute and include output from a block of Stata code

```
<<dd_do: attribute>>
block of Stata code ...
<</dd_do>>
```

The «dd_do» tag runs the block of Stata code, replacing the lines between «dd_do» and «/dd_do» with Stata output. Both the start tag, «dd_do», and the end tag, «/dd_do», must be at the beginning of new lines.

<i>attribute</i>	Description
<code>quietly</code>	suppress all output
<code>nocommands</code>	suppress printing of command
<code>nooutput</code>	suppress command output
<code>noprompt</code>	suppress the dot prompt

Include strings and values of scalar expressions in text

```
<<dd_display: display_directive>>
```

The `<<dd_display>>` tag executes Stata's `display` command and then replaces the tag with its output. The tag cannot contain a line break or `»`. Use `> >` (with a space in between) instead if you need to include `»` in the *display_directive*.

The `<<dd_display>>` tag can be used multiple times inside a line of text. For example, say that we want to display the circumference of a circle of radius 1 up to the two digits after the decimal. Instead of computing the number and then copying and pasting the result into the text, we can write

```
2*1*<<dd_display:%4.2f c(pi)>> = <<dd_display:%4.2f 2*1*c(pi)>>
```

which produces

```
2*1*3.14 = 6.28
```

Include values of scalar expressions and formatted text in a .docx file

```
<<dd_docx_display text_options: display_directive>>
```

This tag includes expressions and formatted text within a block of text in a `.docx` file. It can only be used with text enclosed in `putdocx textblock` commands, as follows:

```
putdocx textblock begin
... text <<dd_docx_display directive>> text ...
putdocx textblock end
```

The `<<dd_docx_display>>` tag executes Stata's `display` command and then replaces the tag with its output. The output is formatted according to the *text_options* available with `putdocx text`. The tag cannot contain a line break or `»`. If you need to include `»` in the *display_directive*, use the symbols with a space in between (`> >`).

The `<<dd_docx_display>>` tag can be used multiple times inside a line of text. For example, say that we want to display the circumference of a circle with radius 1 up to the two digits after the decimal. Instead of computing the number and then copying and pasting the result into a block of text, we can write

```
putdocx textblock begin
2*1*<<dd_docx_display bold:%4.2f c(pi)>> = <<dd_docx_display bold:%4.2f 2*1*c(pi)>>
putdocx textblock end
```

which formats the value of π and the product in bold and produces the following in the `.docx` file being created.

```
2*1*3.14 = 6.28
```

For another example demonstrating the use of this dynamic tag, see [Working with blocks of text](#) in [RPT] [putdocx paragraph](#).

Export and include a Stata graph

```
<<dd_graph: attribute>>
```

The «dd_graph» tag exports a Stata graph and then includes a link to the exported image file in the target file.

<i>attribute</i>	Description
<u>saving</u> (<i>filename</i>)	export graph to <i>filename</i>
<u>replace</u>	replace the file if it already exists
<u>graphname</u> (<i>name</i>)	name of graph to be exported
<u>svg</u>	export graph as SVG
<u>png</u>	export graph as PNG
<u>pdf</u>	export graph as PDF
<u>eps</u>	export graph as EPS
<u>ps</u>	export graph as PS
<u>html</u>	output an HTML link
<u>markdown</u>	output a Markdown link; default is html
<u>pathonly</u>	output the path of the file; default is html
<u>alt</u> (<i>text</i>)	alternative text for the graph to be read by voice software; ignored if pathonly in effect
<u>height</u> (#)	height in pixels of the graph in HTML; ignored if markdown or pathonly in effect
<u>width</u> (#)	width in pixels of the graph in HTML; ignored if markdown or pathonly in effect
<u>relative</u>	use file path relative to the <i>targetfile</i> path specified in dyndoc or dyntext ; this is the default
<u>absolute</u>	use absolute path in the link; default is relative
<u>basepath</u> (<i>path</i>)	use <i>path</i> as base directory where graph files will be exported; default is the current working directory if it is not specified
<u>nourlencode</u>	do not encode the path to a percent-encoded URL; ignored if html or markdown in effect

If graphname(*name*) is not specified, the topmost graph is used. You can use the default name “Graph” to export the graph without the name.

For paths specified in the saving() or basepath() attributes, a single backslash (\) is interpreted as an escape character rather than as the directory separator character. When working on Windows, we recommend using a forward slash (/) as the directory separator character (for example, C:/mypath/myfile); otherwise, you must use a double backslash (for example, C:\\mypath\\myfile).

If saving(*filename*) is not specified, a filename will be constructed based on the graph name.

If none of .svg, .png, or .pdf is specified, the saving(*filename*) is checked first; if the name specified in saving(*filename*) has the extension of .svg, .png, or .pdf, then the graph will be exported in the format corresponding to the extension. For example, the dynamic tag

```
<<dd_graph:saving(gr1.png) graphname(gr1)>>
```

produces

```

```

Otherwise, the type `.svg` will be used as in

```
<<dd_graph:saving(gr1.pgg) graphname(gr1)>>
```

which produces

```

```

If `markdown` is specified, a Markdown link will be produced. For example, the dynamic tag

```
<<dd_graph:saving(gr1.svg) graphname(gr1) markdown>>
```

produces

```
![] (gr1.svg)
```

You may use `pathonly` if you want an HTML link with more attributes than `html` or `markdown` can provide or if you want to use the path in a different target file type such as \LaTeX .

By default, the path is outputted as a percent-encoded URL. For example, the dynamic tag

```
<<dd_graph:saving("gr 1.svg") graphname(gr1) pathonly>>
```

produces

```
gr%201.svg
```

You may use `nourlencode` to disable the encoding process as in

```
"<<dd_graph:saving("gr 1.svg") graphname(gr1) pathonly nourlencode>>"
```

which produces

```
"gr 1.svg"
```

The `<<dd_graph>` tag can be used inside a line of text.

Include a text file

```
<<dd_include: filename>>
```

The `<<dd_include>` tag replaces the tag with the contents of the specified text file. The text file is included as is. The tag must be at the beginning of a new line. The *filename* itself may contain Stata macros, but not the file contents.

Disable dynamic text processing

```
«dd_ignore» and «/dd_ignore»
```

The `«dd_ignore»` tag causes `dyntext` and `dyndoc` to ignore the dynamic tag processing, starting from the next line until the line right before a `«/dd_ignore»` tag. Both the beginning and ending tags must be at the beginning of a line. The only tag it does not affect is the `«dd_remove»` tag.

Process contents based on condition

```
<<dd_if: Stata expression>>
lines of text ...
<<dd_endif>>
```

or

```
<<dd_if: Stata expression>>
lines of text ...
<<dd_else>>
lines of text ...
<<dd_endif>>
```

«dd_if: *Stata expression*» evaluates the *Stata expression*; if it evaluates to true (anything but 0 or "0"), the lines before the next «dd_endif» are processed. If there is a «dd_else», the lines before «dd_else» are processed, and the lines between «dd_else» and «dd_endif» are skipped.

If the Stata expression evaluates to false (0 or "0"), the lines before the next «dd_endif» are skipped. If there is a «dd_else», the lines before «dd_else» are skipped, and the lines between «dd_else» and «dd_endif» are processed.

Skip contents based on condition

```
<<dd_skip_if: Stata expression>>
lines of text ...
<<dd_skip_end>>
```

or

```
<<dd_skip_if: Stata expression>>
lines of text ...
<<dd_skip_else>>
lines of text ...
<<dd_skip_end>>
```

«dd_skip_if: *Stata expression*» evaluates the *Stata expression*; if it evaluates to true (anything but 0), the lines before the next «dd_skip_end» are skipped. If there is a «dd_skip_else», the lines before «dd_skip_else» are skipped, and the lines between «dd_skip_else» and «dd_skip_end» are processed as usual.

If the Stata expression evaluates to false (0), the lines before the next «dd_skip_end» are not skipped. If there is a «dd_skip_else», the lines before «dd_skip_else» are not skipped, and the lines between «dd_skip_else» and «dd_skip_end» are skipped.

Remove contents

```
... <<dd_remove>>text to remove ...
lines of text to remove ...
text to remove ... </dd_remove>> ...
```

The «dd_remove» and «/dd_remove» tags remove all the contents between the two tags from the resulting target file. The tags can be used inside a line of text.

«dd_remove» is a postprocessing tag, which means it is processed after all other tags.

Also see

[RPT] **dyndoc** — Convert dynamic Markdown document to HTML or Word (.docx) document

[RPT] **dyntext** — Process Stata dynamic tags in text file

[RPT] **markdown** — Convert Markdown document to HTML file or Word (.docx) document

Description

`dyndoc` converts a dynamic Markdown document—a document containing both formatted text and Stata commands—to an HTML file or Word document. Stata processes the Markdown text and Stata dynamic tags (see [RPT] [Dynamic tags](#)) and creates the output file. Markdown is a simple markup language with a formatting syntax based on plain text. It is easily converted to an output format such as HTML. Stata dynamic tags allow Stata commands, output, and graphs to be interleaved with Markdown text.

If you want to convert a Markdown document without Stata dynamic tags to an HTML file or Word document, see [RPT] [markdown](#). If you want to convert a plain text file containing Stata dynamic tags to a plain text output file, see [RPT] [dyntext](#). If you want to convert an HTML file to a Word document, see [RPT] [html2docx](#).

Quick start

Convert text file `myfile.txt` with Stata dynamic tags and Markdown formatting to an HTML file with Stata output saved as `myfile.html`

```
dyndoc myfile.txt
```

Same as above, but save the HTML file as `mydoc.html`

```
dyndoc myfile.txt, saving(mydoc.html)
```

Same as above, and overwrite the existing `mydoc.html`

```
dyndoc myfile.txt, saving(mydoc.html) replace
```

Convert `myfile.txt` to a Word document saved as `myfile.docx`

```
dyndoc myfile.txt, docx
```

Syntax

`dyndoc srcfile [arguments] [, options]`

srcfile is a plain text file containing Markdown-formatted text and [Stata dynamic tags](#).

arguments are stored in the local macros ‘1’, ‘2’, and so on for use in *srcfile*; see [\[U\] 16.4.1 Argument passing](#).

You may enclose *srcfile* and *targetfile* in double quotes and must do so if they contain blanks or other special characters.

<i>options</i>	Description
<code>saving(<i>targetfile</i>)</code>	specify the target HTML file or Word (.docx) document to be saved
<code>replace</code>	replace the target HTML file or Word (.docx) document if it already exists
<code>hardwrap</code>	replace hard wraps (actual line breaks) with the tag in an HTML file or with line breaks in a Word (.docx) document
<code>nomsg</code>	suppress message with a link to <i>targetfile</i>
<code>nostop</code>	do not stop when an error occurs
<code>embedimage</code>	embed image files as Base64 binary data in the target HTML file
<code>docx</code>	output a Word (.docx) document instead of an HTML file

Options

`saving(targetfile)` specifies the target file to be saved. If `saving()` is not specified, the target filename is constructed using the source filename (*srcfile*) with the .html extension or with the .docx extension if `docx` is specified. If the *targetfile* has the .docx extension, the `docx` option is assumed even if it is not specified.

`replace` specifies that the target file be replaced if it already exists.

`hardwrap` specifies that hard wraps (actual line breaks) in the Markdown document be replaced with the
 tag in the HTML file or with a line break in the Word (.docx) document if the `docx` option is specified.

`nomsg` suppresses the message that contains a link to the target file.

`nostop` allows the document to continue being processed even if an error occurs. By default, `dyndoc` stops processing the document if an error occurs. The error can be caused by either a malformed dynamic tag or Stata code executed within the tag.

`embedimage` allows image files to be embedded as data URI (Base64-encoded binary data) in the HTML file. The supported image file types are portable network graphics (.png), JPEG (.jpg), tagged image file format (.tif), and graphics interchange format (.gif). This option cannot be used to embed SVG and PDF image file types.

The image must be specified in a Markdown link; you cannot embed images specified by URLs. This option is ignored if `docx` is specified.

`docx` specifies that the target file be saved in Microsoft Word (.docx) format. If the target file has the .docx extension, the `docx` option is implied. The conversion process consists of first producing an HTML file and then using [html2docx](#) to produce the final Word document.

Remarks and examples

A dynamic document contains both static narrative and dynamic tags. Dynamic tags are instructions for dyndoc to perform a certain action, such as run a block of Stata code, insert the result of a Stata expression in text, export a Stata graph to an image file, or include a link to the image file. Any changes in the data or in Stata will change the output as the document is created. The main advantages of using dynamic documents are

- results in the document come from executing commands instead of being copied from Stata and pasted into the document;
- no need to maintain parallel do-files; and
- any changes in data or in Stata are reflected in the final document when it is created.

► Example 1: Converting a dynamic document to an HTML file

Let us consider an example. Suppose that we have `dyndoc_ex.txt` with the following Markdown-formatted text that includes [Stata dynamic tags](#).

```
<<dd_version: 2>>
<<dd_include: header.txt >>
```

Using Stata dynamic tags in a text file with the dyndoc command

```
=====

Let us consider an example where we study the mpg and weight variables
in auto.dta. In our examples below, we will first write the commands so
that they will be displayed in our target HTML file. Then, we will write the
commands so that Stata will process the Stata dynamic tags, displaying the
results of the Stata commands in the target HTML file.
```

We first use the **sysuse** command to load the dataset and then describe
the data using the **describe** command.

```
~~~~~
<<dd_ignore>>
<<dd_do>>
sysuse auto, clear
describe
<</dd_do>>
<</dd_ignore>>
~~~~~
```

This produces the following Stata results:

```
~~~~~
<<dd_do>>
sysuse auto, clear
describe
<</dd_do>>
~~~~~
```

Now, we want to check if **mpg** is always greater than 0 and less than 100.
We use the **assert** command to perform the check. In this case, we do not
want to include any output in the target HTML file, so we use the **quietly**
attribute to modify the behavior of the **dd_do** Stata dynamic tag.

```
~~~~~
<<dd_ignore>>
<<dd_do:quietly>>
assert mpg > 0 & mpg < 100
<</dd_do>>
<</dd_ignore>>

<<dd_do:quietly>>
assert mpg > 0 & mpg < 100
<</dd_do>>

~~~~~
```

If the data do not satisfy the conditions, **dyndoc** will fail with an error message, which will occur if we run the same **assert** command in a do-file.

Next, we want to summarize the **weight** variable:

```
~~~~~
<<dd_ignore>>
<<dd_do>>
summarize weight
<</dd_do>>
<</dd_ignore>>

~~~~~
```

This produces the following in the target HTML file:

```
~~~~~
<<dd_do>>
summarize weight
<</dd_do>>

~~~~~
```

We want to use the minimum and maximum values of **weight** in a sentence. Instead of copying and pasting the numbers from the **summarize** output, we can use the **dd_display** Stata dynamic tag with the **r(min)** and **r(max)** stored results:

```
~~~~~
<<dd_ignore>>
The variable weight has minimum value <<dd_display: %4.2f 'r(min)'\>> and
has maximum value <<dd_display: %4.2f 'r(max)'\>>.
<</dd_ignore>>

~~~~~
```

This produces the following in the target HTML file:

```
~~~~~

> The variable weight has minimum value <<dd_display: %4.2f 'r(min)'\>>
and has maximum value <<dd_display: %4.2f 'r(max)'\>>.

~~~~~
```

The **dd_display** dynamic tag uses the **display** command to evaluate expressions. It can be used as a calculator. For example, if we want to include the **range = max - min** in a sentence, instead of calculating the number and then copying and pasting it, we can use

```
~~~~~
<<dd_ignore>>
```

```
The variable weight has range <<dd_display: %4.2f 'r(max)''-r(min)''>>.
<</dd_ignore>>
~~~~
```

which produces the following in the target HTML file:

```
~~~~
```

```
> The variable weight has range <<dd_display: %4.2f 'r(max)''-r(min)''>>.
```

```
~~~~
```

Now, we want to graph **mpg** and **weight** using a scatterplot. We use the **dd_do** tag with the **nooutput** attribute to generate the scatterplot first. The **nooutput** attribute leaves the command in the output only,

```
~~~~
```

```
<<dd_ignore>>
<<dd_do:nooutput>>
scatter mpg weight, mcolor(blue%50)
<</dd_do>>
<</dd_ignore>>
~~~~
```

which generates a scatterplot of **mpg** and **weight** with 50% opacity color markers.

```
~~~~
```

```
<<dd_do:nooutput>>
scatter mpg weight, mcolor(blue%50)
<</dd_do>>
~~~~
```

Now, we want to export the graph to a file and include an image link to the file.

```
~~~~
```

```
<<dd_ignore>>
<<dd_graph: sav("graph.svg") alt("scatter mpg weight") replace height(400)>>
<</dd_ignore>>
~~~~
```

This produces a graph of 400 pixels high.

```
<<dd_graph: sav("graph.svg") alt("scatter mpg weight") replace height(400)>>
```

end dyndoc_ex.txt

□ Technical note

We use four tildes in a row, ~~~~, in our source file around parts of the document that we want to appear in plain text, such as Stata commands and output. Without the ~~~~, Stata's output would be interpreted as HTML in the final document and would not look as it should. This applies regardless of whether we are creating an HTML file or a Word document.

□

You will notice that we used the «dd_include» dynamic tag to include the `header.txt` file. The `header.txt` file contains HTML code to include at the top of our target HTML file. It refers to the `stmarkdown.css` file, which is a stylesheet that defines how the HTML document is to be formatted. This formatting would also apply if we were creating a Word document. We can copy these files and `dyndoc_ex.txt` to our working directory by typing

```
. copy https://www.stata-press.com/data/r19/reporting/header.txt .  
. copy https://www.stata-press.com/data/r19/reporting/stmarkdown.css .  
. copy https://www.stata-press.com/data/r19/reporting/dyndoc_ex.txt .
```

To generate the target HTML file in Stata, we type

```
. dyndoc dyndoc_ex.txt
```

The HTML file `dyndoc_ex.html` is saved. Here is a portion of this file:

stata-press.com/data/r19/repo

+

stata-press.com/data/r19/reporting/dyndoc_ex.html

Guest

Using Stata dynamic tags in a text file with the dyndoc command

Let us consider an example where we study the **mpg** and **weight** variables in **auto.dta**. In our examples below, we will first write the commands so that they will be displayed in our target HTML file. Then, we will write the commands so that Stata will process the Stata dynamic tags, displaying the results of the Stata commands in the target HTML file.

We first use the **sysuse** command to load the dataset and then describe the data using the **describe** command.

```
<<dd_do>>
sysuse auto, clear
describe
<</dd_do>>
```

This produces the following Stata results:

```
. sysuse auto, clear
(1978 automobile data)

. describe

Contains data from /usr/local/stata19/ado/base/a/auto.dta
Observations:      74      1978 automobile data
Variables:         12      13 Apr 2024 17:45
                        (_dta has notes)
```

Variable	Storage	Display	Value	
name	type	format	label	Variable label
make	str18	%-18s		Make and model
price	int	%8.0gc		Price
mpg	int	%8.0g		Mileage (mpg)
rep78	int	%8.0g		Repair record 1978
headroom	float	%6.1f		Headroom (in.)
trunk	int	%8.0g		Trunk space (cu. ft.)
weight	int	%8.0gc		Weight (lbs.)
length	int	%8.0g		Length (in.)
turn	int	%8.0g		Turn circle (ft.)
displacement	int	%8.0g		Displacement (cu. in.)
gear_ratio	float	%6.2f		Gear ratio
foreign	byte	%8.0g	origin	Car origin

Sorted by: foreign

You can see the whole file at https://www.stata-press.com/data/r19/reporting/dyndoc_ex.html.

□ Technical note

Because `quietly` and `capture` suppress the results of the command from being produced, you should not use these prefix commands with Stata code to be converted by dyndoc.



➤ Example 2: Converting a dynamic document to a Word document

Same as above, we could easily create a Word document from the dynamic text file used in [example 1](#) by specifying the `docx` option.

```
. dyndoc dyndoc_ex.txt, docx
```

This creates `dyndoc_ex.docx`, shown below:

dyndoc_ex.docx - Word

File Home Insert Design Layout References Mailings Review View Help Tell me

Using Stata dynamic tags in a text file with the dyndoc command

Let us consider an example where we study the **mpg** and **weight** variables in **auto.dta**. In our examples below, we will first write the commands so that they will be displayed in our target HTML file. Then, we will write the commands so that Stata will process the Stata dynamic tags, displaying the results of the Stata commands in the target HTML file.

We first use the **sysuse** command to load the dataset and then describe the data using the **describe** command.

```
<<dd_do>>
sysuse auto, clear
describe
<</dd_do>>
```

This produces the following Stata results:

```
. sysuse auto, clear
(1978 automobile data)

. describe

Contains data from /usr/local/stata19/ado/base/a/auto.dta
Observations:      74      1978 automobile data
Variables:         12      13 Apr 2024 17:45
                        (_dta has notes)
```

Variable name	Storage type	Display format	Value label	Variable label
make	str18	%-18s		Make and model
price	int	%8.0gc		Price
mpg	int	%8.0g		Mileage (mpg)
rep78	int	%8.0g		Repair record 1978
headroom	float	%6.1f		Headroom (in.)
trunk	int	%8.0g		Trunk space (cu. ft.)
weight	int	%8.0gc		Weight (lbs.)
length	int	%8.0g		Length (in.)
turn	int	%8.0g		Turn circle (ft.)
displacement	int	%8.0g		Displacement (cu. in.)
gear_ratio	float	%6.2f		Gear ratio

Page 1 of 3 596 words 100%

Specifying arguments in dyndoc allows even more flexibility. For instance, `dyndoc_ex.txt` a b c passes a, b, and c in local macros '1', '2', and '3', which can be used in the text file. Below, we demonstrate how to use arguments with dyndoc to create multiple HTML files from a single text file.

► Example 3: Specifying arguments

Suppose that we create histograms on a regular basis. To reduce our workload, we create a general text file that creates histograms for pairs of variables, which we specify as arguments to dyndoc. This way we do not create a new text file every time we create new histograms. To generalize the text file, we refer to macro 1 when loading the dataset, so that we may create histograms using any dataset we specify. We refer to macros 2 and 3 in the `<<dd_do>>` tags, instead of referring to actual variable names, because these will change. We use the `<<dd_display>>` dynamic tag to display the contents of the macros in the output HTML file, while suppressing the command lines with the `nocommands` attribute. Our text file is shown below:

```

<<dd_version: 2>>

The distribution of <<dd_display: "2">> and <<dd_display: "3">>
=====
Let's look at the histograms for <<dd_display: "2">> and <<dd_display: "3">>.

~~~~
. use <<dd_display: "1">>, clear
~~~~

~~~~
<<dd_do: nocommands>>
use '1'.dta, clear
<</dd_do>>
~~~~

~~~~
. histogram <<dd_display: "2">>, freq
~~~~
~~~~
<<dd_do: nocommands>>
histogram '2', freq
<</dd_do>>
~~~~
<<dd_graph>>

~~~~
. histogram <<dd_display: "3">>, freq
~~~~
~~~~
<<dd_do: nocommands>>
histogram '3', freq
<</dd_do>>
~~~~
<<dd_graph>>

```

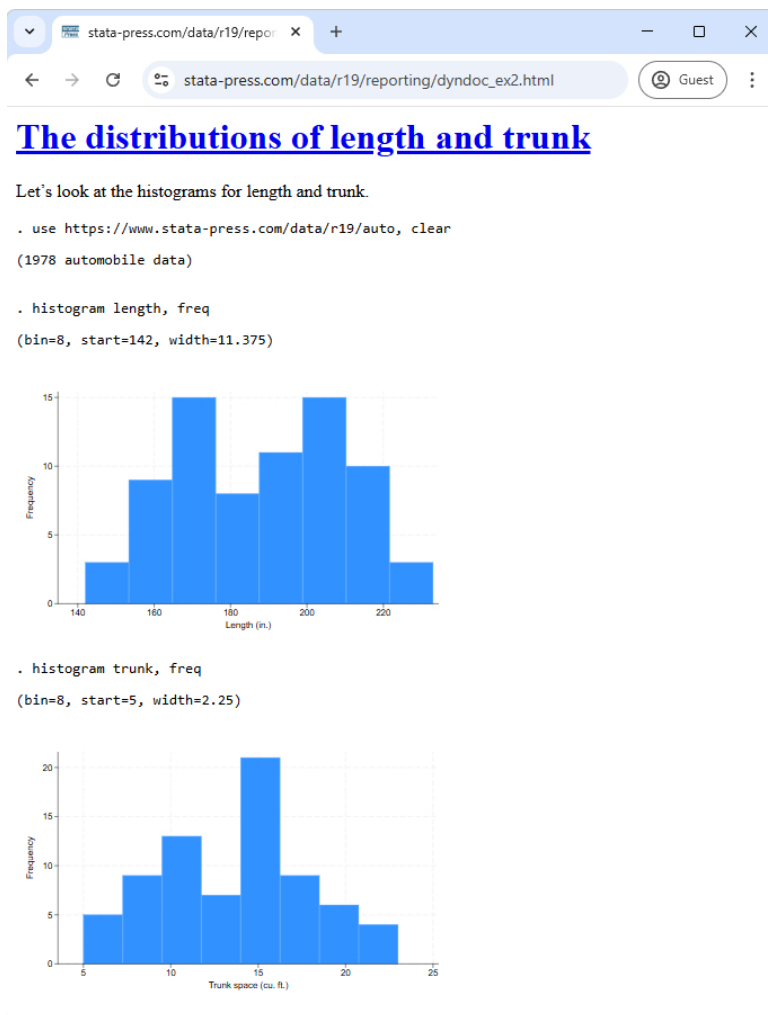
You can copy this file to your working directory by typing

```
. copy https://www.stata-press.com/data/r19/reporting/dyndoc_ex2.txt .
```

Now we issue dyndoc with three arguments: a link to `auto.dta` followed by variable names `length` and `trunk`.

```
. dyndoc dyndoc_ex2.txt "https://www.stata-press.com/data/r19/auto" length trunk
```

This creates `dyndoc_ex2.html`, shown below:



We optionally displayed the information on bin size, width, and starting values that `histogram` reports, but we could suppress that as well by specifying the `quietly` attribute instead.

If we want to create histograms with other variables, we do not have to edit our text file. We can simply issue dyndoc with the corresponding dataset and variable names to create this file. This is as easy as

```
. dyndoc dyndoc_ex2.txt "https://www.stata-press.com/data/r19/auto" price weight,  
> replace
```

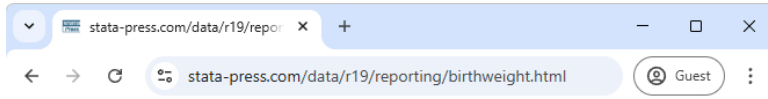
which produces the following:



Next we use the data from [Hosmer, Lemeshow, and Sturdivant \(2013, 24\)](#) to create histograms for mother's age and the baby's birthweight, bwt.

```
. dyndoc dyndoc_ex2.txt "https://www.stata-press.com/data/r19/lbw" age bwt,  
> saving(birthweight.html)
```

This produces the following:

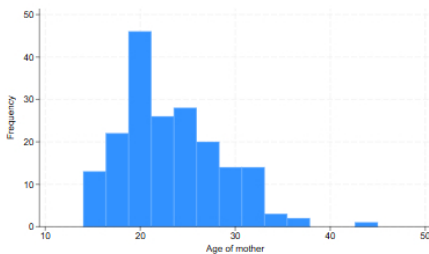


The distributions of age and bwt

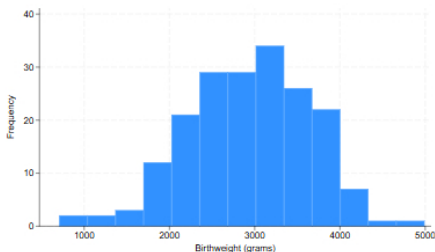
Let's look at the histograms for age and bwt.

```
. use https://www.stata-press.com/data/r19/lbw, clear  
(Hosmer & Lemeshow data)
```

```
. histogram age, freq  
(bin=13, start=14, width=2.3846154)
```



```
. histogram bwt, freq  
(bin=13, start=709, width=329.30769)
```



Whether you are creating the same graphics for different sets of variables or you need to create the same reports using different datasets, you can use arguments with dyndoc to streamline your work process.



References

- Gillman, M. S. 2018. [Some commands to help produce Rich Text Files from Stata](#). *Stata Journal* 18: 197–205.
- Hosmer, D. W., Jr., S. A. Lemeshow, and R. X. Sturdivant. 2013. *Applied Logistic Regression*. 3rd ed. Hoboken, NJ: Wiley.
- Jakubowski, M., and A. Pokropek. 2019. [piaactools: A program for data analysis with PIAAC data](#). *Stata Journal* 19: 112–128.
- Jann, B. 2017. [Creating HTML or Markdown documents from within Stata using webdoc](#). *Stata Journal* 17: 3–38.
- Rodríguez, G. 2017. [Literate data analysis with Stata and Markdown](#). *Stata Journal* 17: 600–618.

Also see

- [\[RPT\] Dynamic tags](#) — Dynamic tags for text files
- [\[RPT\] dyntext](#) — Process Stata dynamic tags in text file
- [\[RPT\] markdown](#) — Convert Markdown document to HTML file or Word (.docx) document

Description

dyntext converts a dynamic text file—a file containing both plain text and Stata commands—to an output file in text format. Stata processes the Stata dynamic tags (see [RPT] **Dynamic tags**) in the dynamic text file and creates the output text file.

If you want to convert a dynamic text file to an HTML or Word (.docx) document, see [RPT] **dyndoc**. If you want to convert a Markdown document to an HTML or Word document, see [RPT] **markdown**.

Quick start

Convert text file myfile.txt with Stata dynamic tags to a text file output.txt with Stata output
dyntext myfile.txt, saving(output.txt)

Same as above, and overwrite the existing output.txt
dyntext myfile.txt, saving(output.txt) replace

Syntax

```
dyntext srcfile [arguments], saving(targetfile) [options]
```

srcfile is a plain text file containing **Stata dynamic tags**. *srcfile* and *targetfile* may be any text format (.txt, .html, .do).

arguments are stored in the local macros '1', '2', and so on for use in *srcfile*; see [U] **16.4.1 Argument passing**.

You may enclose *srcfile* and *targetfile* in double quotes and must do so if they contain blanks or other special characters.

<i>options</i>	Description
* <u>s</u> aving(<i>targetfile</i>)	specify the target file to be saved
<u>r</u> eplace	replaces the target file if it already exists
<u>n</u> oremove	do not process «dd_remove» and «/dd_remove» dynamic tags
<u>n</u> ostop	do not stop when an error occurs

*saving(*targetfile*) is required.

Options

`saving(targetfile)` specifies the target file to be saved. `saving()` is required.

`replace` specifies that the target file be replaced if it already exists.

`noremove` specifies that `<<dd_remove>>` and `<</dd_remove>>` tags not be processed.

`nostop` allows the document to continue being processed even if an error occurs. By default, `dyntext` stops processing the document if an error occurs. The error can be caused either by a malformed dynamic tag or by executing Stata code within the tag.

Remarks and examples

A dynamic document contains both static narrative and dynamic tags. Dynamic tags are instructions for `dyntext` to perform a certain action, such as run a block of Stata code, insert the result of a Stata expression in text, export a Stata graph to an image file, or include a link to the image file. Any changes in the data or in Stata will change the output as the document is created. The main advantages of using dynamic documents are

- results in the document come from executing commands instead of being copied from Stata and pasted into the document;
- no need to maintain parallel do-files; and
- any changes in data or in Stata are reflected in the final document when it is created.

► Example 1

Let's consider an example. Suppose that we have `dyntext_ex.txt` with the following text that includes [Stata dynamic tags](#). Because we are writing in plain text, we use `-` to indicate Stata command names, variable names, etc.

```

--begin dyntext_ex.txt--
<<dd_version: 2>>

Using Stata dynamic tags in a text file with the -dyntext- command
=====

Let us consider an example where we study the -mpg- and -weight- variables in
-auto.dta-. In our examples below, we will first write the commands so that
they will be displayed in our output text file. Then, we will write the
commands so that Stata will process the Stata dynamic tags, displaying the
results of the Stata commands in the output text file.

We first use the -sysuse- command to load the dataset and then describe
the data using the -describe- command.

<<dd_ignore>>
<<dd_do>>
sysuse auto, clear
describe
<</dd_do>>
<</dd_ignore>>

This produces the following Stata results:

<<dd_do>>
sysuse auto, clear
describe
<</dd_do>>

Now, we want to check if -mpg- is always greater than 0 and less than 100.
```

We use the `-assert-` command to perform the check. In this case, we do not want to include any output in the output text file, so we use the `-quietly-` attribute to modify the behavior of the `-dd_do-` Stata dynamic tag.

```
<<dd_ignore>>
<<dd_do:quietly>>
assert mpg > 0 & mpg < 100
<</dd_do>>
<</dd_ignore>>

<<dd_do:quietly>>
assert mpg > 0 & mpg < 100
<</dd_do>>
```

If the data do not satisfy the conditions, `-dyntext-` will fail with an error message, which will occur if we run the same `-assert-` command in a do-file.

Next, we want to summarize the `-weight-` variable:

```
<<dd_ignore>>
<<dd_do>>
summarize weight
<</dd_do>>
<</dd_ignore>>
```

This produces the following in the output text file:

```
<<dd_do>>
summarize weight
<</dd_do>>
```

We want to use the minimum and maximum values of `-weight-` in a sentence. Instead of copying and pasting the numbers from the `-summarize-` output, we can use the `-dd_display-` Stata dynamic tag with the `-r(min)-` and `-r(max)-` stored results

```
<<dd_ignore>>
The variable weight has minimum value <<dd_display: %4.2f 'r(min)''>> and
has maximum value <<dd_display: %4.2f 'r(max)''>>.
<</dd_ignore>>
```

which produces the following in the output text file:

```
> The variable weight has minimum value <<dd_display: %4.2f 'r(min)''>>
and has maximum value <<dd_display: %4.2f 'r(max)''>>.
```

The `-dd_display-` dynamic tag uses Stata's `-display-` command to evaluate expressions. It can be used as a calculator. For example, if we want to include the range = max - min in a sentence, instead of calculating the number and then copying and pasting it, we can use

```
<<dd_ignore>>
The variable weight has range <<dd_display: %4.2f 'r(max)''-'r(min)''>>.
<</dd_ignore>>
```

which produces the following in the output text file:

```
> The variable weight has range <<dd_display: %4.2f 'r(max)''-'r(min)''>>.
```

end dyntext_ex.txt

We can copy this file to our working directory by typing

```
. copy https://www.stata-press.com/data/r19/reporting/dyntext_ex.txt .
```

To generate the output file in Stata, we then type

```
. dyntext dyntext_ex.txt, saving(dyntext_res.txt)
```

which produces the following:

 begin dyntext_res.txt

 Using Stata dynamic tags in a text file with the -dyntext- command

Let us consider an example where we study the -mpg- and -weight- variables in -auto.dta-. In our examples below, we will first write the commands so that they will be displayed in our output text file. Then, we will write the commands so that Stata will process the Stata dynamic tags, displaying the results of the Stata commands in the output text file.

We first use the -sysuse- command to load the dataset and then describe the data using the -describe- command.

```
<<dd_do>>
sysuse auto, clear
describe
<</dd_do>>
```

This produces the following Stata results:

```
. sysuse auto, clear
(1978 automobile data)
```

```
. describe
```

```
Contains data from C:\Program Files\Stata19\ado\base/a/auto.dta
Observations:      74                1978 automobile data
Variables:         12                13 Apr 2024 17:45
                                   (_dta has notes)
```

Variable name	Storage type	Display format	Value label	Variable label
make	str18	%-18s		Make and model
price	int	%8.0gc		Price
mpg	int	%8.0g		Mileage (mpg)
rep78	int	%8.0g		Repair record 1978
headroom	float	%6.1f		Headroom (in.)
trunk	int	%8.0g		Trunk space (cu. ft.)
weight	int	%8.0gc		Weight (lbs.)
length	int	%8.0g		Length (in.)
turn	int	%8.0g		Turn circle (ft.)
displacement	int	%8.0g		Displacement (cu. in.)
gear_ratio	float	%6.2f		Gear ratio
foreign	byte	%8.0g	origin	Car origin

Sorted by: foreign

Now, we want to check if -mpg- is always greater than 0 and less than 100. We use the -assert- command to perform the check. In this case, we do not want to include any output in the output text file, so we use the -quietly- attribute to modify the behavior of the -dd_do- Stata dynamic tag.

```
<<dd_do:quietly>>
assert mpg > 0 & mpg < 100
<</dd_do>>
```

If the data do not satisfy the conditions, -dyntext- will fail with an error message, which will occur if we run the same -assert- command in a do-file.

Next, we want to summarize the -weight- variable:

```
<<dd_do>>
summarize weight
<</dd_do>>
```


This produces the following in the output text file:

```
. summarize weight
```

Variable	Obs	Mean	Std. dev.	Min	Max
weight	74	3019.459	777.1936	1760	4840

We want to use the minimum and maximum values of `-weight-` in a sentence. Instead of copying and pasting the numbers from the `-summarize-` output, we can use the `-dd_display-` Stata dynamic tag with the `-r(min)-` and `-r(max)-` stored results

The variable `weight` has minimum value `<<dd_display: %4.2f 'r(min)'\>>` and has maximum value `<<dd_display: %4.2f 'r(max)'\>>`.

which produces the following in the output text file:

```
> The variable weight has minimum value 1760.00
and has maximum value 4840.00.
```

The `-dd_display-` dynamic tag uses Stata's `-display-` command to evaluate expressions. It can be used as a calculator. For example, if we want to include the range = max - min in a sentence, instead of calculating the number and then copying and pasting it, we can use

The variable `weight` has range `<<dd_display: %4.2f 'r(max)'\-'r(min)'\>>`.

which produces the following in the output text file:

```
> The variable weight has range 3080.00.
```

end dyntext_res.txt



□ Technical note

Because [quietly](#) and [capture](#) suppress the results of the command from being produced, you should not use these prefix commands with Stata code to be converted by `dyntext`.



References

- Gillman, M. S. 2018. [Some commands to help produce Rich Text Files from Stata](#). *Stata Journal* 18: 197–205.
- Jann, B. 2016. [Creating L^AT_EX documents from within Stata using texdoc](#). *Stata Journal* 16: 245–263.

Also see

- [RPT] [Dynamic documents intro](#) — Introduction to dynamic documents
- [RPT] [Dynamic tags](#) — Dynamic tags for text files
- [RPT] [dyndoc](#) — Convert dynamic Markdown document to HTML or Word (.docx) document
- [RPT] [markdown](#) — Convert Markdown document to HTML file or Word (.docx) document

Description

html2docx converts an HTML file to a Word (.docx) document. The HTML file can be either a file on the local disk or a URL on a remote website.

Quick start

Convert HTML file myfile.html to a Word document saved as myfile.docx

```
html2docx myfile
```

Same as above, but save the Word document as mydoc.docx

```
html2docx myfile, saving(mydoc)
```

Same as above, and overwrite the existing mydoc.docx

```
html2docx myfile, saving(mydoc) replace
```

Syntax

```
html2docx srcfile [ , options ]
```

srcfile is an HTML file, either a local file or a URL. If *srcfile* is specified without an extension, .html is assumed. If *srcfile* contains embedded spaces or other special characters, enclose it in double quotes.

<i>options</i>	Description
<code>saving(<i>targetfile</i>)</code>	specify the target Word (.docx) document to be saved
<code>replace</code>	replace the target Word (.docx) document if it already exists
<code>nomsg</code>	suppress message with link to <i>targetfile</i>
<code>base(<i>string</i>)</code>	specify the base directory or base URL for relative links in <i>srcfile</i>

Options

`saving(targetfile)` specifies the target Word (.docx) document file to be saved. If *targetfile* is specified without an extension, .docx is assumed. If *targetfile* contains embedded spaces or other special characters, enclose it in double quotes. If `saving()` is not specified, the target filename is constructed using the source filename (*srcfile*) with the .docx extension. `saving()` is required if the *srcfile* is a URL.

`replace` specifies that the target Word (.docx) document be replaced if it already exists.

`nomsg` suppresses the message that contains a link to the target file.

`base(string)` specifies the base directory or the base URL for the relative links in the *srcfile*.

Remarks and examples

html2docx converts HTML files to Word (.docx) documents. It attempts to preserve the styles of various HTML elements in the .docx file. However, for some HTML elements, there is no direct translation for a .docx file. For instance, an apostrophe in an HTML file may be replaced with another character in the .docx file. Thus, your target Word document may require some cleaning after the html2docx conversion.

html2docx expects a valid HTML file—one that contains essential HTML elements such as `<!DOCTYPE html>`, `<html>`, `<head>`, and `<body>`. If the HTML file is not valid, html2docx will go through a tidying process to attempt to make it valid. html2docx will produce an error message if this tidying process fails. You may check whether an HTML file is valid by using the W3C online Markup Validation Service at https://validator.w3.org/#validate_by_upload+with_options.

If you are working with a Markdown-formatted text file, you can convert this file directly to a Word document by specifying the docx option with markdown; see [RPT] **markdown**. Similarly, you can use dyndoc to convert a text file with Stata commands and Markdown-formatted text to a Word document with Stata output; see [RPT] **dyndoc**.

► Example 1: Converting an HTML file to a Word document

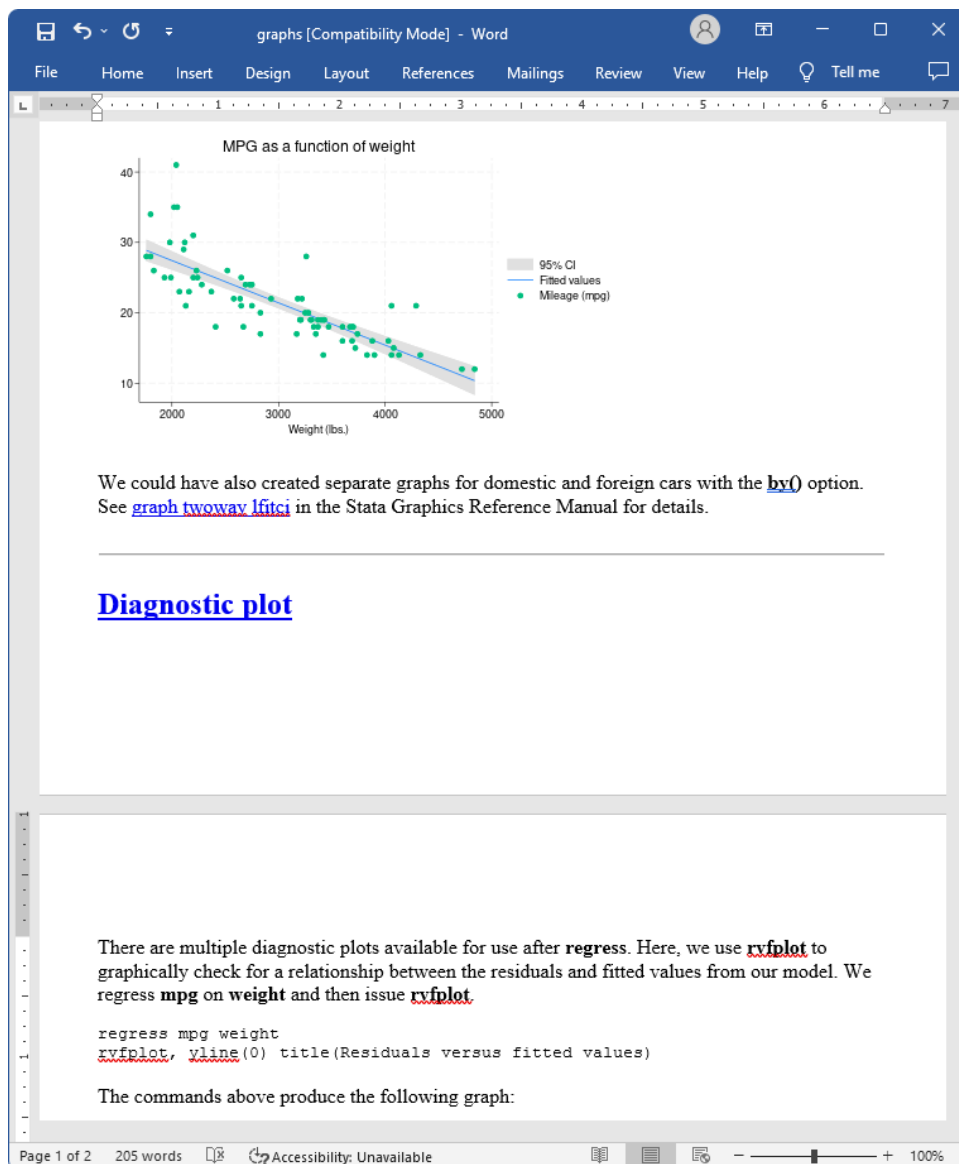
We have an HTML file, `graphs.html`, that includes some Stata graphs. You can copy this file to your current working directory by typing

```
. copy https://www.stata-press.com/data/r19/reporting/graphs.html .
```

To convert `graphs.html` to a Word document, we type

```
. html2docx graphs.html
```

The file is saved as `graphs.docx`. Here is a portion of this file:



We display the second page of the document to show both text and images found in the Word document. You can see the whole file at <https://www.stata-press.com/data/r19/reporting/graphs.docx>.

Also see

[RPT] [docx2pdf](#) — Convert a Word (.docx) document to a PDF file

[RPT] [dyndoc](#) — Convert dynamic Markdown document to HTML or Word (.docx) document

[RPT] [markdown](#) — Convert Markdown document to HTML file or Word (.docx) document

[RPT] [putdocx intro](#) — Introduction to generating Office Open XML (.docx) files

[Description](#)

[Remarks and examples](#)

[Quick start](#)

[References](#)

[Syntax](#)

[Also see](#)

[Options](#)

Description

markdown converts a Markdown document to an HTML file or a Word document.

Quick start

Convert Markdown document `myfile.txt` to an HTML file saved as `myfile.html`

```
markdown myfile.txt, saving(myfile.html)
```

Same as above, and overwrite the existing `myfile.html`

```
markdown myfile.txt, saving(myfile.html) replace
```

Convert Markdown document `myfile.txt` to a Word document saved as `myfile.docx`

```
markdown myfile.txt, saving(myfile.docx) docx
```

Syntax

```
markdown srcfile, saving(targetfile) [options]
```

srcfile is the Markdown document to be converted.

You may enclose *srcfile* and *targetfile* in double quotes and must do so if they contain spaces or special characters.

<i>options</i>	Description
* <u>saving</u> (<i>targetfile</i>) <u>replace</u>	HTML file or Word (.docx) document to be saved replace the target HTML file or Word (.docx) document if it already exists
hardwrap	replace hard wraps (actual line breaks) with the tag in an HTML file or with line breaks in a Word (.docx) document
nomsg	suppress message with a link to <i>targetfile</i>
embedimage	embed image files as Base64 binary data in the target HTML file
basedir(<i>string</i>)	specify the base directory for relative links in <i>srcfile</i>
docx	output a Word .docx document instead of an HTML file

* saving(*targetfile*) is required.

Options

`saving(targetfile)` specifies the target file to be saved. If the *targetfile* has the .docx extension, the docx option is assumed even if it is not specified. `saving()` is required.

`replace` specifies that the target file be replaced if it already exists.

`hardwrap` specifies that hard wraps (actual line breaks) in the Markdown document be replaced with the `
` tag in the HTML file or with a line break in the Word (.docx) file if the docx option is specified.

`nomsg` suppresses the message that contains a link to the target file.

`embedimage` allows image files to be embedded as data URI (Base64-encoded binary data) in the HTML file. The supported image file types are portable network graphics (.png), JPEG (.jpg), tagged image file format (.tif), and graphics interchange format (.gif). This option cannot be used to embed SVG and PDF image file types.

The image must be specified in a Markdown link; you cannot embed images specified by URLs. This option is ignored if docx is specified.

`basedir(string)` specifies the base directory for the relative links in the *srcfile*. This option only applies when specifying either the docx option or the embedimage option; otherwise, this option is ignored.

`docx` specifies that the target file be saved in Microsoft Word (.docx) format. If the target file has the .docx extension, the docx option is implied. The conversion process consists of first producing an HTML file and then using [html2docx](#) to produce the final Word document.

Remarks and examples

markdown converts a Markdown document to an HTML file or a Word (.docx) document. A Markdown document is written using an easy-to-read, plain text, lightweight markup language. For a detailed discussion and the syntax of Markdown, see the [Markdown Wikipedia page](#).

Stata uses Flexmark's Pegdown emulation as its default Markdown document processing engine. For information on Pegdown's flavor of Markdown, see the [Pegdown GitHub page](#).

See [\[RPT\] dyndoc](#) and [\[RPT\] dyntext](#) for a full description of Stata's dynamic document-generation commands. markdown is used by dyndoc but may also be used directly by programmers.

If you are not familiar with Markdown, we recommend that you first review [example 1](#) below for a brief introduction to this markup language. If you have used Markdown before, see [example 2](#) for details on embedding Stata graphs in your HTML file.

► Example 1: Create a basic HTML file with text

Suppose that we want to create a webpage with tips on working with Stata. We have created `markdown1.txt`, shown below, using Markdown-formatted text. We include comments to describe the formatting.

 begin markdown1.txt

```
Tips on working in Stata
=====

<!-- To create a heading, we underline text with equal signs. -->
<!-- Text enclosed in these arrows will be ignored. -->
This webpage will provide useful tips on working with Stata.
---

<!-- The three dashes above create a horizontal line. -->
## Working with strings
<!-- We create a sub-heading with two pound signs. -->
We begin by demonstrating when to use destring and encode.
<!-- We use two asterisks around each word we want to format as bold. -->
You should only use destring if a variable contains numeric values.
For example, in the following dataset income is stored as a string:
'''
webuse destring1, clear
destring income, replace
'''

<!-- We use three back-ticks for blocks of code. -->
```

 end markdown1.txt

You can copy this file into your working directory by typing

```
. copy https://www.stata-press.com/data/r19/reporting/markdown1.txt .
```

To convert this text file to an HTML file, we type the following command:

```
. markdown markdown1.txt, saving(tips.html)
```

This creates `tips.html`, which looks like the following:



`tips.html` is also available at <https://www.stata-press.com/data/r19/reporting/>.

► Example 2: Create an HTML file with Stata graphs

The do-file below creates a couple of graphs using Stata.

```
sysuse auto, clear
twoway lfitci mpg weight || scatter mpg weight, title(MPG as a function of weight)
graph export mpg1.png
regress mpg weight
rvfplot, yline(0) title(Residuals versus fitted values)
graph export diagplot.png, replace
```

end markdown.do

Suppose we wish to create a webpage (an HTML file) with the instructions on how to produce these graphs. First, we write `markdown2.txt` containing Markdown-formatted text and the Stata code to create the graphs above.

begin markdown2.txt

Creating graphs in Stata

=====

Below we review some diagnostic plots available in Stata, and we demonstrate how to overlay plots. We use 'auto.dta', which contains pricing and mileage data for 1978 automobiles.

<!-- In the previous example, we used three back-ticks for a block of code. Here we use a single back-tick for inline code. -->

Plotting predictions

We are interested in modeling the mean of `**mpg**`, miles per gallon, as a function of `**weight**`, car weight in pounds. We can use `**twoway lfitci**` to graph the predicted miles per gallon from a linear regression, as well as the confidence interval:

'''

```
sysuse auto, clear
twoway lfitci mpg weight
'''
```

To see how these predictions compare to our data, we can overlay a scatterplot of the actual data

'''

```
twoway lfitci mpg weight || scatter mpg weight, title(MPG as a function of weight)
'''
```

which produces the following graph:

![Graph of mpg](mpg1.png)

<!-- We previously used `**graph export**` to save the graph as a .png file, which we now embed in the document. -->

We could have also created separate graphs for domestic and foreign cars with the `**by()**` option. See [graph twoway lfitci](<https://www.stata.com/manuals/g-2graphptwowaylfitci.pdf>) in the Stata Graphics Reference Manual for details.

```
## Diagnostic plot
```

There are multiple diagnostic plots available for use after `**regress**`. Here, we use `**rvfplot**` to graphically check for a relationship between the residuals and fitted values from our model. We regress `**mpg**` on `**weight**` and then issue `**rvfplot**`.

```
'''
```

```
regress mpg weight
rvfplot, yline(0) title(Residuals versus fitted values)
'''
```

The commands above produce the following graph:

```
![Diagnostic plot](diagplot.png)
---
```

end markdown2.txt

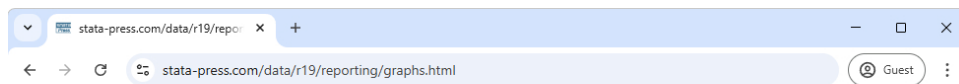
We can copy both of these files to our current working directory by typing

```
. copy https://www.stata-press.com/data/r19/reporting/markdown.do .
. copy https://www.stata-press.com/data/r19/reporting/markdown2.txt .
```

We now type the following to execute the commands in `markdown.do` that create the graphs and export them to PNG files. We then convert `markdown2.txt` to an HTML file by using the `markdown` command.

```
. do markdown.do
. markdown markdown2.txt, saving(graphs.html)
```

This creates `graphs.html`, which looks like the following:



Creating graphs in Stata

Below we review some diagnostic plots available in Stata, and we demonstrate how to overlay plots. We use `auto.dta`, which contains pricing and mileage data for 1978 automobiles.

Plotting predictions

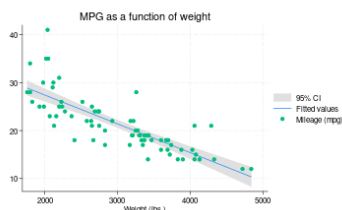
We are interested in modeling the mean of **mpg**, miles per gallon, as a function of **weight**, car weight in pounds. We can use `twoway lfitci` to graph the predicted miles per gallon from a linear regression, as well as the confidence interval:

```
sysuse auto, clear
twoway lfitci mpg weight
```

To see how these predictions compare to our data, we can overlay a scatterplot of the actual data

```
twoway lfitci mpg weight || scatter mpg weight, title(MPG as a function of weight)
```

which produces the following graph:



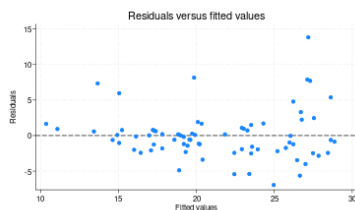
We could have also created separate graphs for domestic and foreign cars with the `by()` option. See [graph twoway lfitci](#) in the Stata Graphics Reference Manual for details.

Diagnostic plot

There are multiple diagnostic plots available for use after `regress`. Here, we use `rvfplot` to graphically check for a relationship between the residuals and fitted values from our model. We regress **mpg** on **weight** and then issue `rvfplot`.

```
regress mpg weight
rvfplot, yline(0) title(Residuals versus fitted values)
```

The commands above produce the following graph:



You can view the HTML file at <https://www.stata.press.com/data/r19/reporting/graphs.html>.

In this example, we ran the Stata commands and created the HTML file in two separate steps. However, we could do both in a single step with `dyndoc`, which processes Stata code to embed Stata output and graphs in the destination HTML file, and calls `markdown` to process the Markdown-formatted text (like we did above). See [\[RPT\] dyndoc](#) for information on how to create an HTML or Word (.docx) file with Stata output from a Markdown text file.

References

- Haghighi, E. F. 2016. [markdoc: Literate programming in Stata](#). *Stata Journal* 16: 964–988.
- . 2020a. [Software documentation with markdoc 5.0](#). *Stata Journal* 20: 336–362.
- . 2020b. [Developing, maintaining, and hosting Stata statistical software on GitHub](#). *Stata Journal* 20: 931–951.
- Jann, B. 2017. [Creating HTML or Markdown documents from within Stata using webdoc](#). *Stata Journal* 17: 3–38.

Also see

- [\[RPT\] Dynamic tags](#) — Dynamic tags for text files
- [\[RPT\] dyndoc](#) — Convert dynamic Markdown document to HTML or Word (.docx) document
- [\[RPT\] dyntext](#) — Process Stata dynamic tags in text file

Description

The putdocx suite of commands creates Office Open XML (.docx) documents that include text, formatted images, and tables of Stata estimation results and summary statistics. The following commands are used to create, format, add content to, and save .docx files that are compatible with Microsoft Word 2007 and later:

Create, save, and append .docx files (see [RPT] putdocx begin)

putdocx begin	Creates a .docx file for export
putdocx describe	Describes contents of the active .docx file
putdocx save	Saves and closes the .docx file
putdocx clear	Closes the .docx file without saving the changes
putdocx append	Appends the contents of multiple .docx files

Insert page breaks in a .docx file (see [RPT] putdocx pagebreak)

putdocx pagebreak	Adds a page break to the document
putdocx sectionbreak	Adds a new section to the document

Add paragraphs with text and images (see [RPT] putdocx paragraph)

putdocx paragraph	Adds a new paragraph to the active document
putdocx text	Adds text to the active paragraph
putdocx textblock	Adds a block of text to the active paragraph or to a new paragraph
putdocx textfile	Adds a block of preformatted text to a new paragraph with a predefined style
putdocx image	Appends an image to the active paragraph
putdocx pagenumber	Adds page numbers to a paragraph in a header or footer

Add tables to a .docx file (see [RPT] putdocx table)

putdocx table	Creates a new table in the .docx file containing estimation results, summary statistics, or data in memory
---------------	--

Add a table from a collection to a .docx file (see [RPT] putdocx collect)

putdocx collect	Adds a customized table created by collect or table to the .docx file
-----------------	---

In this manual entry, we show you how to use the putdocx commands by walking you through a first example that creates a simple report as a .docx file. We also provide some suggestions for choosing the best workflow for creating your own .docx files.

Remarks and examples

Remarks are presented under the following headings:

- Introduction*
- A first example*
 - Create a document*
 - Add a paragraph with text*
 - Add an image to a paragraph*
 - Add a table of estimation results*
- Automating a report*
- Workflow options for report building*
 - Create a complete document in Stata*
 - Create a document from Stata and Word*
 - Append files in Stata*
 - Append files in Word*

Introduction

putdocx is a suite of commands used to write paragraphs, images, and tables to an Office Open XML (.docx) file. This allows you to create Word documents that include Stata results and graphs. putdocx generates files compatible with Microsoft Word 2007 and later.

A first example

To get started with the putdocx commands, it is best to see them in action. Here, we demonstrate how to create a .docx file, include text, add a graph, and incorporate an estimation table all from within Stata.

This example shows the basic tools you need to create your own document. However, this is only a starting point. You may want to create more extensive and more customized documents, and putdocx allows you to do that. We save the details of customizing text, tables, and images for the individual entries of the commands listed [above](#).

Create a document

To demonstrate, we create a report on low birthweight using data from the study described in [Hosmer, Lemeshow, and Sturdivant \(2013, 24\)](#).

```
. use https://www.stata-press.com/data/r19/lbw
(Hosmer & Lemeshow data)
```

Before we can add any content to the report, we first need to create an active .docx document in memory. We do this with the putdocx begin command.

```
. putdocx begin
```

Because we did not include any options with putdocx begin, the document created uses the letter page size and the portrait orientation.

Add a paragraph with text

Now that the document is created, we can add other objects such as paragraphs, images, and tables to it. We begin by adding a title to our report. To do this, we add a paragraph using the Title style. Then we add the text of our title using putdocx text.

```
. putdocx paragraph, style(Title)
. putdocx text ("Report on low birthweights")
```

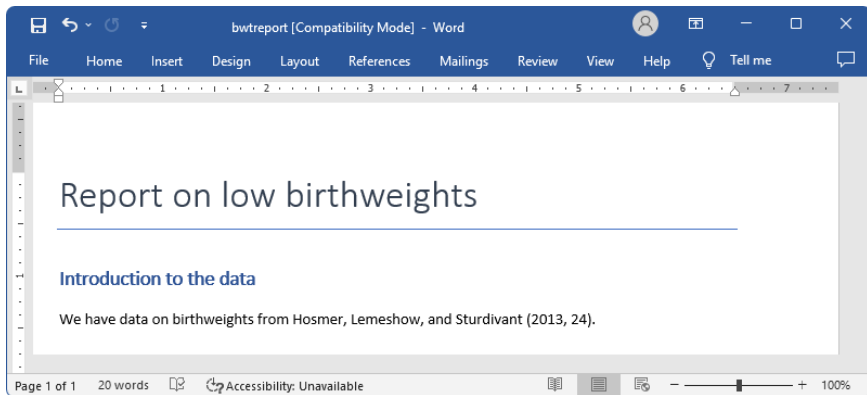
Next we add a heading for the description of our data:

```
. putdocx paragraph, style(Heading1)
. putdocx text ("Introduction to the data")
```

Now we are ready to add a standard paragraph where we cite the source of our dataset.

```
. putdocx paragraph
. putdocx text ("We have data on birthweights from Hosmer, Lemeshow, and ")
. putdocx text ("Sturdivant (2013, 24).")
. putdocx save bwtreport
successfully created "C:/mypath/bwtreport.docx"
```

We save the document we created in memory under the filename `bwtreport.docx`. When we open the document in Word, we see the following:



When we typed `putdocx save` above, our work was saved and the document was closed, so we now type `putdocx begin` to continue our work. So far, we have only added strings to our paragraphs, but text can also include any valid Stata expression. In the next section of our report, we add text with summary statistics for our data by referring directly to the results stored after `summarize`. We type `return list` and see that the mean is stored in the `r(mean)` scalar and with more decimal places than we wish to include in our sentence. Therefore, we use the `%5.2f` format to request that only two digits be displayed after the decimal.

```
. putdocx begin
. putdocx paragraph, style(Heading1)
. putdocx text ("Summary statistics")
. summarize bwt
```

Variable	Obs	Mean	Std. dev.	Min	Max
bwt	189	2944.286	729.016	709	4990

```

. return list
scalars:
      r(N) = 189
    r(sum_w) = 189
    r(mean) = 2944.285714285714
    r(Var) = 531464.3541033434
    r(sd) = 729.0160177275554
    r(min) = 709
    r(max) = 4990
    r(sum) = 556470

. putdocx paragraph
. putdocx text ("We have the recorded weight for 'r(N)' babies ")
. putdocx text ("with an average birthweight of ")
. putdocx text (" 'r(mean)' "), nformat(%5.2f)
. putdocx text (".")

```

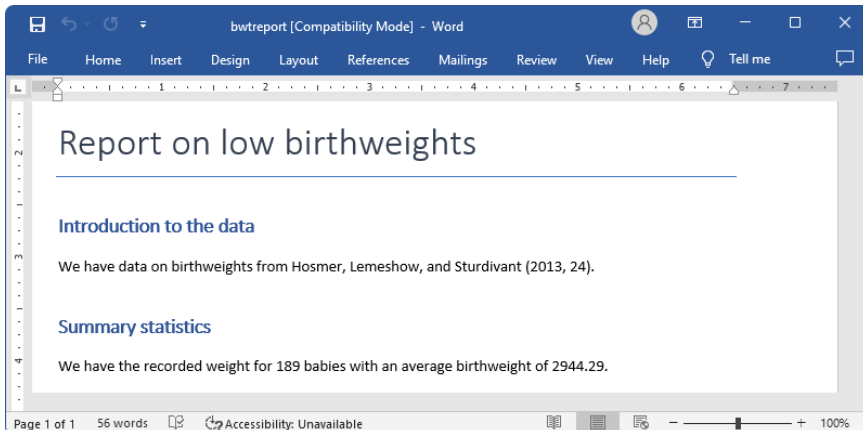
To review our document after adding this section, we save it again. However, because we want to add this new content to our existing `bwtreport.docx` file, we specify that we are appending to the file.

```

. putdocx save bwtreport, append
successfully appended to "C:/mypath/bwtreport.docx"

```

Our updated `bwtreport.docx` now looks like this:



Add an image to a paragraph

Next we graphically compare the average birthweights for babies according to the mother's characteristics. We begin this section of the report by adding another heading.

```

. putdocx begin
. putdocx paragraph, style(Heading1)
. putdocx text ("Birthweight by mother's smoking status")

```

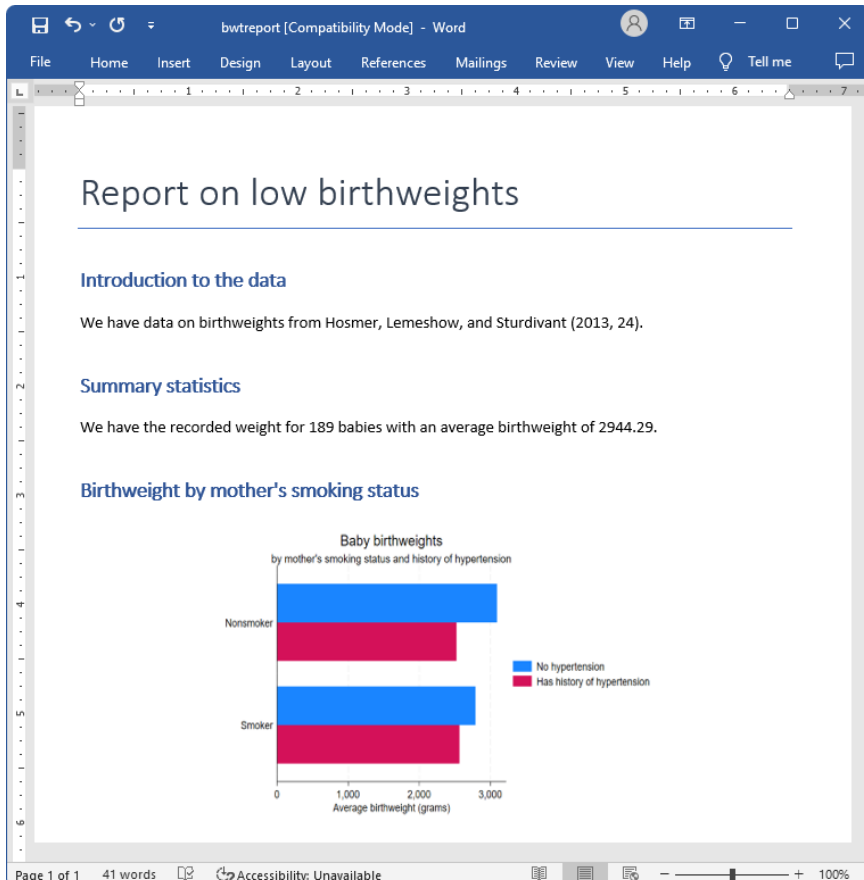
We graph the mean birthweight for babies with mothers who smoke versus those who do not, and separately for mothers with and without a history of hypertension. We use `graph hbar` to create our graph, specifying a title for the overall graph as well as for the *y* axis. We must convert the graph to one of the supported image formats: `.jpg`, `.emf`, `.tif`, or `.png`. We save it as a `.png` file with graph export.


```
. graph hbar bwt,
> over(ht, relabel(1 "No hypertension" 2 "Has history of hypertension"))
> over(smoke) asyvars ytitle(Average birthweight (grams)) title(Baby birthweights)
> subtitle(by mother's smoking status and history of hypertension)
. graph export bweight.png
file bweight.png saved as PNG format
```

Now we use `putdocx image` to append it to the active paragraph. To center the image, we specify the alignment of the paragraph. We also resize the image by setting the width at 4 inches and the height at 2.8 inches.

```
. putdocx paragraph, halign(center)
. putdocx image bweight.png, width(4) height(2.8)
. putdocx save bwtreport, append
successfully appended to "C:/mypath/bwtreport.docx"
```

We are ready to save our work again and take a look at our report. Again, we specify the `append` option with `putdocx save` to add the bar graph to the existing content of `bwtreport.docx`. Now the document contains the sections we previously exported plus the bar graph:



Add a table of estimation results

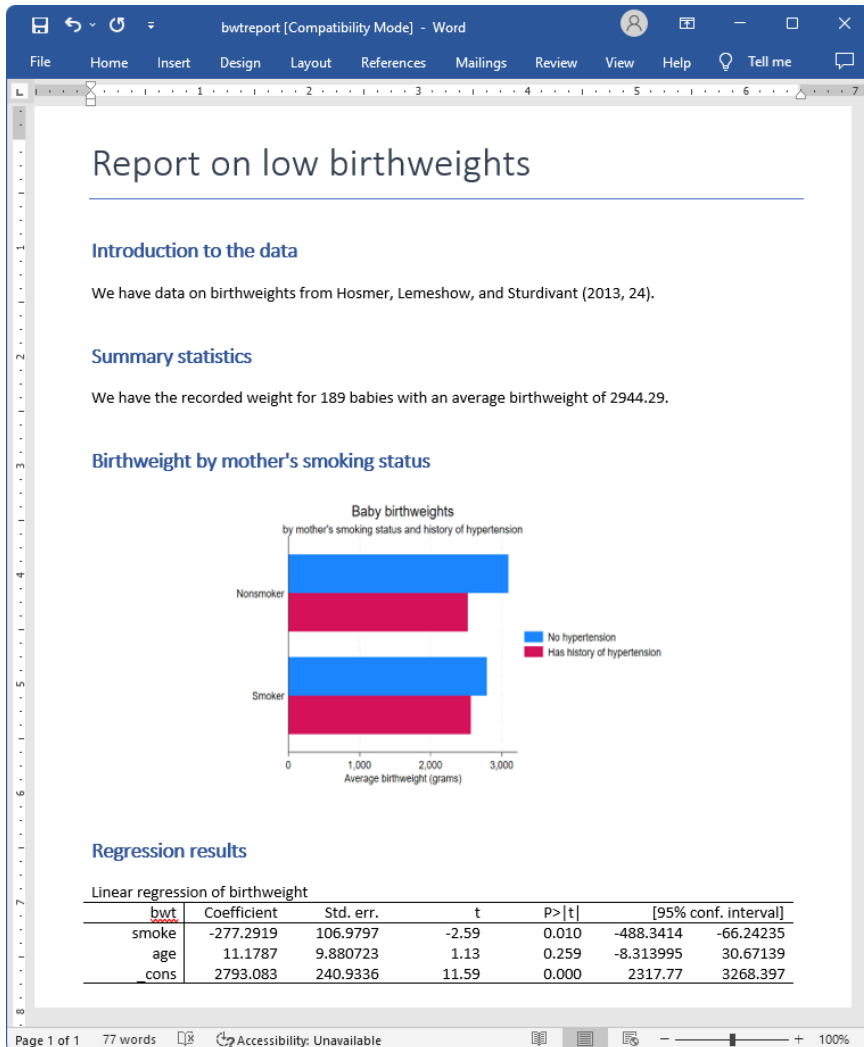
Next we add a table with regression results after modeling birthweight as a function of the mother’s age and whether she smokes. We export a table named bweight, including all the statistics shown in the regression output below. This is done effortlessly by specifying the etable output type with putdocx table. We also use the title() option to add a title to our table.

```
. putdocx begin
. putdocx paragraph, style(Heading1)
. putdocx text ("Regression results")
. regress bwt smoke age, noheader
```

bwt	Coefficient	Std. err.	t	P> t	[95% conf. interval]	
smoke	-277.2919	106.9797	-2.59	0.010	-488.3414	-66.24235
age	11.1787	9.880723	1.13	0.259	-8.313995	30.67139
_cons	2793.083	240.9336	11.59	0.000	2317.77	3268.397

```
. putdocx table bweight = etable, title("Linear regression of birthweight")
. putdocx save bwtreport, append
successfully appended to "C:/mypath/bwtreport.docx"
```

Again, we use `append` to add to the existing content in the file. `bwtreport.docx` now looks like this:



We will treat this as our final report. However, you will likely want to create `.docx` files with more content and perhaps more customization. See [RPT] [putdocx begin](#) for information on formatting the document as a whole, including specifying page size, page layout, font, and headers and footers. See [RPT] [putdocx paragraph](#) for information on adding entire blocks of text to a document; modifying the style, font, alignment, and other formatting of a paragraph; customizing the size and location of an image; and adding content to a header or footer. See [RPT] [putdocx table](#) for information on creating tables from stored results, matrices, data, and even images and for information on customizing these tables. See [RPT] [putdocx collect](#) for information on adding customized tables created with the `collect` suite of commands to a document. Finally, see [RPT] [putdocx pagebreak](#) for information on adding page breaks and section breaks to your document.

Automating a report

In the process of creating our report, we saved the document after exporting each section, specifying `append` to add on to our previous work. Saving the document intermittently allowed us to view the document in each stage of progress to ensure that it looked the way we wanted.

Once we have the layout we want, we do not need to view the Word document at each stage. Perhaps we need to create variations of this report rather frequently. Say that we receive monthly data on birthweights from a local hospital and we want to update the report with the new data. We condense the series of commands we previously ran by omitting all but the final `putdocx save` command and all but the initial `putdocx begin` command. We save our series of commands in a do-file. We add `version 19.5` or, if you do not have [StataNow](#), `version 19.0` to the top of our do-file to ensure that in future versions of Stata, our commands will continue to run and produce the same results they do today. In addition, because we do not want to save the report with the same name each time we run the do-file, we add the `args filename` command to the top of our file; see [\[P\] syntax](#) for information on this command. Now we can specify the name of the new Word document to be created when we run this file with the `do` command.

```

version 19.5      // (or version 19 if you do not have StataNow)
args filename
putdocx begin
putdocx paragraph, style(Title)
putdocx text ("Report on low birthweights")
putdocx paragraph, style(Heading1)
putdocx text ("Introduction to the data")
putdocx paragraph
putdocx text ("We have data on birthweights from Hosmer, Lemeshow, and ")
putdocx text ("Sturdivant (2013, 24).")
putdocx paragraph, style(Heading1)
putdocx text ("Summary statistics")
summarize bwt
putdocx paragraph
putdocx text ("We have the recorded weight for 'r(N)' babies ")
putdocx text ("with an average birthweight of ")
putdocx text (" 'r(mean)'  "), nformat(%5.2f)
putdocx text (".")
putdocx paragraph, style(Heading1)
putdocx text ("Birthweight by mother's smoking status")
graph hbar bwt,                                     ///
over(ht, relabel(1 "No hypertension" 2 "Has history of hypertension")) ///
over(smoke) asyvars ytitle(Average birthweight (grams)) ///
title(Baby birthweights)                          ///
subtitle(by mother's smoking status and history of hypertension)
graph export bweight.png
putdocx paragraph, halign(center)
putdocx image bweight.png, width(4) height(2.8)
putdocx paragraph, style(Heading1)
putdocx text ("Regression results")
regress bwt smoke age, noheader
putdocx table bweight = etable, title("Linear regression of birthweight")
putdocx save "'filename'", replace

```

After saving our do-file with the name `bwtreport.do`, we can now type

```

. use lbw_june, clear
. do bwtreport lbwreport_june

```

to create a new report in the same format as the previous one. This report will be run on `lbw_june.dta` (whereas we used `lbw.dta` earlier), and it will be saved under the filename `lbwreport_june.docx`.

Workflow options for report building

The `putdocx` suite is both capable and flexible because it has an abundance of formatting options to create and format your document completely from within Stata. However, by exporting content to a `.docx` file, it also allows you to interact Stata's capabilities with Word's additional formatting features. Depending on the contents of your document, you may find that one of the following methods of interacting features in Stata and Word is most suitable to creating your document:

1. Create a Word document completely from within Stata.
2. Use Stata to append documents created in both Stata and Word to complete a report.
3. Within a Word document, insert files created with Stata to complete a report.

We discuss the advantages of each approach below.

Create a complete document in Stata

In [A first example](#) and [Automating a report](#), we demonstrated how to create a Word document directly from Stata. We showed how to automate the process of creating a report once you have decided on specific formatting. Automation of reports and easy reproducibility are advantages of creating your document completely from within Stata. In addition, you can incorporate many of Stata's other features in the same do-file that creates the `.docx` file. For instance, you might include `assert` commands in your do-file to verify expectations you have of your data before generating your report.

One disadvantage of this approach is that it is not as conducive to reports with large amounts of text. You may prefer to type text in Word to take advantage of spell checking and other features. Also, if you create documents using Word's themes, tables of contents, bibliographies, and the like, you will need to access those from directly within Word. If either of these apply to the document you wish to create, consider method 2 or 3.

Create a document from Stata and Word

Methods 2 and 3 correspond to building your document in fragments, which may be preferable for three reasons. One is that you can write lengthy segments that are not dependent on Stata results or graphs directly in Word. While you can add blocks of text with `putdocx`, it may be done more easily in Word. You can then combine the file created in Word with another file created by using `putdocx`. Another reason is that, if you are writing a lengthy report, you can focus on one section at a time. Perhaps you have not quite decided how best to display your data graphically or what statistics you want to include in your estimation tables. You can save each section under its own filename as you complete it and then combine all the components. The third reason is that you do not have to re-create any formatting with `putdocx` that you already have in an existing Word document.

Whether you piece your document together in Stata or Word will likely depend on the amount of graphs and tables you will be including, and whether you are still deciding on formatting options or already have a customized template. If your report is centered on graphs and estimation results produced in Stata, you may find it easier to append all your files with `putdocx`—method 2. This method might also be preferred if you already have a template in Word with customized formatting.

Append files in Stata

Using this approach, the final .docx file is created in multiple steps. One or more .docx files are created in Word directly. Likewise, one or more .docx files are created using putdocx. We then use putdocx append or putdocx save, append to combine all of these files into a final .docx file.

The advantage of appending files in Stata is best explained with a hypothetical example. Suppose you are creating a report that will consist of an introduction, a graphics section, and an estimation section. You have already written a long introduction, added a header and footer, and saved your work under the filename report.docx. You create a file in Stata for your work on the graphics section and add the heading “Graphics”. After revising and formatting your graphs, you save your work:

```
. putdocx save graphs
```

Next you create a file for your work on estimation results with the heading “Estimation”. You test whether your table of results is better displayed under a portrait or landscape layout and whether the table should include certain statistics. Once you decide which results to display, you save your work for the estimation section:

```
. putdocx save estimation
```

Now you can complete your report by appending the graphics and estimation sections to your file containing the introduction.

```
. putdocx append report graphs estimation
```

The document report.docx will now contain your introduction, followed by the graphs, and finally the estimation results. The header and footer you formatted initially for report.docx will be applied throughout the complete document. Also, if you choose to add a table of contents to your report, the graphics and estimation headers will be incorporated into it. Using this method to build your document allowed you to experiment with the formatting of your graphs and tables, while easily applying your customized header and footer.

Append files in Word

Using this approach, the final document is created using Word. Typically, a single Word file is created that includes the majority of the content as well as the desired formatting. One or more segments of the document that contain Stata results are created using putdocx. We use Word’s insert features to incorporate the documents created by putdocx into the main Word document.

One advantage of inserting portions of your report into a main Word document is that you can experiment with different layouts. Suppose you are discussing a graph in your document, and you are not sure where it should be placed. Within Word, you can try inserting the graph in different gaps between the text. You can instantly see which structure makes the most sense for your report.

Another advantage is that you can also interact Excel’s capabilities by linking in an Excel worksheet in your Word document. Whether you worked directly in Excel or you worked in Stata and exported some results by using [putexcel](#), you can link content from Excel in Word. This way, you can interact Excel’s features, Word’s features, and Stata results all in one document.

This approach to building your document might also be preferable if you only need to include a limited number of graphs or statistics from Stata and you have already created a Word document customized to your preference.

References

- Chatfield, M. D. 2018. [Graphing each individual's data over time](#). *Stata Journal* 18: 503–516.
- Hosmer, D. W., Jr., S. A. Lemeshow, and R. X. Sturdivant. 2013. *Applied Logistic Regression*. 3rd ed. Hoboken, NJ: Wiley.
- Jann, B. 2016. [Creating L^AT_EX documents from within Stata using texdoc](#). *Stata Journal* 16: 245–263.
- Xue, Y., C. Li, and H. Si. 2023. [Reporting empirical results to .docx files](#). *Stata Journal* 23: 545–577.

Also see

- [RPT] [putdocx begin](#) — Create an Office Open XML (.docx) file
- [RPT] [putdocx collect](#) — Add a table from a collection to an Office Open XML (.docx) file
- [RPT] [putdocx pagebreak](#) — Add breaks to an Office Open XML (.docx) file
- [RPT] [putdocx paragraph](#) — Add text or images to an Office Open XML (.docx) file
- [RPT] [putdocx table](#) — Add tables to an Office Open XML (.docx) file

Description

`putdocx begin` creates an Office Open XML (.docx) file. This is the active document that the remaining `putdocx` commands modify.

`putdocx describe` describes the active .docx file.

`putdocx save` saves and closes the .docx file.

`putdocx clear` closes the .docx file without saving.

`putdocx append` appends the contents of one or more .docx files to another .docx file.

Quick start

Create a document in memory onto which subsequent contents are added

```
putdocx begin
```

Same as above, and specify half-inch margins on the left and right side of the page

```
putdocx begin, margin(left,0.5) margin(right,0.5)
```

Create a document with page numbers in the footer named `footer1`, using uppercase Roman numerals

```
putdocx begin, pagenum(upper_roman) footer(footer1)
```

Create a document with footer `footer1` on odd pages and footer `footer2` on even pages

```
putdocx begin, footer(footer1, default) footer(footer2, even)
```

Save the document in memory to disk as `myfile.docx`

```
putdocx save myfile
```

Append the contents of `filename2.docx` and `filename3.docx` to the end of the contents in `filename1.docx`

```
putdocx append filename1 filename2 filename3
```

Same as above, but save the resulting document in a file named `filename4.docx`

```
putdocx append filename1 filename2 filename3, saving(filename4)
```


Syntax

Create document for export

```
putdocx begin [ , begin_options ]
```

Describe active document

```
putdocx describe
```

Save and close document

```
putdocx save filename [ , save_options ]
```

Close without saving

```
putdocx clear
```

Append contents of documents

```
putdocx append filename1 filename2 [filename3 [...] ] [ , append_options ]
```

<i>begin_options</i>	Description
<code>pagesize(<i>psize</i>)</code>	set document page size
<code>landscape</code>	change document orientation to landscape
<code>font(<i>fspec</i>)</code>	set font, font size, and font color for the document
<code>pagenum(<i>pnspec</i>)</code>	set page number format
<code>header(<i>hname</i> [, <i>header_opts</i>])</code>	add a header
<code>footer(<i>fname</i> [, <i>footer_opts</i>])</code>	add a footer
<code>margin(<i>type</i> , #[<i>unit</i>])</code>	set page margins for the document

<i>save_options</i>	Description
<code>replace</code>	replace <i>filename</i> with the active document
<code>append</code>	append the active document to the end of <i>filename</i>
<code>append(<i>ap_opts</i>)</code>	append active document to <i>filename</i> and change style definitions along with page break, header, and footer settings
<code>nomsg</code>	suppress message with a link to <i>filename</i>

Only one of `replace`, `append`, or `append(ap_opts)` may be specified.

<i>append_options</i>	Description
<code>saving(filename [, replace])</code>	save the document to <i>filename</i> ; use <code>replace</code> to overwrite existing <i>filename</i>
<code>pagebreak</code>	begin each appended file on a new page
<code>headsrc(first last own)</code>	specify the document from which headers and footers are to be used; default is <code>headsrc(first)</code>
<code>stylesrc(previous own)</code>	specify the document from which style definitions are to be used; default is <code>stylesrc(previous)</code>
<code>pgnumrestart</code>	restart page numbering on the first page of each appended document
<code>nomsg</code>	suppress message with a link to resulting document

`collect` is allowed with `putdocx describe`; see [U] 11.1.10 Prefix commands.

Options

Options are presented under the following headings:

- [Options for putdocx begin](#)
- [Options for putdocx save](#)
- [Options for putdocx append](#)

Options for putdocx begin

`pagesize(psize)` sets the page size of the document. *psize* may be `letter`, `legal`, `A3`, `A4`, or `B4JIS`. The default is `pagesize(letter)`.

`landscape` changes the document orientation from portrait (the default) to landscape.

`font(fontname [, size [, color]])` sets the font, font size, and font color for the document.

fontname may be any valid font installed on the user’s computer. If *fontname* includes spaces, then it must be enclosed in double quotes. If *fontname* is not specified, the computer’s default font will be used.

size is a numeric value that represents font size measured in points. The default is 11.

color sets the text color. *color* may be one of the colors listed in [Colors](#) of [RPT] **Appendix for putdocx**; a valid RGB value in the form `### ### ###`, for example, `171 248 103`; or a valid RRGGBB hex value in the form `#####`, for example, `ABF867`.

The font size and font color may be specified individually without specifying *fontname*. Use `font("", size)` to specify the font size only. Use `font("", "", color)` to specify the font color only.

`pagenum(pnformat [, start [, chapStyle [, chapSep]])` specifies the format and starting page for page numbers.

pnformat specifies the page number format, such as decimals enclosed in parentheses or uppercase Roman numerals. The default is `decimal`. For a complete list of page number formats, see [Page number formats](#) of [RPT] **Appendix for putdocx**.

start specifies the starting page number and must be an integer greater than or equal to 0. The default is 1.

chapStyle specifies the style used for chapter headings. For a complete list of chapter styles, see [Chapter styles](#) of [RPT] **Appendix for putdocx**.

chapSep specifies the symbol used to separate chapter numbers and page numbers. *chapSep* may be colon, hyphen, em_dash, en_dash, or period. The default is hyphen.

This option is not required for including page numbers in your document, unless you want to include chapter numbers as well. To include chapter numbers, specify the style used to indicate chapters (*chapStyle*), and optionally, the symbol used to separate chapter and page numbers.

When specifying *chapStyle* and *chapSep*, chapter numbers will be reported along with the page numbers. To activate these options, you must specify a multilevel list style in Word that includes headings.

`header(hname [, default | first | even])` adds the header named *hname* to the document. The content of *hname*, including page numbers, can be defined with either `putdocx paragraph` or `putdocx table`. *hname* must be a valid name according to Stata's naming conventions; see [\[U\] 11.3 Naming conventions](#).

Specifying `header(hname)` without any suboptions applies the header to all pages in the document. The suboptions allow you to apply the header to the entire document (`default`), the first page (`first`), or the even pages (`even`). `header()` may be specified multiple times in a single command to use different headers throughout the document. For example, you could create a document with alternating headers on the odd and even pages or with headers on only odd pages.

Specifying `header(hname, default) header(hname2, first)` applies header *hname2* to the first page and header *hname* to all other pages.

Specifying `header(hname, default) header(hname2, even)` applies header *hname2* to even pages and header *hname* to the odd pages.

Specifying `header(hname, default) header(hname2, even) header(hname3, first)` applies header *hname3* to the first page, header *hname2* to the even pages, and header *hname* to the odd pages (except the first page).

`footer(fname [, default | first | even])` adds the footer named *fname* to the document. The content of *fname*, including page numbers, can be defined with either `putdocx paragraph` or `putdocx table`. *fname* must be a valid name according to Stata's naming conventions; see [\[U\] 11.3 Naming conventions](#).

Specifying `footer(fname)` without any suboptions applies the footer to all pages in the document. The suboptions allow you to apply the footer to the entire document (`default`), the first page (`first`), or the even pages (`even`). `footer()` may be specified multiple times in a single command to use different footers throughout the document. For example, you could create a document with alternating footers on the odd and even pages or with footers on only odd pages.

Specifying `footer(fname, default) footer(fname2, first)` applies footer *fname2* to the first page and footer *fname* to all other pages.

Specifying `footer(fname, default) footer(fname2, even)` applies footer *fname2* to even pages and footer *fname* to the odd pages.

Specifying `footer(fname, default) footer(fname2, even) footer(fname3, first)` applies footer *fname3* to the first page, footer *fname2* to the even pages, and footer *fname* to the odd pages (except the first page).

`margin(type, #[unit])` sets the page margins of the document. This option may be specified multiple times in a single command to account for different margin settings.

type identifies the location of the margin inside the document. *type* may be top, left, bottom, right, or all.

unit may be in (inch), pt (point), cm (centimeter), or twip (twentieth of a point). An inch is equivalent to 72 points, 2.54 centimeters, or 1440 twips. The default is in.

Options for putdocx save

`replace` specifies to overwrite *filename*, if it exists, with the contents of the document in memory.

`append` specifies to append the contents of the document in memory to the end of *filename*.

`append(ap_opts)` specifies to append the contents of the document in memory to *filename* and indicates whether new content will be added on a new page, which header and footer to use, which style definitions to use, and whether to restart page numbering for each document. *ap_opts* are `pagebreak`, `headsrc()`, `stylesrc()`, and `pgnumrestart`.

`pagebreak` appends the active document beginning on a new page.

`headsrc(file | active | own)` specifies the file whose header and footer will be used in the document.

`headsrc(file)` is the default; it applies the header and footer from *filename* throughout the document.

`headsrc(active)` applies the header and footer from the active document throughout the document.

`headsrc(own)` specifies that each document, *filename* and the active document, use its own header and footer. If the active document does not have a header or footer, it will inherit the header or footer from *filename*.

`stylesrc(file | own)` specifies the file from which style definitions, such as font style, color, size, etc., are to be used for overlapping styles.

`stylesrc(file)` is the default; it applies the style definitions in *filename* throughout the document.

`stylesrc(own)` specifies that each document, *filename* and the active document, use its own style definitions.

`pgnumrestart` restarts page numbering in each document; `pagebreak` must be specified with `pgnumrestart`.

`nomsg` suppresses the message that contains a link to *filename*.

Options for putdocx append

`saving(filename [, replace])` specifies to append the contents of the existing document *filename*₂ to the end of *filename*₁ and then write the result to the new document *filename*. If *filename* already exists, it can be overwritten by specifying `replace`. By default, *filename*₁ is overwritten with the document created by appending content from *filename*₂.

If more than two files are specified, the contents are appended in the order in which the files are listed. For example, *filename*₂ is appended to *filename*₁, *filename*₃ is appended to the result of the first append, and so forth.

`pagebreak` begins each appended file on a new page.

`headsrc(first|last|own)` specifies the document from which headers and footers are to be used.

`headsrc(first)`, the default, specifies that the header and footer from the first document being appended, *filename*₁, be applied throughout the document.

`headsrc(last)` specifies that the header and footer from the last document be applied throughout the document. `headsrc(last)` may not be specified with `pgnumrestart`.

`headsrc(own)` specifies that each file being appended use its own header and footer. If any file does not have a header or footer, it will inherit the header or footer from the previous document.

`stylesrc(previous|own)` specifies the document from which style definitions, such as font style, color, size, etc., are to be used for overlapping styles.

`stylesrc(previous)`, the default, specifies that the styles already defined in a previous document being appended be applied throughout the document.

`stylesrc(own)` specifies that each file being appended use its own style definitions.

`pgnumrestart` restarts page numbering on the first page of each appended document. The `pagebreak` option must be specified with `pgnumrestart`. This option may not be specified with `headsrc(last)`.

`nomsg` suppresses the message that contains a link to the resulting document.

Remarks and examples

Remarks are presented under the following headings:

Creating and formatting a .docx file
Including headers and footers
Describing the document
Saving or clearing the .docx file
Appending .docx files

Creating and formatting a .docx file

Before we can write to a .docx file by using `putdocx`, we need to create an active .docx document in memory by using the `putdocx begin` command. We can simply type

```
. putdocx begin
```

to create a document. We could now add content to this document. For information on adding text or images to the document, see [RPT] [putdocx paragraph](#). For information on adding tables to the document, see [RPT] [putdocx table](#).

Because we did not include any options in the above `putdocx begin` command, it creates a letter-size document with one-inch margins and pages in portrait orientation. It uses our computer's default font type and font color in 11-point size. We can specify other formats for the document as a whole by using the options available with `putdocx begin`. We can specify the page size, page orientation, page margins, and font properties for the document. We can also include page numbers, headers, and footers.

For example, to create a document with letter-size pages in landscape orientation and 11-point Garamond font, we type

```
. putdocx begin, landscape font(Garamond)
```

The page size, orientation, and margins set with `putdocx begin` remain in effect until a [section break](#) is added.

The font properties specified with `putdocx begin` remain in effect throughout the document, but we can change these properties for each [paragraph](#) (or each [sentence](#) or even each [word](#)) with the options available in [\[RPT\] putdocx paragraph](#). When specified without options, `putdocx paragraph` and `putdocx text` will default to the font properties specified in `putdocx begin`. Any text added with `putdocx textblock` will also default to the font properties specified in `putdocx begin`, unless we specify some other format by using dynamic tags within the text block.

Headers and footers remain in effect throughout the document by default, but they can be changed when a new section is added to the document.

Including headers and footers

To create a document that includes a header, footer, or page numbers requires multiple steps. First, we need to include either the `header()` or `footer()` option in our `putdocx begin` command, and if desired, we can customize the page numbers using the `pagenum()` option. Then we use either `putdocx paragraph` or `putdocx table` to add content, including page numbers, to the header or footer.

For instance, suppose we are creating an academic progress report for a local high school, and we want to include a footer with the page number and school name. We can create our document as follows:

```
. putdocx begin, pagenum(decimal) footer(npage)
```

In this command, we specified the decimal format for the page numbers, and we added a blank footer to our document.

Now, we can define the contents of the footer `npage` by using `putdocx paragraph`.

```
. putdocx paragraph, tofooter(npage)
. putdocx text ("Mountain High Report: ")
. putdocx pagenumber
```

This creates a paragraph with the school name and page number. The `tofooter()` option directed the content of this paragraph to the footer, `npage`, rather than including it in the body of the document.

The footer we added above will be applied to every page in our document, but we can also use different headers and footers throughout our document. For example, if we only want to display the footer on the even pages of our document, we would instead type

```
. putdocx begin, pagenum(decimal) footer(npage, even)
```

We could also specify one footer for the even pages and another footer for the odd pages as follows:

```
. putdocx begin, pagenum(decimal) footer(npage, even) footer(oddftr, default)
```

The footer called `npage` will be displayed on the even pages, and the footer called `oddftr` will be displayed on the odd pages. Note that if `oddftr` includes page numbers, they will also be formatted as decimals.

Describing the document

To view a document, we must first save the file. However, if we have been adding text, images, and tables to a document, we can describe the contents of the active document without saving it.

```
. putdocx describe
```

This reports the number of paragraphs and tables that have been added to the document.

Saving or clearing the .docx file

When we have finished adding content to our document, we can save it under a given filename, say, `myfile.docx`.

```
. putdocx save myfile, replace
```

The `replace` option specifies that we wish to overwrite the contents of an existing `myfile.docx`.

If the current document is a continuation of work stored in an existing file—say, `oldfile.docx`—then we can instead append the current document's contents to the existing file by typing

```
. putdocx save oldfile, append
```

This command adds the content of the active document that we have been working on to `oldfile.docx`, beginning directly below its original content. The new combined document is saved as `oldfile.docx`.

Issuing the `putdocx save` command automatically clears the active document from memory and closes it after it is saved.

Sometimes, we may want to clear the active document from memory without saving it. Perhaps we accidentally added a table with the wrong numeric formatting or we forgot to label a graph. We can type

```
. putdocx clear
```

This command clears the document in memory and automatically closes the document without saving.

`putdocx clear` is especially important if we are running a do-file with a series of `putdocx` commands repeatedly. Maybe we have `putdocx begin` at the top of our do-file and when we try to run it again, we get an error message. We must either save our work or close the document with `putdocx clear` before we can issue `putdocx begin` again.

Appending .docx files

Sometimes, it is easiest to create a final document by appending multiple .docx files. Perhaps you create parts of a report directly within Word and create other parts using `putdocx`. Perhaps you prefer to work on your document out of order and then compile the parts at the end. In these cases, we can use the `putdocx append` command.

Say we have four files that we are now ready to merge: `part1.docx`, `part2.docx`, `part3.docx`, and `part4.docx`. We can type

```
. putdocx append part1 part3 part2 part4, saving(allparts)
```

`putdocx append` appends files in the order in which they are specified, so the above command will append `part3` to the end of `part1`, `part2` to the end of `part3`, and `part4` to the end of `part2`. We also specified to save the resulting document in a new file called `allparts.docx`.

Oops! `part2` should have come before `part3` in the merged file. We can fix that by typing

```
. putdocx append part1 part2 part3 part4, saving(allparts, replace)
```

By including `replace` in the `saving()` option, we request that the resulting document overwrite the existing contents of `allparts.docx`.

If we do not use the `saving()` option, for example,

```
. putdocx append part1 part2 part3 part4
```

then the merged document is saved in the first-listed file, namely, `part1.docx`.

Reference

Weinreb, M. D., and J. Trinitapoli. 2022. [printcase: A command for visualizing single observations](#). *Stata Journal* 22: 958–968.

Also see

[RPT] [putdocx intro](#) — Introduction to generating Office Open XML (.docx) files

[RPT] [putdocx collect](#) — Add a table from a collection to an Office Open XML (.docx) file

[RPT] [putdocx pagebreak](#) — Add breaks to an Office Open XML (.docx) file

[RPT] [putdocx paragraph](#) — Add text or images to an Office Open XML (.docx) file

[RPT] [putdocx table](#) — Add tables to an Office Open XML (.docx) file

[RPT] [Appendix for putdocx](#) — Appendix for putdocx entries

Description

`putdocx collect` allows you to export a customized table from a collection to a table in the active `.docx` file. A collection contains a set of results that have been collected from one or more Stata commands using the `collect:` prefix or the `collect get` command. With the suite of `collect` commands, you can specify the layout and style of your table and customize the table.

Unlike `putdocx table`, which allows you to create tables and modify them as needed, `putdocx collect` is designed for exporting a table that you have already customized using the `collect` suite of commands. To learn more about creating customized tables, see [\[TABLES\] Intro](#).

`putdocx collect` also allows you to include tables created by `table` and `etable` in your report. The `table` command is a powerful command for producing tabulations, tables of summary statistics, tables of regression results, and more. The `etable` command creates tables with the active estimation results, results from margins, and results stored with `estimates store`. `table` and `etable` are unique in that they automatically create a collection. Their results can be further styled using the `collect` commands, or they can be included in your report as is with `putdocx collect`. See [\[R\] table intro](#) and [\[R\] etable](#) for more information on these commands.

Quick start

Create a table in the document using items from the current collection

```
putdocx collect
```

Same as above, and display the `putdocx` commands used to export to the `.docx` file

```
putdocx collect, noisily
```

Same as above, but instead of displaying the commands, save them to the file `myfile.do`

```
putdocx collect, dofile(myfile.do)
```

Syntax

```
putdocx collect [ , options ]
```

options	Description
<code>name(cname)</code>	use collection <i>cname</i>
<code>mentable</code>	keep table in memory rather than add it to document
<code>noisily</code>	show the <code>putdocx</code> commands used to export to the <code>.docx</code> file
<code>dofile(filename[, replace])</code>	save the <code>putdocx</code> commands used for exporting to the named do-file
<code>tablename(tablename)</code>	specify a name for the table

Options

`name(cname)` specifies a collection *cname* from which to export the customized table. By default, the customized table is taken from the current collection.

`memtable` specifies that the table be created and held in memory instead of being added to the active document. By default, the table is added to the document. This option is useful if the table is intended to be added to a cell of another table or to be used multiple times later.

`noisily` specifies that putdocx collect show the putdocx commands used to export to the .docx file.

`dofile(filename[, replace])` specifies that putdocx collect save to *filename* the putdocx commands used to export to the .docx file. If *filename* already exists, it can be overwritten by specifying *replace*. If *filename* is specified without an extension, .do is assumed.

`tablename(tablename)` specifies a name for the table. By naming the table, you can make further edits with putdocx table. The name must be a valid name according to Stata's naming conventions; see [U] 11.3 Naming conventions.

If the current collection contains multiple tables, the table names will contain the prefix *tablename* and an integer as the suffix. For example, if you specify `tablename(myreg)` and the collection contains three tables, the table names will be `myreg1`, `myreg2`, and `myreg3`. Also, note that a name is a sequence of 1 to 32 letters, digits, and underscores. If you are exporting multiple tables, consider using a shorter sequence to account for the integer suffix.

Remarks and examples

Remarks are presented under the following headings:

[Introduction](#)
[Export a collection](#)
[Specify the style for a collection](#)

Introduction

putdocx collect exports a table from a collection to the active .docx file. A collection is a set of results that have been obtained from one or more Stata commands with the `collect` suite of commands, from the `table` command, or from the `etable` command.

To create a table with the `collect` commands, you first collect results from one or more Stata commands. Then you create a table by specifying a layout that determines which results are to be included in the table and how they are to be arranged. For example, you can have the rows correspond to variables and the columns correspond to the statistics, or vice versa. You can also modify the labels in the table, format the results, add borders, change the font style, and make other styling changes to the table. Once you have finalized your table, you can add it to your active .docx file with putdocx collect.

It is possible to have many collections in memory. However, there is only one current collection, the one you are working with. The current collection is the one that will be exported with putdocx collect, but you can export a table from another collection by specifying the `name()` option.

A table that is conveniently added to a .docx file using putdocx collect could equivalently be added using a series of putdocx table commands. To see what those commands look like, use the `noisily` option with putdocx collect. This long list will quickly fill up your Results window, so

you may want to store those commands in a do-file instead by using the `dofile()` option with `putdocx collect`. For reproducibility purposes, you may choose to include your `collect` and `putdocx collect` commands in your do-file, or you may instead incorporate the commands from the do-file created by the `dofile()` option.

While the `collect` commands provide many generic styling options that will be applied to the table you export to your .docx file, the suite also includes `collect style putdocx`, which provides styling options specifically for tables exported to the .docx format. You can use `collect style putdocx` to specify the alignment of the table on the page, adjust column widths relative to the page width, and more.

In the following sections, we demonstrate how to create a customized table and add the table to the active .docx file. In these examples, we assume some familiarity with the `table` and `collect` commands. We recommend that you see [\[R\] table intro](#) for information on `table` and [\[TABLES\] Intro](#) for more information on `collect` to learn more about creating and customizing tables before incorporating them into your .docx file.

Export a collection

Suppose we would like to create a table comparing some measures of health between males and females. We will use data from the Second National Health and Nutrition Examination Survey (NHANES II) ([McDowell et al. 1981](#)). Our first step will be to collect the statistics we want, then we will specify the layout for our table and make styling edits as needed.

We would like to compare the mean and standard deviation of systolic blood pressure (`bpsystol`), body mass index (`bmi`), and weight for males and females. We compute the statistics using `table`. We specify `var` in the first set of parentheses so that the variables will appear on the rows. We specify `sex` and `result` in the second set of parentheses so that the `sex` and the statistics will define the columns. We also specify that we want two digits reported after the decimal for each statistic and that we do not want totals to be reported. The `table` command will display a table with these statistics and store them in a collection.

```
. use https://www.stata-press.com/data/r19/nhanes2l, clear
. table (var) (sex result), statistic(mean bpsystol bmi weight)
> statistic(sd bpsystol bmi weight) nformat(%7.2f) nototals
```

	Sex			
	Male		Female	
	Mean	Standard deviation	Mean	Standard deviation
Systolic blood pressure	132.89	20.99	129.07	25.13
Body mass index (BMI)	25.51	4.02	25.56	5.60
Weight (kg)	77.98	13.64	66.39	14.73

We now have a table with the statistics we want, and we can export it to a .docx file. But let's make a few styling edits before exporting this table. We can format all cells in the collection, all cells in a particular dimension, or specific cells in a particular dimension. Note that our table has three dimensions—`var`, `sex`, and `result`.

First, we can modify a table header using `collect style header`. The word `Sex` we see at the top of the table is the title for the dimension `sex`. `table` displays the title for this dimension by default. Because the labels `Male` and `Female` make this clear, we can specify `title(hide)` for this header. The full label `Standard deviation` is fairly long for a column header, so we can shorten it to `SD` using `collect label levels`. In this case, we modify only the `sd` level of the `result` dimension.

```
. collect style header sex, title(hide)
. collect label levels result sd "SD", modify
```

We now type collect layout without any arguments to view the current layout.

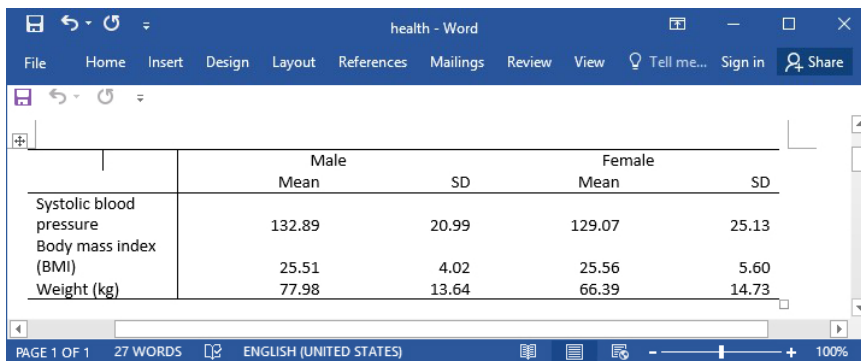
```
. collect layout
Collection: Table
  Rows: var
  Columns: sex#result
Table 1: 3 x 4
```

	Male		Female	
	Mean	SD	Mean	SD
Systolic blood pressure	132.89	20.99	129.07	25.13
Body mass index (BMI)	25.51	4.02	25.56	5.60
Weight (kg)	77.98	13.64	66.39	14.73

Now that we are done polishing our table of results, we create an active document, export the table, and save our work under the filename `health.docx`.

```
. putdocx begin
. putdocx collect
(collection Table posted to putdocx)
. putdocx save health, replace
successfully created "C:/mypath/health.docx"
```

The table is displayed as follows:



	Male		Female	
	Mean	SD	Mean	SD
Systolic blood pressure	132.89	20.99	129.07	25.13
Body mass index (BMI)	25.51	4.02	25.56	5.60
Weight (kg)	77.98	13.64	66.39	14.73

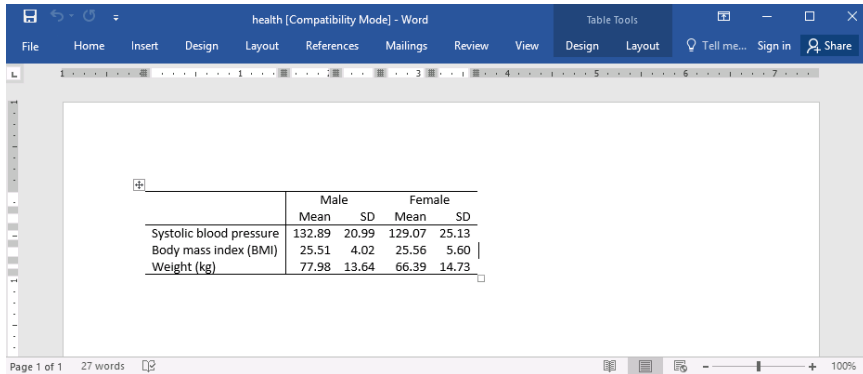
We have a nicely formatted table in our file; however, it is much wider than it needs to be. Also, it would be nice to have the variable labels placed on a single row. In the next section, we demonstrate how to change the width of the table as a whole and the width of the columns.

Specify the style for a collection

To finalize the look for our table from the previous section, we need to use one of the formatting features available with `putdocx table`. We can use `collect style putdocx` to apply some of these formatting options to our collection. We will use the `autofitcontents` layout, which will resize the column widths so that the contents fit.

```
. collect style putdocx, layout(autofitcontents)
. putdocx begin
. putdocx collect
(collection Table posted to putdocx)
. putdocx save health, replace
successfully replaced "C:/mypath/health.docx"
```

Now `health.docx` contains the following:



The screenshot shows a Microsoft Word document titled "health [Compatibility Mode] - Word". The document contains a table with the following data:

	Male		Female	
	Mean	SD	Mean	SD
Systolic blood pressure	132.89	20.99	129.07	25.13
Body mass index (BMI)	25.51	4.02	25.56	5.60
Weight (kg)	77.98	13.64	66.39	14.73

The status bar at the bottom indicates "Page 1 of 1" and "27 words".

Overall, our table looks neater, and the variable labels are not spread out across multiple rows.

Stored results

`putdocx collect` stores the following in `s()`:

Macros

```
s(collection)      name of collection
s(dofile)          name of the new do-file
```

References

- Huber, C. 2021. Customizable tables in Stata 17, part 2: The new `collect` command. *The Stata Blog: Not Elsewhere Classified*. <https://blog.stata.com/2021/06/07/customizable-tables-in-stata-17-part-2-the-new-collect-command/>.
- McDowell, A., A. Engel, J. T. Massey, and K. Maurer. 1981. "Plan and operation of the Second National Health and Nutrition Examination Survey, 1976–1980". In *Vital and Health Statistics*, ser. 1, no. 15. Hyattsville, MD: National Center for Health Statistics.

Also see

- [RPT] **putdocx intro** — Introduction to generating Office Open XML (.docx) files
- [RPT] **putdocx begin** — Create an Office Open XML (.docx) file
- [RPT] **putdocx pagebreak** — Add breaks to an Office Open XML (.docx) file
- [RPT] **putdocx paragraph** — Add text or images to an Office Open XML (.docx) file
- [RPT] **putdocx table** — Add tables to an Office Open XML (.docx) file
- [R] **etable** — Create a table of estimation results
- [R] **table intro** — Introduction to tables of frequencies, summaries, and command results
- [TABLES] **Intro** — Introduction
- [TABLES] **collect layout** — Specify table layout for the current collection
- [TABLES] **collect style putdocx** — Collection styles for putdocx

Description

putdocx pagebreak adds a page break to the document, placing subsequent content on the next page of the document.

putdocx sectionbreak adds a new section to the document and begins the section on the next page. It lets you vary the formatting of the pages within a single document. This command is most useful when you want to mix portrait and landscape layouts or apply different headers and footers across sections of your document.

Quick start

- Add a page break to the document
- putdocx pagebreak
- Begin a new section with a landscape layout in the document
- putdocx sectionbreak, landscape
- Begin a new section with the header appendix, and add the text “Appendix” to this header
- putdocx sectionbreak, header(appendix)
- putdocx paragraph, toheader(appendix)
- putdocx text ("Appendix")

Syntax

Add page break to document

putdocx pagebreak

Add section break to document

putdocx sectionbreak [, options]

options	Description
pagesize(<i>psize</i>)	set page size of the section
landscape	use a landscape orientation for the section
pagenum(<i>pnspec</i>)	set page number format
header(<i>hname</i> [, <i>header_opts</i>])	add a header
footer(<i>fname</i> [, <i>footer_opts</i>])	add a footer
margin(<i>type</i> , #[<i>unit</i>])	set page margins of section

Options

`pagesize(psize)` sets the page size of the section. *psize* may be `letter`, `legal`, `A3`, `A4`, or `B4JIS`. The default is `pagesize(letter)`.

`landscape` changes the section orientation from portrait (the default) to landscape.

`pagenum(pnformat [, start [, chapStyle [, chapSep]]])` specifies the format and starting page for page numbers.

pnformat specifies the page number format, such as decimals enclosed in parentheses or uppercase Roman numerals. The default is decimal. For a complete list, see [Page number formats](#) of [RPT] [Appendix for putdocx](#).

start specifies the starting page number and must be an integer greater than or equal to 0. The default is to continue page numbering from the previous section.

chapStyle specifies the style used for chapter headings. For a complete list of chapter styles, see [Chapter styles](#) of [RPT] [Appendix for putdocx](#).

chapSep specifies the symbol used to separate chapter numbers and page numbers. *chapSep* may be `colon`, `hyphen`, `em_dash`, `en_dash`, or `period`. The default is `hyphen`.

This option is not required for including page numbers in your document, unless you want to include chapter numbers as well. To include chapter numbers, specify the style used to indicate chapters (*chapStyle*), and optionally, the symbol used to separate chapter and page numbers.

When specifying *chapStyle* and *chapSep*, chapter numbers will be reported along with the page numbers. To activate these options, you must specify a multilevel list style in Word that includes headings.

`header(hname[, default | first | even])` adds the header named *hname* to the section. The content of *hname*, including page numbers, can be defined with either `putdocx paragraph` or `putdocx table`. *hname* must be a valid name according to Stata's naming conventions; see [\[U\] 11.3 Naming conventions](#).

Specifying `header(hname)` without any suboptions applies the header to all pages in the section. The suboptions allow you to apply the header to the entire section (`default`), the first page (`first`), or the even pages (`even`). `header()` may be specified multiple times in a single command to use different headers throughout the section. For example, you could create a section with alternating headers on the odd and even pages or with headers on only odd pages.

Specifying `header(hname, default) header(hname2, first)` applies header *hname2* to the first page and header *hname* to all other pages.

Specifying `header(hname, default) header(hname2, even)` applies header *hname2* to even pages and header *hname* to the odd pages.

Specifying `header(hname, default) header(hname2, even) header(hname3, first)` applies header *hname3* to the first page, header *hname2* to the even pages, and header *hname* to the odd pages (except the first page).

`footer(fname[, default | first | even])` adds the footer named *fname* to the section. The content of *fname*, including page numbers, can be defined with either `putdocx paragraph` or `putdocx table`. *fname* must be a valid name according to Stata's naming conventions; see [\[U\] 11.3 Naming conventions](#).

Specifying footer (*fname*) without any suboptions applies the footer to all pages in the section. The suboptions allow you to apply the footer to the entire section (default), the first page (*first*), or the even pages (*even*). footer() may be specified multiple times in a single command to use different footers throughout the section. For example, you could create a section with alternating footers on the odd and even pages or with footers on only odd pages.

Specifying footer(*fname*, default) footer(*fname2*, first) applies footer *fname2* to the first page and footer *fname* to all other pages.

Specifying footer(*fname*, default) footer(*fname2*, even) applies footer *fname2* to even pages and footer *fname* to the odd pages.

Specifying footer(*fname*, default) footer(*fname2*, even) footer(*fname3*, first) applies footer *fname3* to the first page, footer *fname2* to the even pages, and footer *fname* to the odd pages (except the first page).

margin(*type*, #[*unit*]) sets the page margins of the section. This option may be specified multiple times in a single command to account for different margin settings.

type identifies the location of the margin inside the document. *type* may be top, left, bottom, right, or all.

unit may be in (inch), pt (point), cm (centimeter), or twip (twentieth of a point). An inch is equivalent to 72 points, 2.54 centimeters, or 1440 twips. The default is in.

Remarks and examples

The putdocx pagebreak and putdocx sectionbreak commands are useful for organizing your .docx document. Whether you wish to insert page breaks for each new section in your file or you want to format different segments of your document differently, you can do this with these two commands.

To begin all subsequently added content on the next page in the active document, use the putdocx pagebreak command. putdocx pagebreak will retain the document formatting in effect, such as page size and headers, when you issue the command.

To begin all subsequently added content on the next page and also change the document formatting, use the putdocx sectionbreak command with its options. The page size, orientation, and margins specified with putdocx sectionbreak will remain in effect until you issue another putdocx sectionbreak command. putdocx sectionbreak with no options defaults to a letter-size page in portrait orientation, and uses one-inch margins. On the other hand, the headers (or footers) applied with putdocx sectionbreak will continue to be applied after section breaks and page breaks, unless you specify another header (or footer).

► Example 1: Modifying page size and orientation

Suppose we created a new document with letter-size pages in a portrait orientation by typing

```
. putdocx begin
```

After we have added some content to this document, we may need to include an extra-wide table. Say that our table has so many columns that we need to not only change the page orientation but also use a larger page. We need to add a section break:

```
. putdocx sectionbreak, pagesize(legal) landscape
```

Issuing the command above adds a new section that begins on the next page. This section will have legal-size pages and a landscape orientation.

We add our table (see [RPT] [putdocx table](#)) and now wish to return to the original letter-size page in portrait orientation. We do this with another section break.

```
. putdocx sectionbreak
```

As mentioned above, `putdocx sectionbreak` with no options defaults to a letter-size page in portrait orientation, and uses one-inch margins.



The `putdocx sectionbreak` command is helpful if you have, for example, wide tables or large images, but you do not want those inclusions to dictate the size or orientation of all the pages in your document. It is also helpful if your document is divided into sections, and you want to apply a different header or footer to each section.

► Example 2: Modifying header content

Suppose we are creating an annual report for the nearby community college that tracks the changes in the grade point average for each semester. Because this is a lengthy report, we organize it by specifying the semester we are reporting on in the header. We set up our document as follows:

```
putdocx begin, header(fall18)
putdocx paragraph, toheader(fall18)
putdocx text ("Fall 2018 ")
putdocx paragraph, style(Heading1)
putdocx text ("Changes in GPA")
putdocx sectionbreak, header(empty)
putdocx sectionbreak, header(winter18)
putdocx paragraph, toheader(winter18)
putdocx text ("Winter 2018 ")
putdocx paragraph, style(Heading1)
putdocx text ("Changes in GPA")
```

With the first `putdocx sectionbreak` command above, we added a blank page between the “Fall 2018” section and the “Winter 2018” section. Because we did not want to include any headers in this section, we specified the `header()` option with the header name `empty` for this section but did not define the content of `empty`, essentially adding a blank header.

This is just the frame for our document. We can now use other `putdocx` commands to insert any text, tables with summary statistics, graphics, and additional headings.



Also see

[RPT] [putdocx intro](#) — Introduction to generating Office Open XML (.docx) files

[RPT] [putdocx begin](#) — Create an Office Open XML (.docx) file

[RPT] [putdocx collect](#) — Add a table from a collection to an Office Open XML (.docx) file

[RPT] [putdocx paragraph](#) — Add text or images to an Office Open XML (.docx) file

[RPT] [putdocx table](#) — Add tables to an Office Open XML (.docx) file

[RPT] [Appendix for putdocx](#) — Appendix for `putdocx` entries

Description

`putdocx paragraph` adds a new paragraph to the document. The newly created paragraph becomes the active paragraph. All subsequent text or images will be appended to the active paragraph.

`putdocx text` adds text to the paragraph created by `putdocx paragraph`. The text may be a plain text string or any valid Stata expression (see [\[U\] 13 Functions and expressions](#)).

`putdocx textblock begin` adds a new paragraph to which a block of text can be added.

`putdocx textblock append` appends a block of text to the active paragraph.

`putdocx textblock end` ends a block of text initiated by `putdocx textblock begin` or `putdocx textblock append`.

`putdocx textfile` adds a block of preformatted text to a new paragraph with a predefined style.

`putdocx pagenumber` adds page numbers to a paragraph that is to be added to the header or footer.

`putdocx image` embeds a Portable Network Graphics (.png), Joint Photographic Experts Group (.jpg), Device-Independent Bitmap (.dib), Enhanced Metafile (.emf), Bitmap (.bmp), Tagged Image File Format (.tif), or Scalable Vector Graphics (.svg) file in the active paragraph.

Quick start

Add a paragraph to the document with centered horizontal alignment

```
putdocx paragraph, halign(center)
```

Append the text “This is paragraph text.” to the active paragraph and format the text as bold

```
putdocx text ("This is paragraph text."), bold
```

Add a PNG image saved as `myimg` to the active paragraph

```
putdocx image myimg.png
```

Add content to the header `myheading` with the text “My Header ” followed by the page number

```
putdocx paragraph, toheader(myheading)
```

```
putdocx text ("My Header ")
```

```
putdocx pagenumber
```

Add a new paragraph containing the text between the two `putdocx textblock` commands

```
putdocx textblock begin
```

```
This paragraph is written as a block of text.
```

```
putdocx textblock end
```

Add file `intro.txt` to the document as a new paragraph

```
putdocx textfile intro.txt
```

Same as above, but append `intro.txt` to the active paragraph

```
putdocx textfile intro.txt, append
```

Syntax

Add paragraph to document

```
putdocx paragraph [ , paragraph_options ]
```

Add text to paragraph

```
putdocx text (exp) [ , text_options ]
```

Add a paragraph with a block of text

```
putdocx textblock begin [ , textblock_options paragraph_options ]
```

Append a block of text to the active paragraph

```
putdocx textblock append [ , textblock_options ]
```

End the block of text initiated with putdocx textblock begin or putdocx textblock append

```
putdocx textblock end
```

Add page number to paragraph (to be added to header or footer)

```
putdocx pagenumber [ , text_options totalpages ]
```

Add a textfile as a block of preformatted text

```
putdocx textfile textfile [ , append stopat(string[ , stopatopt]) ]
```

Add image to paragraph

```
putdocx image filename [ , image_options ]
```

filename is the full path to the image file or the relative path from the current working directory. If *filename* contains spaces, it must be enclosed in double quotes.

<i>paragraph_options</i>	Description
style(<i>pstyle</i>)	set paragraph text with specific style
font(<i>fspec</i>)	set font, font size, and font color
halign(<i>hvalue</i>)	set paragraph alignment
valign(<i>vvalue</i>)	set vertical alignment of characters on each line
indent(<i>indenttype</i> , #[<i>unit</i>])	set paragraph indentation
spacing(<i>position</i> , #[<i>unit</i>])	set spacing between lines of text
shading(<i>sspec</i>)	set background color, foreground color, and fill pattern
toheader(<i>hname</i>)	add paragraph content to the header <i>hname</i>
tofooter(<i>fname</i>)	add paragraph content to the footer <i>fname</i>

<i>text_options</i>	Description
* nformat (% <i>fmt</i>)	specify numeric format for text
font (<i>fspec</i>)	set font, font size, and font color
bold	format text as bold
italic	format text as italic
script (sub super)	set subscript or superscript formatting of text
strikeout	strike out text
underline [(<i>upattern</i>)]	underline text using specified pattern
shading (<i>sspec</i>)	set background color, foreground color, and fill pattern
linebreak [(#)]	add line breaks after text
allcaps	format text as all caps
* smallcaps	format text as small caps
* hyperlink (<i>link</i>)	add the text as a hyperlink
* bookmark (<i>bmname</i>)	add the text as a bookmark named <i>bmname</i>
* bookmarklink (<i>bmname</i>)	link the text to bookmark <i>bmname</i>
* trim	remove the leading and trailing spaces in the text

* Cannot be specified with putdocx `pagenumber`.

<i>textblock_options</i>	Description
[no]paramode	specify whether blank lines signal new paragraphs; default is noparamode
[no]hardbreak	specify whether spaces are added after hard line breaks; default is nohardbreak

<i>image_options</i>	Description
width (#[<i>unit</i>])	set image width
height (#[<i>unit</i>])	set image height
linebreak [(#)]	add line breaks after image
link	insert link to image file
alt (<i>text</i>)	add alternative text to be read by voice software

fspec is

fontname [, *size* [, *color*]]

fontname may be any valid font installed on the user’s computer. If *fontname* includes spaces, then it must be enclosed in double quotes.

size is a numeric value that represents font size measured in points. The default is 11.

color sets the text color.

sspec is

bgcolor [, *fgcolor* [, *fpattern*]]

bgcolor specifies the background color.

fgcolor specifies the foreground color. The default foreground color is black.

fpattern specifies the fill pattern. The most common fill patterns are *solid* for a solid color (determined by *fgcolor*), *pct25* for 25% gray scale, *pct50* for 50% gray scale, and *pct75* for 75% gray scale. A complete list of fill patterns is shown in *Shading patterns* of [RPT] **Appendix for putdocx**.

color, *bgcolor*, and *fgcolor* may be one of the colors listed in *Colors* of [RPT] **Appendix for putdocx**; a valid RGB value in the form ### # 000, for example, 171 248 103; or a valid RRGGBB hex value in the form #####, for example, ABF867.

upattern may be any of the patterns listed in *Underline patterns* of [RPT] **Appendix for putdocx**. The most common underline patterns are *double*, *dash*, and *none*.

unit may be in (inch), pt (point), cm (centimeter), or twip (twentieth of a point). An inch is equivalent to 72 points, 2.54 centimeters, or 1440 twips. The default is in.

Options

Options are presented under the following headings:

- Options for putdocx paragraph*
- Options for putdocx text*
- Options for putdocx textblock begin*
- Options for putdocx textblock append*
- Options for putdocx pagenumbers*
- Options for putdocx textfile*
- Options for putdocx image*

Options for putdocx paragraph

style(*pstyle*) specifies that the text in the paragraph be formatted with style *pstyle*. Common values for *pstyle* are *Title*, *Subtitle*, and *Heading1*. See the complete list of paragraph styles in *Paragraph styles* of [RPT] **Appendix for putdocx**.

font(*fontname* [, *size* [, *color*]]) sets the font, font size, and font color for the text within the paragraph. The font size and font color may be specified individually without specifying *fontname*. Use *font*("", *size*) to specify font size only. Use *font*("", "", *color*) to specify font color only. For both cases, the default font will be used.

Specifying *font*() with *putdocx paragraph* overrides font properties specified with *putdocx begin*.

halign(*hvalue*) sets the horizontal alignment of the text within the paragraph. *hvalue* may be *left*, *right*, *center*, *both*, or *distribute*. *distribute* and *both* justify text between the left and right margins equally, but *distribute* also changes the spacing between words and characters. The default is *halign(left)*.

valign(*vvalue*) sets the vertical alignment of the characters on each line when the paragraph contains characters of varying size. *vvalue* may be *auto*, *baseline*, *bottom*, *center*, or *top*. The default is *valign(baseline)*.

`indent(indenttype, #[unit])` specifies that the paragraph be indented by # *units*. *indenttype* may be `left`, `right`, `hanging`, or `para`. `left` and `right` indent # *units* from the left or the right, respectively. `hanging` uses hanging indentation and indents lines after the first line by # inches unless another *unit* is specified. `para` uses standard paragraph indentation and indents the first line by # inches unless another *unit* is specified. This option may be specified multiple times in a single command to accommodate different indentation settings. If both `indent(hanging)` and `indent(para)` are specified, only `indent(hanging)` is used.

`spacing(position, #[unit])` sets the spacing between lines of text. *position* may be `before`, `after`, or `line`. `before` specifies the space before the first line of the current paragraph, `after` specifies the space after the last line of the current paragraph, and `line` specifies the space between lines within the current paragraph. This option may be specified multiple times in a single command to accommodate different spacing settings.

`shading(bgcolor [, fgcolor [, fpattern]])` sets the background color, foreground color, and fill pattern for the paragraph.

`toheader(hname)` specifies that the paragraph be added to the header *hname*. The paragraph will not be added to the body of the document.

`tofooter(fname)` specifies that the paragraph be added to the footer *fname*. The paragraph will not be added to the body of the document.

Options for putdocx text

`nformat(%fmt)` specifies the numeric format of the text when the content of the new text appended to the paragraph is a numeric value. This setting has no effect when the content is a string.

`font(fontname [, size [, color]])` sets the font, font size, and font color for the new text within the active paragraph. The font size and font color may be specified individually without specifying *fontname*. Use `font("", size)` to specify the font size only. Use `font("", "", color)` to specify the font color only. For both cases, the default font will be used.

Specifying `font()` with `putdocx text` overrides all other font settings, including those specified with `putdocx begin` and `putdocx paragraph`.

`bold` specifies that the new text in the active paragraph be formatted as bold.

`italic` specifies that the new text in the active paragraph be formatted as italic.

`script(sub | super)` changes the script style of the new text. `script(sub)` makes the text a subscript. `script(super)` makes the text a superscript.

`strikeout` specifies that the new text in the active paragraph have a strikeout mark.

`underline[upattern]` specifies that the new text in the active paragraph be underlined and optionally specifies the format of the line. By default, a single underline is used.

`shading(bgcolor [, fgcolor [, fpattern]])` sets the background color, foreground color, and fill pattern for the active paragraph. Specifying `shading()` with `putdocx text` overrides shading specifications from `putdocx paragraph`.

`linebreak[(#)]` specifies that one or # line breaks be added after the new text.

`allcaps` specifies that all letters of the new text in the active paragraph be capitalized.

`smallcaps` specifies that all letters of the new text in the active paragraph be capitalized, with larger capitals for uppercase letters and smaller capitals for lowercase letters.

`hyperlink(link)` adds the text as a hyperlink to the webpage address specified in *link*.

`bookmark(bmname)` adds the text as a bookmark named *bmname*. You can link to this bookmark from other text within a paragraph or from an expression in a table cell by using the `bookmarklink(bmname)` option.

`bookmarklink(bmname)` adds the text as a link to the bookmark named *bmname*. Note that bookmark links can point only to bookmarks in the current document. Additionally, these links cannot point to bookmarks in documents to which you will be appending.

`trim` removes the leading and trailing spaces in the text.

Options for putdocx textblock begin

`style(pstyle)` specifies that the text in the paragraphs be formatted with style *pstyle*. Common values for *pstyle* are `Title`, `Subtitle`, and `Heading1`. See the complete list of paragraph styles in [Paragraph styles of \[RPT\] Appendix for putdocx](#).

`font(fontname [, size [, color]])` sets the font, font size, and font color for the text within the paragraphs. The font size and font color may be specified individually without specifying *fontname*. Use `font("", size)` to specify font size only. Use `font("", "", color)` to specify font color only. For both cases, the default font will be used.

Specifying `font()` with `putdocx textblock begin` overrides font properties specified with `putdocx begin`.

`halign(hvalue)` sets the horizontal alignment of the text within the paragraphs. *hvalue* may be `left`, `right`, `center`, `both`, or `distribute`. `distribute` and `both` justify text between the left and right margins equally, but `distribute` also changes the spacing between words and characters. The default is `halign(left)`.

`valign(vvalue)` sets the vertical alignment of the characters on each line when the paragraphs contain characters of varying size. *vvalue* may be `auto`, `baseline`, `bottom`, `center`, or `top`. The default is `valign(baseline)`.

`indent(indenttype, #[unit])` specifies that the paragraphs be indented by # *units*. *indenttype* may be `left`, `right`, `hanging`, or `para`. `left` and `right` indent # *units* from the left or the right, respectively. `hanging` uses hanging indentation and indents lines after the first line by # inches unless another *unit* is specified. `para` uses standard paragraph indentation and indents the first line by # inches unless another *unit* is specified. This option may be specified multiple times in a single command to accommodate different indentation settings. If both `indent(hanging)` and `indent(para)` are specified, only `indent(hanging)` is used.

`spacing(position, #[unit])` sets the spacing between lines of text. *position* may be `before`, `after`, or `line`. `before` specifies the space before the first line of each paragraph in the text block, `after` specifies the space after the last line of each paragraph, and `line` specifies the space between lines within each paragraph. This option may be specified multiple times in a single command to accommodate different spacing settings.

`shading(bgcolor [, fgcolor [, fpattern]])` sets the background color, foreground color, and fill pattern for the paragraphs.

`toheader(hname)` specifies that the first paragraph in a text block be added to the header *hname*. The first paragraph will not be added to the body of the document.

When you specify the `paramode` option with `toheader()`, all paragraphs after the first will be added to the body of the document, not the header.

`tofooter(fname)` specifies that the first paragraph in a text block be added to the footer *fname*. The first paragraph will not be added to the body of the document.

When you specify the `paramode` option with `tofooter()`, all paragraphs after the first will be added to the body of the document, not the footer.

`noparamode` and `paramode` specify whether blank lines within a block of text signal the beginning of a new paragraph in the document. `noparamode`, the default, exports the block of text as a single paragraph, removing any empty lines. `paramode` treats blank lines as an indication of a new paragraph. When `paramode` is specified, the text following each blank line will begin a new paragraph. The treatment of paragraphs within text blocks can also be specified by using `set docx_paramode`; see [RPT] [set docx](#).

`nohardbreak` and `hardbreak` specify whether `putdocx textblock` inserts a space at the beginning of lines or respects spaces exactly as they are typed. `nohardbreak`, the default, respects spaces at the beginning and end of lines exactly as they are typed in the text block. The `hardbreak` option inserts a space at the beginning of all nonempty lines, excluding the first line in the block of text and the first line of each paragraph. Thus, `hardbreak` adds a space after the hard line breaks in the text block. The treatment of spaces at line breaks can also be specified by using `set docx_hardbreak`; see [RPT] [set docx](#).

Options for `putdocx textblock append`

`noparamode` and `paramode` specify whether blank lines within a block of text signal the beginning of a new paragraph in the document. `noparamode`, the default, exports the block of text as a single paragraph, removing any empty lines. `paramode` treats blank lines as an indication of a new paragraph. When `paramode` is specified, the text following each blank line will begin a new paragraph. The treatment of paragraphs within text blocks can also be specified by using `set docx_paramode`; see [RPT] [set docx](#).

`nohardbreak` and `hardbreak` specify whether `putdocx textblock` inserts a space at the beginning of lines or respects spaces exactly as they are typed. `nohardbreak`, the default, respects spaces at the beginning and end of lines exactly as they are typed in the text block. The `hardbreak` option inserts a space at the beginning of all nonempty lines, excluding the first line in the block of text and the first line of each paragraph. Thus, `hardbreak` adds a space after the hard line breaks in the text block. The treatment of spaces at line breaks can also be specified by using `set docx_hardbreak`; see [RPT] [set docx](#).

Options for `putdocx pagenumber`

`font(fontname [, size [, color]])` sets the font, font size, and font color for the page numbers. The font size and font color may be specified individually without specifying *fontname*. Use `font("", size)` to specify the font size only. Use `font("", "", color)` to specify the font color only. For both cases, the default font will be used.

Specifying `font()` with `putdocx pagenumber` overrides all other font settings, including those specified with `putdocx begin` and `putdocx paragraph`.

`bold` specifies that the page numbers be formatted as bold.

italic specifies that the page numbers be formatted as italic.

`script(sub | super)` changes the script style of the page numbers. `script(sub)` makes the page numbers subscripts. `script(super)` makes the page numbers superscripts.

`strikeout` specifies that the page numbers have a strikeout mark.

`underline[(upattern)]` specifies that the page numbers be underlined and optionally specifies the format of the line. By default, a single underline is used.

`shading(bgcolor [, fgcolor [, fpattern]])` sets the background color, foreground color, and fill pattern for the page numbers. Specifying `shading()` with `putdocx pagenumber` overrides shading specifications from `putdocx paragraph`.

`linebreak[(#)]` specifies that one or # line breaks be added after the page numbers.

`allcaps` specifies that the page numbers be capitalized; this only applies with the page number formats `lower_letter`, `lower_roman`, `cardinal_text`, and `ordinal_text`.

`totalpages` specifies that the total number of pages be displayed, instead of the current number of the page.

Options for putdocx textfile

`append` specifies that the text file be appended to the current paragraph, rather than being added as a new paragraph.

`stopat(string[, stopatopt])` specifies that only a portion of the text file be included, using the specified string as the end point.

stopatopt must be one of `contain`, `begin`, or `end`. The default is `contain`, which matches the string inside the line. `begin` matches the string at the beginning of the line. `end` matches the string at the end of the line. If the line matches the string, the rest of the text file, including the line, will not be included in the document.

Options for putdocx image

`width(#[unit])` sets the width of the image. If the width is larger than the body width of the document, then the body width is used. If `width()` is not specified, then the default size is used; the default is determined by the image information and the body width of the document.

`height(#[unit])` sets the height of the image. If `height()` is not specified, then the height of the image is determined by the width and the aspect ratio of the image.

`linebreak[(#)]` specifies that one or # line breaks be added after the new image.

`link` specifies that a link to the image *filename* be inserted into the document. If the image is linked, then the referenced file must be present so that the document can display the image.

`alt(text)` specifies alternative text for the image to be read by voice software.

Remarks and examples

Text, headers, footers, page numbers, and images are added to a .docx file via paragraphs. You first create a paragraph and then append text and other content to it. A paragraph can be formatted as a whole, and portions of the text within a paragraph can also be formatted individually.

To a paragraph, you can add valid Stata expressions, including strings, algebraic expressions, and direct references to stored results using `putdocx text`. For lengthier additions, you can add blocks of text using `putdocx textblock`. You can also insert preformatted text files in your `.docx` file with `putdocx textfile`. With `putdocx image`, you can add images such as Stata graphs to a paragraph.

With `putdocx paragraph`, you not only can add standard paragraphs to a document but also can create, with the variety of formatting options available to customize paragraphs, a document complete with a title, subtitle, headings, and headers and footers.

Remarks are presented under the following headings:

- Adding a paragraph*
- Formatting text*
- Working with blocks of text*
- Adding an image to the document*
- Adding a bookmark to the document*
- Inserting text files in the document*

Adding a paragraph

With each `putdocx paragraph` command we execute, we have the opportunity to format all the text within the paragraph. For instance, we can specify font properties such as size and color along with alignment, indentation, and line spacing.

Specifying `putdocx paragraph` without any options will create an active paragraph in the body of the document. To instead add the paragraph content to a header or footer, use the `toheader()` or `tofooter()` option.

► Example 1: Add a title to the document

Suppose we want to create a report on some of the contributing factors to low birthweight and save it in a file called `bweight.docx`. We use the Hosmer and Lemeshow dataset.

```
. use https://www.stata-press.com/data/r19/lbw
(Hosmer & Lemeshow data)
```

We must first create our document as described in [RPT] **putdocx begin**. Because we want to add a header to our document, we specify the `header()` option, and we specify the font size for the whole document.

```
. putdocx begin, header(bwtreport) font(, 14)
```

Now we create a paragraph to which we can append text or an image. Here we indicate that we are adding a title, and then we use `putdocx text` to add the title itself.

```
. putdocx paragraph, style(Title)
. putdocx text ("Factors of low birthweight")
```

The paragraph will remain active until we add a new paragraph, a table, a section break, or a page break (see [RPT] **putdocx table** and [RPT] **putdocx pagebreak**).

◀

► Example 2: Define the header content

Next we define the content of our header. To do so, we use the `toheader()` option, which places the contents of the paragraph in the header rather than in the body of our document. We also include page numbers in the header:

```
. putdocx paragraph, toheader(bwtreport)
. putdocx text ("Analysis of birthweight: ")
. putdocx pagenumber
```

Now that our file is set up with a title and header, we add some text to the body of our document. In the next example, we will add a description of our dataset. Because the active paragraph is the one with the header content, we need to start a new paragraph that does not use the `toheader()` option.

```
. putdocx paragraph
```

`putdocx paragraph` with no options will use your computer's default font and the 14-point size that we requested with our previous `putdocx begin` command. See [Options for putdocx paragraph](#) for paragraph formatting options that can be specified to override defaults and settings specified in `putdocx begin`.



Formatting text

In each `putdocx text` command, we can specify any valid Stata expression (see [\[U\] 13 Functions and expressions](#)) including, but not limited to, a string of plain text. The expression can be formatted using the [Options for putdocx text](#), which would override the settings specified in `putdocx paragraph` and `putdocx begin`.

► Example 3: Format text individually

Suppose we want to write a description of `lbw.dta` to `bweight.docx`, including the number of women in the dataset and the average birthweight in grams (`bwt`) for their infants. We use the `summarize` command to get these descriptive statistics for the `bwt` variable.

```
. summarize bwt
```

Variable	Obs	Mean	Std. dev.	Min	Max
bwt	189	2944.286	729.016	709	4990

We can use the results from `summarize` in the text we write. To see the available returned results, we type `return list`. (See [\[P\] return](#) for more about stored results from Stata commands.)

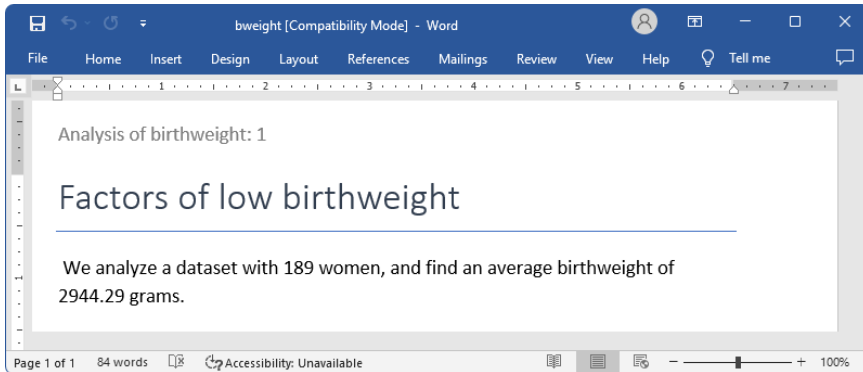
```
. return list
scalars:
      r(sum) = 556470
      r(max) = 4990
      r(min) = 709
      r(sd) = 729.0160177275554
      r(Var) = 531464.3541033434
      r(mean) = 2944.285714285714
      r(sum_w) = 189
      r(N) = 189
```

We see that `r(N)` stores the number of observations and `r(mean)` stores the average birthweight. Below, we create a paragraph containing this information and save the changes to our document.

```
. putdocx text (" We analyze a dataset with 'r(N)' women,")
. putdocx text (" and find an average birthweight of ")
. putdocx text ( r(mean)), nformat(%5.2f)
. putdocx text (" grams.")
```

```
. putdocx save bweight
successfully created "C:/mypath/bweight.docx"
```

Above, we formatted the mean birthweight to report only two digits after the decimal. The `bweight.docx` file now looks like this:



Working with blocks of text

While `putdocx text` is great for customizing small bits of text, it is more efficient to use the `putdocx textblock` commands when adding big blocks of text to your document. (Note that the commands `putdocx textblock begin` and `putdocx textblock append` cannot be used interactively.) To add a paragraph with a block of text to your document, simply enclose your text between the `putdocx textblock` commands as follows:

```
putdocx textblock begin
... block of text to add
putdocx textblock end
```

To instead append a block of text to the active paragraph, use `putdocx textblock append`.

For example, continuing with the Hosmer and Lemeshow dataset, suppose we wanted to add more details about the data. We might include a text block by typing the following:

```
putdocx textblock begin
We use data presented in Hosmer, Lemeshow, and Sturdivant (2013, 24).
These data record women's demographics, such as age and race, and their medical
history, including whether they have a history of hypertension.
putdocx textblock end
```

Inserting a block of text does not limit us to using the same format throughout the block. Before we include this text block in our document, we want to add a bit of text that is formatted differently from the rest of the text block.

We can use the `<<dd_docx_display>>` dynamic tag within our text block to include Stata results and modify the text style. This dynamic tag will execute Stata's `display` command and then replace the tag with its output. We can format the output by specifying any of the [text options](#) that are also available with `putdocx text`. The entire `<<dd_docx_display text_option : display_directive>>` tag must be contained on one line—there cannot be a line break within the tag. See [\[RPT\] Dynamic tags](#) for more information on the `<<dd_docx_display>>` dynamic tag.

► Example 4: Formatting text within a block

Let's modify the code above to include the name of the book that presented the dataset. Because we saved our document to view it, we must first create a new active document. Then we add the underlined book title within our text block as follows:

```
putdocx begin, font(, 14)
putdocx textblock begin
We use data presented in the book
<<dd_docx_display underline: "Applied Logistic Regression, 3rd Edition">>
by Hosmer, Lemeshow, and Sturdivant. These data record women's demographics, such
as age and race, and their medical history, including whether they have
a history of hypertension.
putdocx textblock end
```

In addition to formatting text, we can use the «dd_docx_display» dynamic tag to include Stata results. For example, we briefly mention how many women in this dataset smoked while pregnant and how many of them gave birth to an infant that weighed less than 2,500 grams. To add this content, we first store some results in local macros and then summarize the data.

```
. count if smoke==1
74
. local smoke = 'r(N) '
. count if low==1
59
. local lbw = 'r(N) '
. summarize
```

Variable	Obs	Mean	Std. dev.	Min	Max
id	189	121.0794	63.30363	4	226
low	189	.3121693	.4646093	0	1
age	189	23.2381	5.298678	14	45
lwt	189	129.8201	30.57515	80	250
race	189	1.846561	.9183422	1	3
smoke	189	.3915344	.4893898	0	1
ptl	189	.1957672	.4933419	0	3
ht	189	.0634921	.2444936	0	1
ui	189	.1481481	.3561903	0	1
ftv	189	.7936508	1.059286	0	6
bwt	189	2944.286	729.016	709	4990

We can now reference these macros and any stored results in our block of text, as follows:

```
putdocx textblock begin
Out of the <<dd_docx_display: 'r(N)''>> women in this dataset,
<<dd_docx_display shading(yellow): 'smoke''>> smoked during their pregnancy, and
<<dd_docx_display : %4.1f 100*'lbw'/'r(N)''>> % gave birth to an infant that weighed
less than 2,500 grams.
putdocx textblock end
```

Adding an image to the document

We can add any existing .png, .jpg, .dib, .emf, .bmp, .tif, and .svg image files to a .docx file with `putdocx image`. For example, you could include a company logo. We can also add graphs from Stata output. Because Stata graphs use the .gph extension, we must first use `graph export` to convert the Stata graph to one of the supported image formats; see [G-2] [graph export](#).

□ Technical note

When you add a .svg file to a .docx file using `putdocx`, an extra .png file is generated using the original .svg file and embedded in the .docx file. This .png file is used to render the original .svg file in viewers that do not support rendering .svg files.



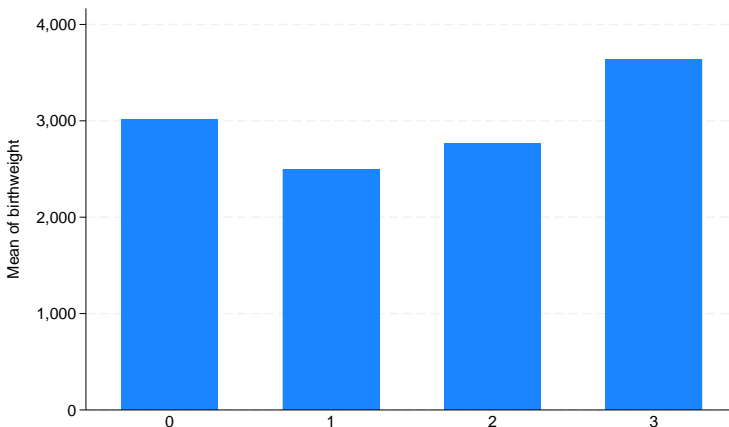
By default, images are embedded in the file. If the image is embedded, it becomes a part of the document and has no further relationship with the original image on the disk. We can instead link the image by specifying the `link` option. Using linked images means that if the saved image file is updated, then the linked image in the document will reflect the change.

If we add an image after text and want the paragraph that contains the image to have the same format as the active paragraph, we insert the image with no additional step. However, when we want to change the formatting or if there is no active paragraph, we must create one using `putdocx paragraph`. Note that we do not need to declare a new paragraph to insert an image into the cell of a table; see [example 7](#) in [RPT] [putdocx table](#).

▷ Example 5: Export a Stata graph

Another probable factor of low birthweight is the mother's history of premature labor. We have a variable (`ptl`) that records the number of times a mother prematurely went into labor. Let's add a graph showing the mean birthweight separately for mothers who never went into premature labor and those who went into premature labor once, twice, and three times. We use the `graph bar` command followed by `graph export` to create a .png file.

```
. graph bar bwt, over(ptl) ytitle(Mean of birthweight)
```

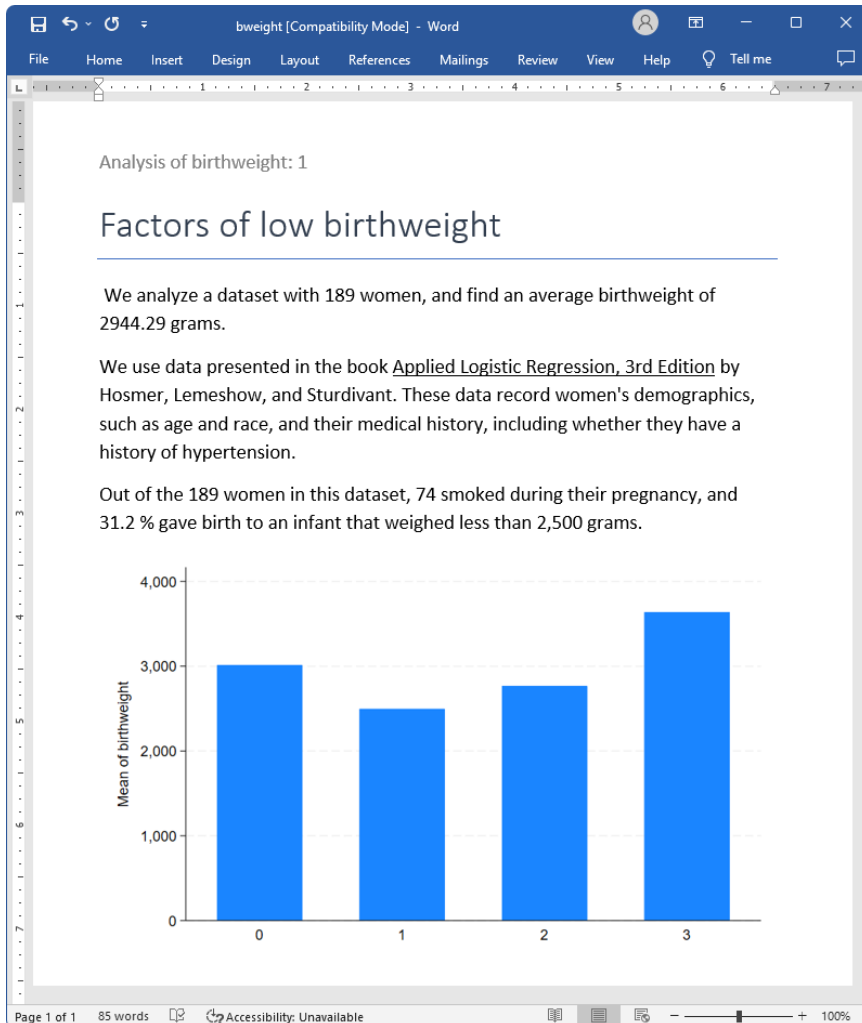


```
. graph export ptl.png
file ptl.png saved as PNG format
```

Now we use `putdocx image` to add the `.png` file to a document. There is currently no active paragraph, so we declare a new paragraph and specify the `halign()` option to center our image; by default, paragraphs are left-aligned.

```
. putdocx paragraph, halign(center)
. putdocx image pt1.png
. putdocx save bweight, append
successfully appended to "C:/mypath/bweight.docx"
```

We append our active document, which includes the two text blocks along with this graph, to the existing content of `bweight.docx`. Here is what the file contains:



Adding a bookmark to the document

When creating a large document, it can be helpful to add bookmarks to sections that you will refer to throughout the document. You can add a bookmark with `putdocx text` or with the `<<dd_docx_display>>` dynamic tag. Then, to link to the bookmark from other text, specify the `bookmarklink(bmname)` option.

► Example 6: Export a Stata graph with a bookmark

For example, suppose we plan to refer to the bar graph we created above in a later section of the document. Let's revise our code from the previous example as follows:

```
putdocx paragraph, halign(center)
putdocx image ptl.png, linebreak
putdocx text ("Figure 1. Birthweights and premature labor"), bookmark("bmark1")
```

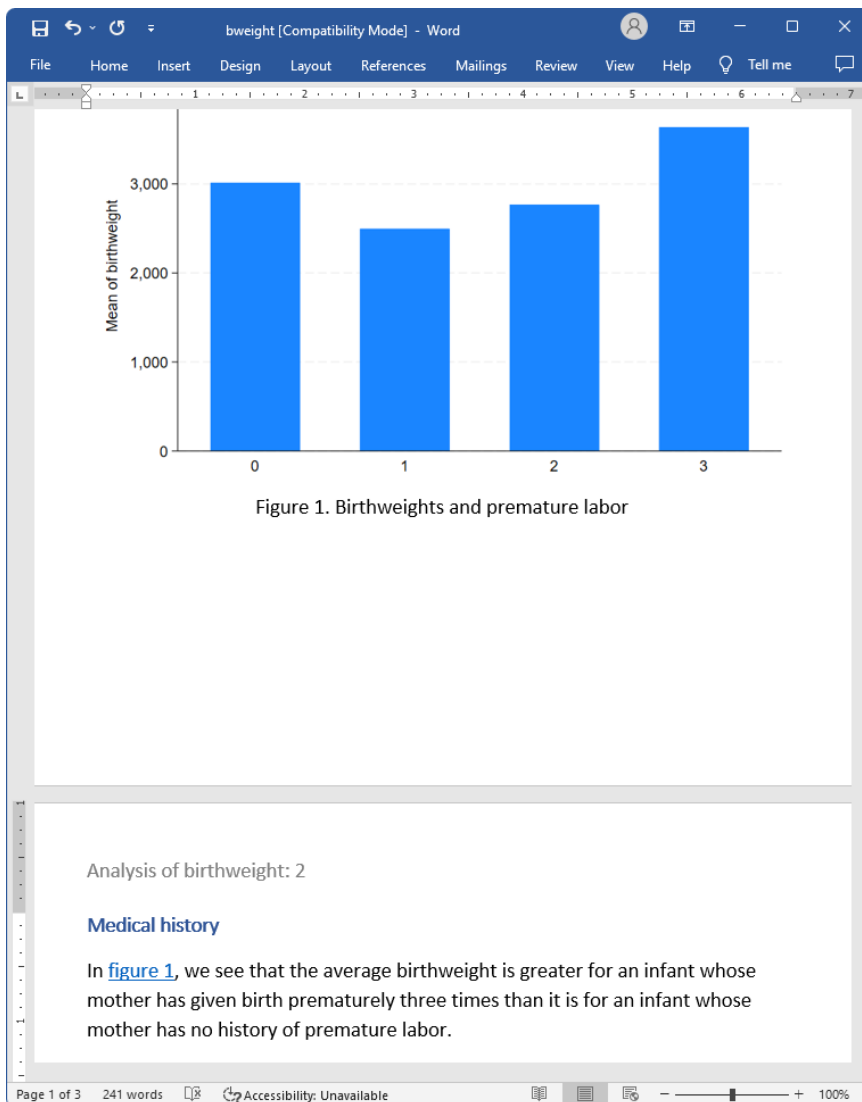
We have now added a line break after the image and labeled it as “Figure 1”. We included the `bookmark("bmark1")` option to add this text as a bookmark named `bmark1`. Below, we add a heading for a section on medical history.

```
putdocx paragraph, style(Heading1)
putdocx text ("Medical history")
```

We can point to the bookmark with the following block of text. The `<<dd_docx_display>>` dynamic tag allows us to specify text options such as `bookmarklink()` that apply to a portion of the text within the textblock. Here we link the text “figure 1” to our `bmark1` bookmark, and we save our work.

```
putdocx textblock begin
In <<dd_docx_display bookmarklink("bmark1"): "figure 1">>, we see that the
average birthweight is greater for an infant whose mother has given birth
prematurely three times than it is for an infant whose mother has no history
of premature labor.
putdocx textblock end
putdocx save bweight, append
```

Our updated document appears as follows:



We can see that “figure 1” serves as a link to the graph.

Inserting text files in the document

There may be instances where you are building a document in pieces or where you are collaborating with others to complete a project. For these situations, it may be useful to include previously created text files in your .docx file with `putdocx textfile`. You can even specify the portion of the text file that you want to include in your .docx file.

For example, let's say our classmate wrote a section about the discovery of the effects of smoking during pregnancy. We want to incorporate this in a birthweight report we have saved as `bweight2.docx`. However, the text file he sent also includes a section on hypertension, which we do not want to include. To add his text file to our document, excluding the section on hypertension, we type the following in Stata:

```
putdocx begin
putdocx textfile smoke.txt, stopat(hypertension)
putdocx save bweight2, append
```

This tells Stata that whenever it comes across a line in the text file that contains the word “hypertension”, it should exclude that line and anything that follows.

Also see

[RPT] [putdocx intro](#) — Introduction to generating Office Open XML (.docx) files

[RPT] [putdocx begin](#) — Create an Office Open XML (.docx) file

[RPT] [putdocx collect](#) — Add a table from a collection to an Office Open XML (.docx) file

[RPT] [putdocx pagebreak](#) — Add breaks to an Office Open XML (.docx) file

[RPT] [putdocx table](#) — Add tables to an Office Open XML (.docx) file

[RPT] [Appendix for putdocx](#) — Appendix for putdocx entries

[RPT] [set docx](#) — Format settings for blocks of text

Description	Quick start	Syntax	Options
Remarks and examples	Stored results	Reference	Also see

Description

`putdocx table` creates and modifies tables in the active `.docx` file. Tables may be created from several output types, including the data in memory, matrices, and estimation results.

`putdocx describe` describes a table within the active `.docx` file.

`set docx_maxtable` allows you to set the maximum number of tables allowed in `putdocx`.

Quick start

Add a table named `tbl1` with three rows and four columns to the document

```
putdocx table tbl1 = (3,4)
```

Set the content of the cell on the first row and first column of `tbl1` to be “My graph” and align the text to the right

```
putdocx table tbl1(1,1) = ("My graph"), halign(right)
```

Insert the image in `mygraph.png` into the second cell in the first column

```
putdocx table tbl1(2,1) = image(mygraph.png)
```

Format the contents in columns two through four with two decimal places

```
putdocx table tbl1(.,2/4), nformat(%5.2f)
```

Describe the contents of table `tbl1`

```
putdocx describe tbl1
```

Add a table named `tbl2` with regression results after `regress`

```
putdocx table tbl2 = etable
```

Same as above, and add a title

```
putdocx table tbl2 = etable, title("First regression")
```

Add a table of all returned scalars stored after `summarize`

```
putdocx table tbl3 = rscalars
```

Add a table of the data stored in variables `var1` and `var2` from the dataset in memory

```
putdocx table tbl4 = data(var1 var2)
```

Add a table with the contents of matrix `matrix1`, including the row and column names of the matrix

```
putdocx table tbl5 = matrix(matrix1), rownames colnames
```

Add a table with no borders to the header `report1`

```
putdocx begin, header(report1)
```

```
putdocx table tbl6 = (1,2), border(all,nil) toheader(report1)
```

Syntax

Add table to document

```
putdocx table tablename = (nrows, ncols) [ , table_options ]

putdocx table tablename = data(varlist) [ if ] [ in ] [ , varnames obsno
table_options ]

putdocx table tablename = matrix(matname) [ , nformat(%fmt) rownames colnames
table_options ]

putdocx table tablename = mata(matname) [ , nformat(%fmt) table_options ]

putdocx table tablename = etable[ (#1 #2 ... #n) ] [ , table_options ]

putdocx table tablename = returnset [ , table_options ]
```

Add content to cell

```
putdocx table tablename(i, j) = (exp) [ , cell_options exp_options ]

putdocx table tablename(i, j) = image(filename) [ , image_options cell_options ]

putdocx table tablename(i, j) = table(mem_tablename) [ , cell_options ]
```

Alter table layout

```
putdocx table tablename(i, .) , row_col_options

putdocx table tablename(. , j) , row_col_options
```

Customize format of cells or table

```
putdocx table tablename(i, j) , cell_options

putdocx table tablename(numlisti, .) , cell_fmt_options

putdocx table tablename(. , numlistj) , cell_fmt_options

putdocx table tablename(numlisti, numlistj) , cell_fmt_options

putdocx table tablename(. , .) , cell_fmt_options
```

Describe table

```
putdocx describe tablename
```

Set the maximum number of tables

```
set docx_maxtable # [ , permanently ]
```

is any number between 1 and 10,000; the default is 500.

tablename specifies the name of a table. The name must be a valid name according to Stata's naming conventions; see [U] 11.3 Naming conventions.

<i>table_options</i>	Description
<code>mentable</code>	keep table in memory rather than add it to document
<code>width(#[<i>unit</i> %] <i>matname</i>)</code>	set table width
<code>halign(<i>hvalue</i>)</code>	set table horizontal alignment
<code>indent(#[<i>unit</i>])</code>	set table indentation
<code>layout(<i>layouttype</i>)</code>	adjust column width
<code>cellmargin(<i>cmarg</i>, #[<i>unit</i>])</code>	set margins for each table cell
<code>cellspacing(#[<i>unit</i>])</code>	set spacing between adjacent cells and the edges of the table
<code>border(<i>bspec</i>)</code>	set pattern, color, and width for border
<code>headerrow(#)</code>	set number of the top rows that constitute the table header
<code>title(string[, <i>cell_fmt_options</i>])</code>	add a title to the table
<code>note(string[, <i>cell_fmt_options</i>])</code>	add a note to the table
<code>toheader(<i>hname</i>)</code>	add the table to the header <i>hname</i>
<code>tofooter(<i>fname</i>)</code>	add the table to the footer <i>fname</i>

<i>cell_options</i>	Description
<code>append</code>	append objects to current content of cell
<code>rowspan(#)</code>	merge cells vertically
<code>colspan(#)</code>	merge cells horizontally
<code>span(#₁, #₂)</code>	merge cells both horizontally and vertically
<code>linebreak[(#)]</code>	add line breaks into the cell
<i>cell_fmt_options</i>	options that control the look of cell contents

<i>exp_options</i>	Description
<code>pagenumber</code>	append the current page number to the end of <i>exp</i>
<code>totalpages</code>	append the number of total pages to the end of <i>exp</i>
<code>trim</code>	remove the leading and trailing spaces in <i>exp</i>
<code>hyperlink(<i>link</i>)</code>	add the expression as a hyperlink
<code>bookmark(<i>bmname</i>)</code>	add the expression as a bookmark named <i>bmname</i>
<code>bookmarklink(<i>bmname</i>)</code>	link the expression to bookmark <i>bmname</i>

<i>image_options</i>	Description
<code>width(#[<i>unit</i>])</code>	set image width
<code>height(#[<i>unit</i>])</code>	set image height
<code>linebreak[(#)]</code>	add line breaks after image
<code>link</code>	insert link to image file
<code>alt(<i>text</i>)</code>	alternative text to be read by voice software

<i>row_col_options</i>	Description
<code>nosplit</code>	prevent row from breaking across pages
<code>addrows(#[, before after])</code>	add # rows in specified location
<code>addcols(#[, before after])</code>	add # columns in specified location
<code>drop</code>	drop specified row or column
<code>*width(#[<i>unit</i> %])</code>	set the column width
<code>*height(#[<i>unit</i>][, atleast exact])</code>	set the row height
<i>cell_fmt_options</i>	options that control the look of cell contents

*width() may only be specified when formatting a column.

*height() may only be specified when formatting a row.

<i>cell_fmt_options</i>	Description
<code>halign(<i>hvalue</i>)</code>	set horizontal alignment
<code>valign(<i>vvalue</i>)</code>	set vertical alignment
<code>border(<i>bspec</i>)</code>	set pattern, color, and width for border
<code>shading(<i>sspec</i>)</code>	set background color, foreground color, and fill pattern
<code>nformat(%<i>fmt</i>)</code>	specify numeric format for cell text
<code>font(<i>fspec</i>)</code>	set font, font size, and font color
<code>bold</code>	format text as bold
<code>italic</code>	format text as italic
<code>*script(sub super)</code>	set subscript or superscript formatting of text
<code>strikeout</code>	strikeout text
<code>underline[(<i>upattern</i>)]</code>	underline text using specified pattern
<code>allcaps</code>	format text as all caps
<code>smallcaps</code>	format text as small caps

* May only be specified when formatting a single cell.

unit may be in (inch), pt (point), cm (centimeter), or twip (twentieth of a point). An inch is equivalent to 72 points, 2.54 centimeters, or 1440 twips. The default is in.

bspec is

bordername [, *bpattern* [, *bcolor* [, *bwidth*]]]

bordername specifies the location of the border.

bpattern is a keyword specifying the look of the border. The most common patterns are single, dashed, dotted, and double. The default is single. For a complete list of border patterns, see [Border patterns](#) of [RPT] [Appendix for putdocx](#). To remove an existing border, specify nil as the *bpattern*.

bcolor specifies the border color.

bwidth is defined as #[*unit*] and specifies the border width. The default border width is 0.5 points. If # is specified without the optional *unit*, inches is assumed. *bwidth* may be ignored if you specify a width larger than that allowed by the program used to view the .docx file. We suggest using 12 points or less or an equivalent specification.

sspec is

bcolor [, *fgcolor* [, *fpattern*]]

bcolor specifies the background color.

fgcolor specifies the foreground color. The default foreground color is black.

fpattern specifies the fill pattern. The most common fill patterns are `solid` for a solid color (determined by *fgcolor*), `pct25` for 25% gray scale, `pct50` for 50% gray scale, and `pct75` for 75% gray scale. A complete list of fill patterns is shown in [Shading patterns](#) of [RPT] [Appendix for putdocx](#).

fspec is

fontname [, *size* [, *color*]]

fontname may be any valid font installed on the user's computer. If *fontname* includes spaces, then it must be enclosed in double quotes.

size is a numeric value that represents font size measured in points. The default is 11.

color sets the text color.

bcolor, *bcolor*, *fgcolor*, and *color* may be one of the colors listed in [Colors](#) of [RPT] [Appendix for putdocx](#); a valid RGB value in the form `### #`, for example, 171 248 103; or a valid RRGGBB hex value in the form `#####`, for example, ABF867.

Output types for tables

The following output types are supported when creating a new table using `putdocx table tablename:`

`(nrows, ncols)` creates an empty table with *nrows* rows and *ncols* columns. Microsoft Word allows a maximum of 63 columns in a table.

`data(varlist)` adds the current Stata dataset in memory as a table to the active document. *varlist* contains a list of the variable names from the current dataset in memory.

`matrix(matname)` adds a [matrix](#) called *matname* as a table to the active document.

`mata(matname)` adds a Mata [matrix](#) called *matname* as a table to the active document.

`etable[(#1 #2 ... #n)]` adds an automatically generated table to the active document. The table may be derived from the coefficient table of the last estimation command, from the table of margins after the last [margins](#) command, or from the table of results from one or more models displayed by [estimates table](#).

If the estimation command outputs $n > 1$ coefficient tables, the default is to add all tables and assign the corresponding table names *tablename1*, *tablename2*, ..., *tablename_n*. To specify which tables to add, supply the optional numlist to `etable`. For example, to add only the first and third tables from the estimation output, specify `etable(1 3)`. A few estimation commands do not support the `etable` output type. See [Unsupported estimation commands](#) of [RPT] [Appendix for putdocx](#) for a list of estimation commands that have displayed output that is not supported by `putdocx`.

returnset exports a group of Stata [return](#) values to a table in the active document. It is intended primarily for use by programmers and by those who want to do further processing of their exported results in the active document. *returnset* may be one of the following:

<i>returnset</i>	Description
<u>escalars</u>	all ereturned scalars
<u>rscalars</u>	all returned scalars
<u>emacros</u>	all ereturned macros
<u>rmacros</u>	all returned macros
<u>ematrices</u>	all ereturned matrices
<u>rmatrices</u>	all returned matrices
<u>e*</u>	all ereturned scalars, macros, and matrices
<u>r*</u>	all returned scalars, macros, and matrices

The following output types are supported when adding content to an existing table using `putdocx table tablename(i, j)`:

[\(exp\)](#) writes a valid Stata expression to a cell; see [\[U\] 13 Functions and expressions](#).

`image(filename)` adds a Portable Network Graphics (.png), Joint Photographic Experts Group (.jpg), Device-Independent Bitmap (.dib), Enhanced Metafile (.emf), Bitmap (.bmp), Tagged Image File Format (.tif), or Scalable Vector Graphics (.svg) file to the table cell. If *filename* contains spaces, it must be enclosed in double quotes.

`table(mem_tablename)` adds a previously created table, identified by *mem_tablename*, to the table cell.

The following combinations of `tablename(numlisti, numlistj)` (see [\[U\] 11.1.8 numlist](#) for valid specifications) can be used to format a cell or range of cells in an existing table:

`tablename(i, j)` specifies the cell on the *i*th row and *j*th column.

`tablename(i, .)` and `tablename(numlisti, .)` specify all cells on the *i*th row or on the rows identified by *numlist_i*.

`tablename(. , j)` and `tablename(. , numlistj)` specify all cells in the *j*th column or in the columns identified by *numlist_j*.

`tablename(. , .)` specifies the whole table.

Options

Options are presented under the following headings:

[table_options](#)
[cell_options](#)
[row_col_options](#)
[cell_fmt_options](#)
[exp_options](#)
[image_options](#)
[Option for set docx_maxtable](#)

table_options

`memtable` specifies that the table be created and held in memory instead of being added to the active document. By default, the table is added to the document immediately after it is created. This option is useful if the table is intended to be added to a cell of another table or to be used multiple times later.

`width(#[unit | %] [matname])` sets the table width. `width(#)`, `width(#unit)`, or `width(#%)` may be specified with `width(matname)`.

`width(#[unit | %])` sets the width based on a specified value. `#` may be an absolute width or a percentage of the default table width, which is determined by the page width of the document. For example, `width(50%)` sets the table width to 50% of the default table width. The default is `width(100%)`.

When specifying the table width as a percentage, it cannot be greater than 100%.

`width(matname)` sets the table width based on the dimensions specified in the Stata matrix *matname*, which has contents in the form of (*#*₁, *#*₂, ..., *#*_{*n*}) to denote the percentage of the default table width for each column. *n* is the number of columns of the table, and the sum of *#*₁ to *#*_{*n*} must be equal to 100.

`halign(hvalue)` sets the horizontal alignment of the table within the page. *hvalue* may be `left`, `right`, or `center`. The default is `halign(left)`.

`indent(#[unit])` specifies the table indentation from the left margin of the current document.

`layout(layouttype)` adjusts the column width of the table. *layouttype* may be `fixed`, `autofitwindow`, or `autofitcontents`. `fixed` means the width is the same for all columns in the table. When `autofitwindow` is specified, the column width automatically resizes to fit the window. When `autofitcontents` is specified, the table width is determined by the overall table layout algorithm, which automatically resizes the column width to fit the contents. The default is `layout(autofitwindow)`.

`cellmargin(cmarg, #[unit])` sets the cell margins for table cells. *cmarg* may be `top`, `bottom`, `left`, or `right`. This option may be specified multiple times in a single command to accommodate different margin settings.

`cellspacing(#[unit])` sets the spacing between adjacent cells and the edges of the table.

`border(bordername [, bpattern [, bcolor [, bwidth]])` adds a single border in the location specified by *bordername*, which may be `start`, `end`, `top`, `bottom`, `insideH` (inside horizontal borders), `insideV` (inside vertical borders), or `all`. Optionally, you may change the pattern, color, and width for the border by specifying *bpattern*, *bcolor*, and *bwidth*.

This option may be specified multiple times in a single command to accommodate different border settings. If multiple `border()` options are specified, they are applied in the order specified from left to right.

`headerrow(#)` sets the top *#* rows to be repeated as header rows at the top of each page on which the table is displayed. This setting has a visible effect only when the table crosses multiple pages.

`varnames` specifies that the variable names be included as the first row in the table when the table is created using the dataset in memory. By default, only the data values are added to the table.

`obsno` specifies that the observation numbers be included as the first column in the table when the table is created using the dataset in memory. By default, only the data values are added to the table.

`nformat(%fmt)` specifies the numeric format to be applied to the source values when creating the table from a Stata or Mata matrix. The default is `nformat(%12.0g)`.

`rownames` specifies that the row names of the Stata matrix be included as the first column in the table. By default, only the matrix values are added to the table.

`colnames` specifies that the column names of the Stata matrix be included as the first row in the table. By default, only the matrix values are added to the table.

`title(string[, cell_fmt_options])` inserts a row without borders above the current table. The added row spans all the columns of the table and contains *string* as text. The added row shifts all other table contents down by one row. You should account for this when referencing table cells in subsequent commands.

This option may be specified multiple times in a single command to add titles on new lines within the same cell. Title text is inserted in the order it was specified from left to right. Each title can be customized using *cell_fmt_options*.

`note(string[, cell_fmt_options])` inserts a row without borders to the bottom of the table. The added row spans all the columns of the table. This option may be specified multiple times in a single command to add notes on new lines within the same cell. Note that text is inserted in the order it was specified from left to right. Each note can be customized using *cell_fmt_options*.

`toheader(hname)` specifies that the table be added to the header *hname*. The table will not be added to the body of the document.

`tofooter(fname)` specifies that the table be added to the footer *fname*. The table will not be added to the body of the document.

cell_options

`append` specifies that the new content for the cell be appended to the current content of the cell. If `append` is not specified, then the current content of the cell is replaced by the new content.

`rowspan(#)` sets the specified cell to span vertically *#* cells downward. If the span exceeds the total number of rows in the table, the span stops at the last row.

`colspan(#)` sets the specified cell to span horizontally *#* cells to the right. If the span exceeds the total number of columns in the table, the span stops at the last column.

`span(#1 , #2)` sets the specified cell to span *#*₁ cells downward and span *#*₂ cells to the right.

`linebreak[(#)]` specifies that one or *#* line breaks be added after the text, the image, or the table within the cell.

row_col_options

`nosplit` specifies that row *i* not split across pages. When a table row is displayed, a page break may fall within the contents of a cell on the row, causing the contents of that cell to be displayed across two pages. `nosplit` prevents this behavior. If the entire row cannot fit on the current page, the row will be moved to the start of the next page.

`addrows(#[, before|after])` adds *#* rows to the current table before or after row *i*. If *before* is specified, the rows are added before the specified row. By default, rows are added after the specified row.

`addcols(#[, before | after])` adds # columns to the current table to the right or the left of column *j*. If *before* is specified, the columns are added to the left of the specified column. By default, the columns are added after, or to the right of, the specified column.

`drop` deletes row *i* or column *j* from the table.

`width(#[unit | %])` sets the width of column *j*. # may be an absolute width or a percentage of the table width. For example, `width(50%)` sets the column width to take up 50% of the table width.

When specifying the column width as a percentage, it cannot be greater than 100%.

`height(#[unit] [, atleast | exact])` sets the height of row *i*.

atleast means the row will have the minimum height of the specified value. This is the default.

exact means the row will have the exact height of the specified value.

cell_fmt_options

`halign(hvalue)` sets the horizontal alignment of the specified cell or of all cells in the specified row, column, or range. *hvalue* may be *left*, *right*, or *center*. The default is `halign(left)`.

`valign(vvalue)` sets the vertical alignment of the specified cell or of all cells in the specified row, column, or range. *vvalue* may be *top*, *bottom*, or *center*. The default is `valign(top)`.

`border(bordername [, bpattern [, bcolor [, bwidth]]])` adds a single border to the specified cell or to all cells in the specified row, column, or range in the given location. *bordername* may be *start*, *end*, *top*, *bottom*, or *all*. Optionally, you may change the pattern, color, and width for the border by specifying *bpattern*, *bcolor*, and *bwidth*.

This option may be specified multiple times in a single command to accommodate different border settings. If multiple `border()` options are specified, they are applied in the order specified from left to right.

`shading(bgcolor [, fgcolor [, fpattern]])` sets the background color, foreground color, and fill pattern for the specified cell or for all cells in the specified row, column, or range.

`nformat(%fmt)` applies the Stata numeric format %*fmt* to the text within the specified cell or within all cells in the specified row, column, or range. This setting only applies when the content of the cell is a numeric value.

`font(fontname [, size [, color]])` sets the font, font size, and font color for the current text within the specified cell or within all cells in the specified row, column, or range. The font size and font color may be specified individually without specifying *fontname*. Use `font("", size)` to specify font size only. Use `font("", "", color)` to specify font color only. For both cases, the default font will be used.

`bold` applies bold formatting to the current text within the specified cell or within all cells in the specified row, column, or range.

`italic` applies italic formatting to the current text within the specified cell or within all cells in the specified row, column, or range.

`script(sub | super)` changes the script style of the current text. `script(sub)` makes the text a subscript. `script(super)` makes the text a superscript. `script()` may only be specified when formatting a single cell.

`strikeout` adds a strikeout mark to the current text within the specified cell or within all cells in the specified row, column, or range.

`underline` adds an underline to the current text within the specified cell or within all cells in the specified row, column, or range. By default, a single underline is used. `underline(upattern)` can be used to change the format of the line, where *upattern* may be any of the patterns listed in [Underline patterns](#) of [RPT] [Appendix for putdocx](#). The most common patterns are double, dash, and none.

`allcaps` uses capital letters for all letters of the current text within the specified cell or within all cells in the specified row, column, or range.

`smallcaps` uses capital letters for all letters of the current text within the specified cell or within all cells in the specified row, column, or range. `smallcaps` uses larger capitals for uppercase letters and smaller capitals for lowercase letters.

exp_options

`pagenumber` specifies that the current page number be appended to the end of the new content for the cell. `pagenumber` applies only to tables being added to a header or footer and cannot be combined with `totalpages`.

`totalpages` specifies that the number of total pages be appended to the end of the new content for the cell. `totalpages` applies only to tables being added to a header or footer and cannot be combined with `pagenumber`.

`trim` removes the leading and trailing spaces in the expression to be added to the table cell.

`hyperlink(link)` adds the expression as a hyperlink to the webpage address specified in *link*.

`bookmark(bmname)` adds the expression as a bookmark named *bmname*. You can link to this bookmark from other text within a paragraph or from an expression in a table cell by using the `bookmarklink(bmname)` option.

`bookmarklink(bmname)` adds the expression as a link to the bookmark named *bmname*. Note that bookmark links can point only to bookmarks in the current document. Additionally, these links cannot point to bookmarks in documents to which you will be appending.

image_options

`width(#unit)` sets the width of the image. If the width is larger than the width of the cell, then the width is used. If `width()` is not specified, then the default size is used; the default is determined by the image information and the width of the cell.

`height(#unit)` sets the height of the image. If `height()` is not specified, then the height of the image is determined by the width and the aspect ratio of the image.

`linebreak[(#)]` specifies that one or *#* line breaks be added after the new image.

`link` specifies that a link to the image *filename* be inserted into the document. If the image is linked, then the referenced file must be present so that the document can display the image.

`alt(text)` specifies alternative text for the image to be read by voice software.

Option for set docx_maxtable

permanently specifies that, in addition to making the change right now, the settings be remembered and become the default settings when you invoke Stata.

Remarks and examples

Remarks are presented under the following headings:

Introduction
Creating basic tables
Exporting summary statistics
Exporting estimation results
Creating advanced tables
Customizing headers and footers with tables

Introduction

The suite of putdocx commands makes it easy to export summary statistics, estimation results, data, and images in neatly formatted tables. There are different output types available to export a whole coefficient table, matrix, or dataset in a single step. Alternatively, you can create a table by specifying the dimensions and then gradually inserting contents, such as text, images, and stored results. When you create a table, you specify a name for it, which allows you to make further modifications to its contents. You can customize each cell or apply specific formatting to a range of cells with row and column indexes. The variety of formatting options allows you to export a table complete with a title, notes, and a repeating header for tables that span multiple pages.

In the following sections, we demonstrate how to create a variety of tables, ranging from small tables without formatting to more complex, customized tables. In each example, the Word document we create includes only one table, but the putdocx table commands we use work in the same way when you insert these tables into a larger document.

Creating basic tables

When exporting only a few statistics and results, you can create a table from scratch—first specifying the size of the table and then adding content one cell at a time. With this method, it is easy to control the location of each element within the table.

▷ Example 1: Export a basic table

Suppose we want to export a table with just the mean, minimum, and maximum miles per gallon for 1978 automobiles to a .docx file. First, we open the dataset and create an active document.

```
. sysuse auto, clear
. putdocx begin
```

Now we create our table with the necessary dimensions. We are exporting three statistics that will appear in one column. We need to allot another column to label these statistics. Therefore, we create a table named mpgstats with three rows and two columns, and we add a title. By default, the table width is set at 100%, but because our content is rather short, we set the width to 40% of the default. Otherwise, the table would take up a large portion of our document and look very empty.

```
. putdocx table mpgstats = (3,2), title(MPG summary statistics) width(40%)
. putdocx describe mpgstats
```

Table name	mpgstats
No. of rows	4
No. of cols	2

When we describe our table, it reports four rows instead of the three we specified. When we add a title, it is included as the first row of the table without any borders. With our table in place, we now run `summarize mpg` and view the list of stored results:

```
. summarize mpg
```

Variable	Obs	Mean	Std. dev.	Min	Max
mpg	74	21.2973	5.785503	12	41

```
. return list
```

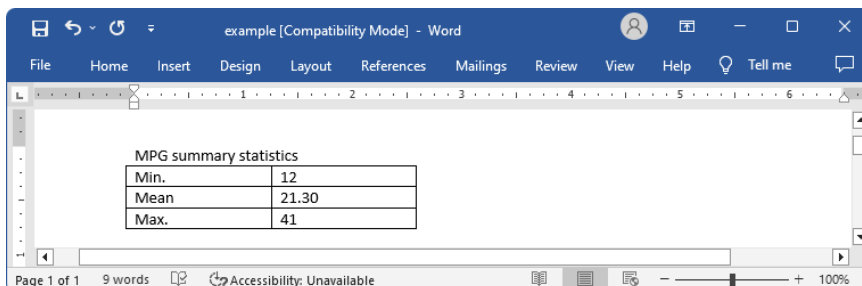
scalars:

```
      r(N) = 74
r(sum_w) = 74
r(mean) = 21.2972972972973
r(Var) = 33.47204738985561
r(sd) = 5.785503209735141
r(min) = 12
r(max) = 41
r(sum) = 1576
```

All the statistics we want are stored in `rclass` scalars. We can directly refer to stored results with `putdocx table`. We begin filling in the table by putting the label for the minimum in the second row and first column, `mpgstats(2,1)`. Then we add the value of the minimum in the cell next to it, `mpgstats(2,2)`, by referring to the `r(min)` scalar shown above. Similarly, we add the mean on row 3 and the maximum on row 4.

```
. putdocx table mpgstats(2,1) = ("Min.")
. putdocx table mpgstats(2,2) = (r(min))
. putdocx table mpgstats(3,1) = ("Mean")
. putdocx table mpgstats(3,2) = (r(mean)), nformat(%5.2f)
. putdocx table mpgstats(4,1) = ("Max.")
. putdocx table mpgstats(4,2) = (r(max))
. putdocx save example
successfully created "C:/mypath/example.docx"
```

Because the mean of `mpg` is reported with a lot of accuracy, we specify a numeric formatting. After saving our work, the `example.docx` file contains the following:



Exporting summary statistics

In the example above, we exported just a few summary statistics by filling in the content of each cell in a table. That method would be tedious if we wanted to export several results. Another option is to create a dataset of summary statistics and export the entire dataset to a table, as shown in the example below.

► Example 2: Export a dataset of summary statistics

Suppose that now we want to report the summary statistics separately for foreign and domestic automobiles. We create a new active document, and then we use the `statsby` command to collect the number of observations along with the minimum, mean, and maximum of `mpg` for each group. Because `statsby` creates a new dataset that overwrites the dataset in memory, we need to preserve the dataset and then restore it after we have finished exporting the data.

```
. putdocx begin
. preserve
. statsby Obs=r(N) Min=r(min) Mean=r(mean) Max=r(max), by(foreign):
> summarize mpg
(running summarize on estimation sample)

      Command: summarize mpg
             Obs: r(N)
             Min: r(min)
             Mean: r(mean)
             Max: r(max)
             By: foreign

Statsby groups:
..
```

We want the variable names to serve as column titles, so we rename `foreign` to `Origin`. Then we export the data in memory as a table by specifying in `data()` the variable names `Origin`, `Obs`, `Min`, `Mean`, and `Max`. In this case, we use the table name `tbl1`. The order of the variable names in the list determines the column order in the table.

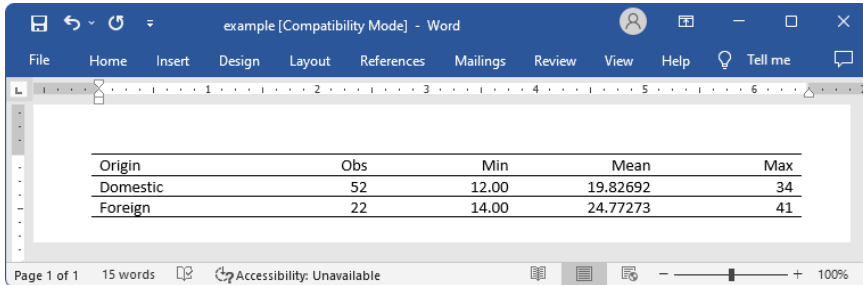
```
. rename foreign Origin
. putdocx table tbl1 = data("Origin Obs Min Mean Max"), varnames
> border(start, nil) border(insideV, nil) border(end, nil)
```

By default, the exported table includes single borders around all cells. We use the `border()` option to remove all vertical borders from the table.

We further customize the table by setting the text of the cells on the second through the fifth columns to be right-aligned instead of the default left alignment. To format all the cells in these columns, we specify the row index as “.” and the column indexes as 2/5 in the $(numlist_i, numlist_j)$ specification for a table. Also, we can choose to display only two digits after the decimal for the means. For this purpose, we specify a range for the row index of column three.

```
. putdocx table tbl1(., 2/5), halign(right)
. putdocx table tbl1(2/3, 3), nformat(%5.2f)
. putdocx save example, replace
successfully replaced "C:/mypath/example.docx"
```

We save our work under the filename `example.docx`, which contains the following:



Afterward, we restore the dataset.

```
. restore
```

Exporting estimation results

One of the primary uses of putdocx table is to export estimation results. Suppose we fit a linear regression model of mpg as a function of the car's gear ratio (gear_ratio), turning radius (turn), and whether the car is of foreign origin (foreign) using `regress`.

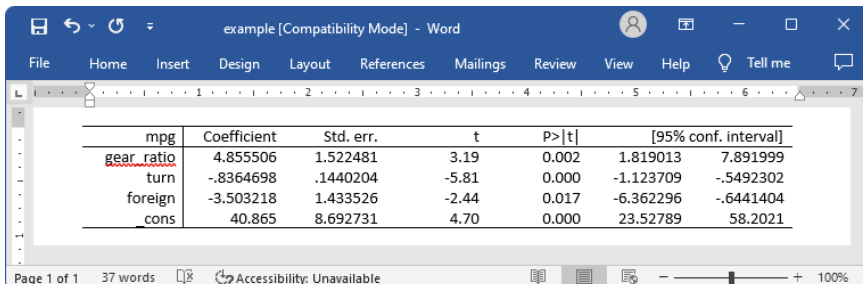
```
. regress mpg gear_ratio turn foreign, noheader
```

mpg	Coefficient	Std. err.	t	P> t	[95% conf. interval]	
gear_ratio	4.855506	1.522481	3.19	0.002	1.819013	7.891999
turn	-.8364698	.1440204	-5.81	0.000	-1.123709	-.5492302
foreign	-3.503218	1.433526	-2.44	0.017	-6.362296	-.6441404
_cons	40.865	8.692731	4.70	0.000	23.52789	58.2021

We want to add these regression results to the document. For most estimation commands, we can use the `etable` output type to add the elements of the displayed coefficient table with a single command. As always, we must first create an active document

```
. putdocx begin
. putdocx table reg = etable
. putdocx save example, replace
successfully replaced "C:/mypath/example.docx"
```

After saving the document and overwriting our previous work, the table is displayed as follows:



If we instead want to display only a few of the reported statistics, we can customize the table accordingly. In [example 3](#), we select a subset of the results and format them.

► Example 3: Export selected estimation results

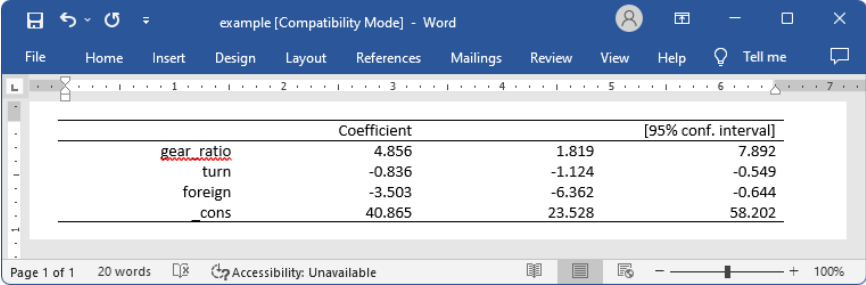
Suppose we want to export only the point estimates and confidence intervals from the table above; we can also use `putdocx table` to remove the components that we do not want.

First, we create an active document with a table that contains the estimation results from `regress`, `tbl2`. Although the default is for the table to occupy 100% of the width as determined by the page width of the document, once we drop columns the table width will be adjusted. Therefore, we specify the `width` option to ensure that the table occupies the full page width of the document. Next, we want to remove the third through the fifth columns. To drop the fifth column, we specify `tbl2(.,5)` followed by the `drop` option. We work from right to left, dropping column five before four, because in this manner, the column numbers to the left of the newly dropped column do not change. Equivalently, we could have dropped the third column three times, because each time we drop it, the previous fourth column becomes the third.

Once we have only the desired statistics, we customize the header row by erasing the text “mpg” from the first column. We then format our table by removing the border on the right side of the first column. To do this, we specify `nil` as the border pattern for the right border. And finally, we format all estimates, in what will now be columns two through four, to have three decimal places by specifying the column indexes as a range.

```
. putdocx begin
. putdocx table tbl2 = etable, width(100%)
. putdocx table tbl2(.,5), drop //drop p-value column
. putdocx table tbl2(.,4), drop //drop t column
. putdocx table tbl2(.,3), drop //drop SE column
. putdocx table tbl2(1,1) = ("") // erase the content of first cell "mpg"
. putdocx table tbl2(.,1), border(right, nil)
. putdocx table tbl2(.,2/4), nformat(%9.3f)
. putdocx save example, replace
successfully replaced "C:/mypath/example.docx"
```

After saving our work, the table appears in the `example.docx` file as follows:



	Coefficient	[95% conf. interval]	
<u>gear_ratio</u>	4.856	1.819	7.892
turn	-0.836	-1.124	-0.549
foreign	-3.503	-6.362	-0.644
_cons	40.865	23.528	58.202

In this example, because we wanted to use the same number of decimals for all estimates in our table and because we are using the `etable` output type, we could have preemptively set the format with our `regress` command. The modified version of the command is

```
. regress mpg gear_ratio turn foreign, noheader cformat(%9.3f)
```

This avoids the need to issue a separate formatting command.

The `etable` output type also works after `estimates table`, and you may find it easier to build a table of selected estimates prospectively. See [example 3](#) in [\[R\] estimates table](#) for an illustration.

Creating advanced tables

Sometimes, we need to create highly customized tables with complex layouts. `putdocx table` has many features that will allow you to create and incorporate sophisticated tables in your documents.

First, we demonstrate how to create a table from a matrix. While this method works with any Stata matrix, we will demonstrate by using a matrix of stored results from `regress`. See [\[U\] 14 Matrix expressions](#) for information on working with Stata matrices.

Exporting a matrix may be helpful when working with one of the few estimation commands that do not support the `etable` output type. After running any of these commands, matrices of stored results can be exported. For a list of estimation commands that do not support the `etable` output type, see [Unsupported estimation commands](#) of [\[RPT\] Appendix for putdocx](#).

In the following examples, we demonstrate how to use `putdocx table` to export tabulation results and create a customized regression table. These examples demonstrate advanced uses of `putdocx table`, which are often helpful. However, for tabulations and regression tables, you can also customize results using `table` or `collect` and export the results to your `.docx` file using `putdocx collect`. See [\[RPT\] putdocx collect](#) for more information on this often simpler solution.

► Example 4: Export selected estimation results from a matrix

To illustrate the basic use of matrix manipulation of stored results to create a table, we re-create the simple estimation table from [example 3](#). The displayed results returned by `regress` are stored in the matrix `r(table)`, which can be viewed by typing `matrix list r(table)`.

```
. matrix list r(table)
r(table) [9,4]
      gear_ratio      turn      foreign      _cons
      b      4.8555057      -.83646975      -3.5032183      40.864996
      se      1.5224812      .14402036      1.4335262      8.6927313
      t      3.1892057      -5.8079965      -2.4437769      4.7010537
pvalue      .0021348      1.704e-07      .01705791      .00001258
      ll      1.8190127      -1.1237093      -6.3622962      23.527891
      ul      7.8919987      -.5492302      -.64414044      58.202102
      df      70      70      70      70
      crit      1.9944371      1.9944371      1.9944371      1.9944371
      eform      0      0      0      0
```

First, we create the matrix `rtable` as the transpose of `r(table)` because we want to see the variable names in rows. The point estimates and confidence intervals in the regression table can be extracted from the matrix `rtable`. We can extract columns 1, 5, and 6 from `rtable`, combine them, and assign them to another new matrix, `r_table`.

```
. matrix rtable = r(table)'
. matrix r_table = rtable[1...,1], rtable[1...,5..6]
```

Then we export `r_table` to a new active document as a table with the name `tbl3`.

```
. putdocx begin
. putdocx table tbl3 = matrix(r_table), nformat(%9.3f) rownames colnames
> border(start, nil) border(insideH, nil) border(insideV, nil) border(end, nil)
```

In this table, all values imported from the matrix have been formatted as %9.3f. In addition, the row and column names of the matrix `r_table` are included as the first column and first row of the table. We keep only the top and bottom borders of the table by specifying `nil` for the leading edge border (`start`), trailing edge border (`end`), inside horizontal edges border (`insideH`), and inside vertical edges border (`insideV`).

The column names from the matrix may not be exactly what we want; we can modify them by customizing the corresponding cells. We can reset the contents and the horizontal alignment of the cells on the first row to give the columns new titles.

```
. putdocx table tbl3(1,2) = ("Coef."), halign(right)
. putdocx table tbl3(1,3) = ("[95% conf. interval]"), halign(right) colspan(2)
```

Afterward, we add back the bottom border of the first row and right-align the cells on the rest of the rows by specifying the row range as two through five.

```
. putdocx table tbl3(1,.), border(bottom)
. putdocx table tbl3(2/5,.), halign(right)
. putdocx save example, replace
successfully replaced "C:/mypath/example.docx"
```

Our final table will be identical to that shown in [example 3](#).



► Example 5: Export tabulation results

For commands that do not support the `etable` output type, we can use matrices to export the results. For example, if we want to include a table containing the cross-tabulation of repair records and car origins in our document, we can export this information from a matrix created by the `tabulate` command. To make our table presentable, we begin by adding a value label to the `rep78` variable:

```
. label define repairs 1 "Poor" 2 "Fair" 3 "Average" 4 "Good" 5 "Excellent"
. label values rep78 repairs
```

Next we `tabulate` our variables of interest, storing the frequencies in a matrix named `repairs`, and create an active document. This matrix does not automatically contain the column totals, so we add a column at the end of the table and label it accordingly.

```
. tabulate foreign rep78, matcell(repairs)
```

Car origin	Repair record 1978					Total
	Poor	Fair	Average	Good	Excellent	
Domestic	2	8	27	9	2	48
Foreign	0	0	3	9	9	21
Total	2	8	30	18	11	69

```
. putdocx begin
. putdocx table tbl4 = matrix(repairs), rownames colnames
. putdocx table tbl4(.,6), addcols(1)
. putdocx table tbl4(1,7) = ("Total")
```

What follows is a loop to fill in the labels we have attached to `rep78`, as well as the labels for the car origin. We begin by looping over the two values of foreign, 0 and 1. We use the extended macro function `label` to store the labels for each category in the macro `'r1b1'`. Within that loop,

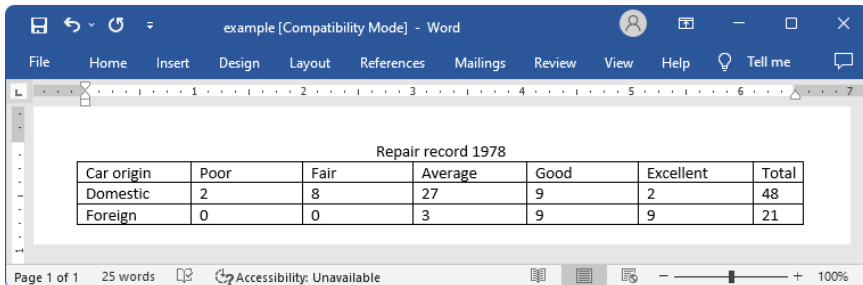
we also loop over the five columns for repair records. To calculate the row totals, we create a macro `freq'row'_val` that points to the frequency of the given cell. We store the cumulative frequencies in the macro `cumul_freq'row'`, which is incremented in each run of the loop.

```
. forvalues i=0/1 {
2.   local rlbl: label (foreign) 'i'
3.   local row = 'i' + 2
4.   putdocx table tbl4('row',1) = ("'rlbl'")
5.
.   forvalues j=1/5 {
6.     local clbl: label (rep78) 'j'
7.     local cstart = 'j' + 1
8.     putdocx table tbl4(1,'cstart') = ("'clbl'")
9.     local freq'row'_val = repairs['i'+1,'j']
10.    local cumul_freq'row' = 'cumul_freq'row' + 'freq'row'_val'
11.    putdocx table tbl4('row',7) = ("'cumul_freq'row'")
12.  }
13. }
```

Once all the frequencies have been exported, we insert a row at the top of the table to provide a title. Additionally, we provide a label for the first column.

```
. putdocx table tbl4(1,.), addrows(1,before)
. putdocx table tbl4(1,1)= ("Repair record 1978"), colspan(7) halign(center)
> border(left,nil) border(top,nil) border(right,nil)
. putdocx table tbl4(2,1)= ("Car origin")
. putdocx save example, replace
successfully replaced "C:/mypath/example.docx"
```

We save our work, overwriting the existing content of our document. The table appears in the `example.docx` file as follows:

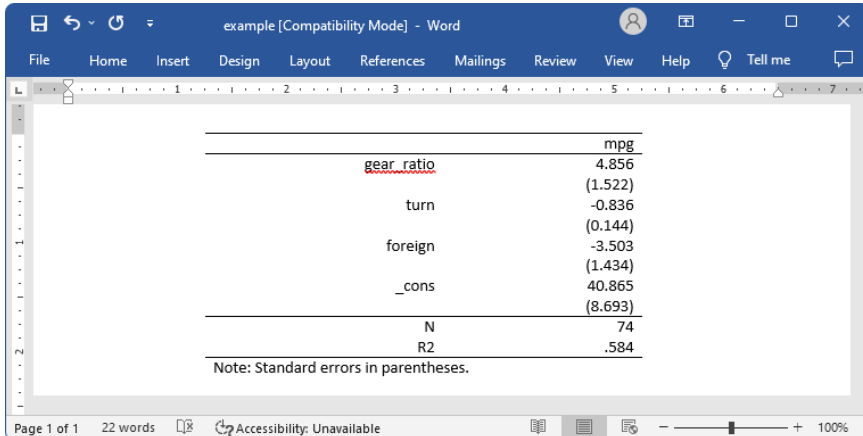


Car origin	Poor	Fair	Average	Good	Excellent	Total
Domestic	2	8	27	9	2	48
Foreign	0	0	3	9	9	21

► Example 6: Create a table dynamically

In [example 3](#), we created a customized table by dropping the columns we did not want after exporting the whole coefficient table. You may have another layout in mind, or you may be adding statistics other than those included in the coefficient table. For these cases, you can create a table dynamically: start with a simple table, and then add rows or columns to it gradually.

Returning to the regression results from [example 3](#), suppose we want to add an estimation table to the document containing the point estimates, their standard errors, the total number of observations, and the coefficient of determination. The estimation table looks like the following:



To create our table of estimation results, we first create an active document. Then we add a 1×2 table with no borders and fill the single row with the dependent variable name, mpg. We also set the table width to be 4 inches and put the table in the center of the document. We add a note stating that standard errors are in parentheses.

```
. putdocx begin
. putdocx table tbl5 = (1,2), border(all, nil) width(4) halign(center)
> note(Note: Standard errors in parentheses.)
. putdocx table tbl5(1,1)="mpg", halign(right) colspan(2) border(top)
> border(bottom)
```

Notice that each regressor in the model takes up 2 rows and 2 columns in the table, which means 4 cells. The cell in the first row and first column contains the variable name. The cell in the first row and second column contains the point estimate. The cell in the second row and second column contains the standard errors. Based on this logic, we add two rows for each regressor at the end of the table every time, and we fill in the content and format for each cell one by one.

```
. local row 1
. local vari 1
. foreach x in gear_ratio turn foreign _cons {
2.     putdocx table tbl5('row',.), addrows(2)
3.
.     local b: display %9.3f rtable['vari',1]
4.     local se: display %9.3f rtable['vari',2]
5.     local ++vari
6.
.     local ++row
7.     putdocx table tbl5('row',1) = ("x"), halign(right)
8.     putdocx table tbl5('row',2) = ("b"), halign(right)
9.     local ++row
10.    local se = strtrim("se")
11.    putdocx table tbl5('row',2) = ("se"), halign(right)
12. }
```

Afterward, we add two more rows to the end of the table for the number of observations and the coefficient of determination.

```
. putdocx table tbl5('row',.), addrows(2)
```

We then add each statistic to the table.

```
. local ++row
. putdocx table tbl5('row',1) = ("N"), border(top) halign(right)
. putdocx table tbl5('row',2) = (e(N)), border(top) halign(right)
. local ++row
. local r2: display %9.3f e(r2)
. putdocx table tbl5('row',1) = ("R2"), halign(right) border(bottom)
. putdocx table tbl5('row',2) = ('r2'), halign(right) border(bottom)
. putdocx save example, replace
successfully replaced "C:/mypath/example.docx"
```

We add a bottom border to the last row to separate the note from the rest of the table and save our work.



Aside from estimation results and summary statistics, you may want to export images to a .docx file. Images can be exported simply by appending them to an active paragraph, or by inserting them in a table. The same formatting options are available regardless of how you export an image. For example, you can specify the height and width dimensions, or include a link to the image. Of course, when inserting an image in a table, the size is constrained to the dimensions of the given cell. Below, we include an example of how to insert a series of images within a table, complete with informative notes.

► Example 7: Nesting images in a table

We might want to add various images to the document and align them side by side, row by row, or both. To be more clear, we use an example to illustrate this purpose. In this example, we use another dataset—the Second National Health and Nutrition Examination Survey (NHANES II) ([McDowell et al. 1981](#)).

First, we fit a three-way full factorial model of systolic blood pressure on age group, sex, and body mass index (BMI). Then, we estimate the predictive margins for each combination of agegrp and sex at levels of BMI from 10 through 40 at intervals of 10 and graph the results.

```
. use https://www.stata-press.com/data/r19/nhanes2, clear
. regress bpsystol agegrp##sex##c.bmi
(output omitted)
. forvalues v=10(10)40 {
2.     margins agegrp, over(sex) at(bmi='v')
3.     marginsplot
4.     graph export bmi'v'.png
5. }
(output omitted)
```

Now, we want to add those four plots into the document, requiring that the margins plots for bmi=10 and bmi=20 lay side by side on top of the other two side-by-side margins plots for bmi=30 and bmi=40. It is also required that each plot have a subtitle indicating the level of BMI and that the final figure have a title. This complicated layout can be accomplished using putdocx table.

We create an active document and add a 2×2 table. We caption our table by using the note() option and remove all of its borders for a neater display. In each cell, we add a plot and then append center-aligned text to it.

```

. putdocx begin
. putdocx table tbl6 = (2,2), border(all,nil)
> note(Figure 1. Predictive margins of age group) halign(center)
. putdocx table tbl6(1,1)=image(bmi10.png)
. putdocx table tbl6(1,1)=("a) BMI = 10"), append halign(center)
. putdocx table tbl6(1,2)=image(bmi20.png)
. putdocx table tbl6(1,2)=("b) BMI = 20"), append halign(center)
. putdocx table tbl6(2,1)=image(bmi30.png)
. putdocx table tbl6(2,1)=("c) BMI = 30"), append halign(center)
. putdocx table tbl6(2,2)=image(bmi40.png)
. putdocx table tbl6(2,2)=("d) BMI = 40"), append halign(center)

```

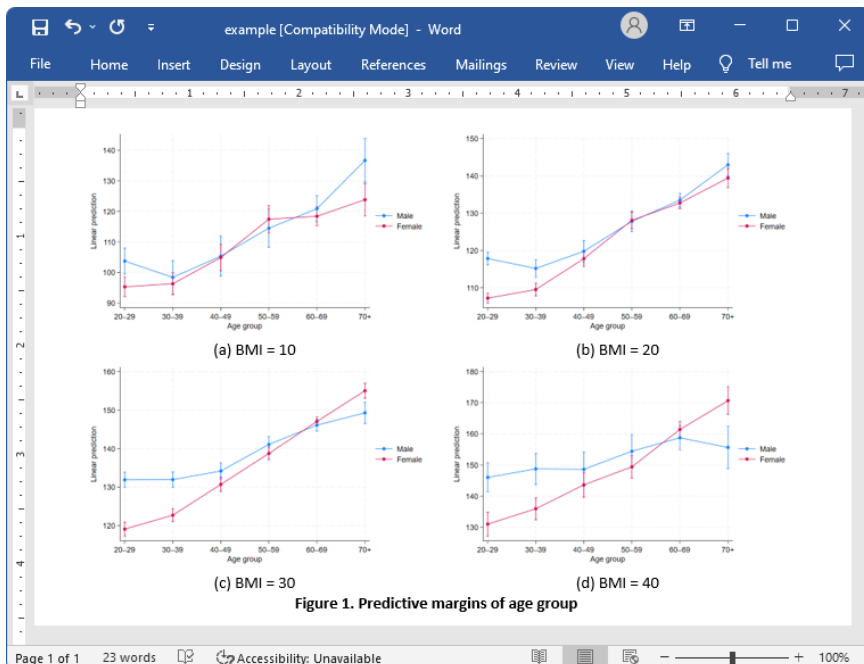
We formatted our table and the cell contents as we added content to the document. However, we would also like to format the caption. The `note()` option adds an additional row at the end of the table that spans all the columns of the table. We can format the text of the note by specifying the last row (in this case, 3) and either `.` or `1` as the column index. Here, we center-align and bold the text of the caption. Because note text is always placed within a single merged cell, it does not matter how short or long the text is when you identify the cell location.

```

. putdocx table tbl6(3,.), halign(center) bold
. putdocx save example, replace
successfully replaced "C:/mypath/example.docx"

```

After saving our work to `example.docx`, the table is displayed as follows:



□ Technical note

When you add a .svg file to a .docx file using putdocx, an extra .png file is generated using the original .svg file and embedded in the .docx file. This .png file is used to render the original .svg file in viewers that do not support rendering .svg files.



Customizing headers and footers with tables

Tables are a great way to organize summary statistics and estimation results, but they can also be used to customize the content of a header or footer. For example, you can create a table with a title or logo in one column and page numbers in another. Simply use the `header()` or `footer()` options to place the table content in either the header or the footer of the document.

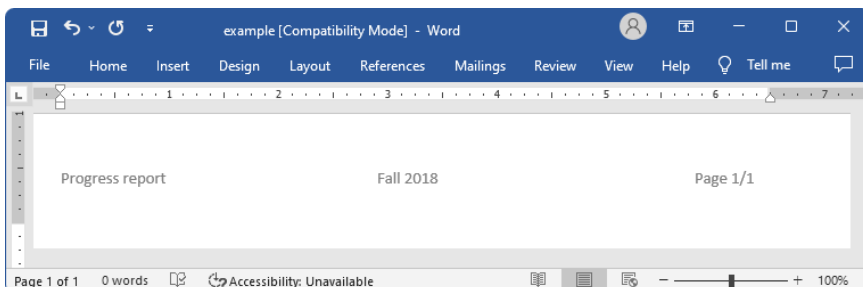
▷ Example 8: Customize header content with a table

Suppose that we are creating a progress report for the nearby community college that tracks the number of enrollees and their grade point averages. To better organize ourselves, we want to include the semester in the header of our report. Additionally, we want to include the page number and the total page count. We can include all this content in the header of our report by placing it in a table.

We add a blank header to our document by specifying the `header` option with `putdocx begin`. Then we can add the table contents to the header by specifying the `toheader()` option. We place our report title in the first column, the semester in the center column, and the page numbers in the last column. We first list the current `pagenumber`, and we then append a forward slash followed by the total page count.

```
. putdocx begin, header(fall_report)
. putdocx table hdr = (1, 3), border(all, nil) toheader(fall_report)
. putdocx table hdr(1, 1) = ("Progress report")
. putdocx table hdr(1, 2) = ("Fall 2018"), halign(center)
. putdocx table hdr(1, 3) = ("Page "), pagenumber
. putdocx table hdr(1, 3) = ("/"), totalpages append
. putdocx table hdr(1, 3), halign(right)
. putdocx save example, replace
successfully replaced "C:/mypath/example.docx"
```

After saving the document, it contains the following:



Stored results

putdocx describe *tablename* stores the following in `r()`:

Scalars

<code>r(nrows)</code>	number of rows in the table
<code>r(ncols)</code>	number of columns in the table

Reference

McDowell, A., A. Engel, J. T. Massey, and K. Maurer. 1981. "Plan and operation of the Second National Health and Nutrition Examination Survey, 1976–1980". In *Vital and Health Statistics*, ser. 1, no. 15. Hyattsville, MD: National Center for Health Statistics.

Also see

[RPT] [putdocx intro](#) — Introduction to generating Office Open XML (.docx) files

[RPT] [putdocx begin](#) — Create an Office Open XML (.docx) file

[RPT] [putdocx collect](#) — Add a table from a collection to an Office Open XML (.docx) file

[RPT] [putdocx pagebreak](#) — Add breaks to an Office Open XML (.docx) file

[RPT] [putdocx paragraph](#) — Add text or images to an Office Open XML (.docx) file

[RPT] [Appendix for putdocx](#) — Appendix for putdocx entries

Description

This is the collective appendix for the putdocx entries.

Border patterns

<i>bpattern</i>	
<code>nil</code>	<code>thickThinMediumGap</code>
<code>single</code>	<code>thinThickThinMediumGap</code>
<code>thick</code>	<code>thinThickLargeGap</code>
<code>double</code>	<code>thickThinLargeGap</code>
<code>dotted</code>	<code>thinThickThinLargeGap</code>
<code>dashed</code>	<code>wave</code>
<code>dotDash</code>	<code>doubleWave</code>
<code>dotDotDash</code>	<code>dashSmallGap</code>
<code>triple</code>	<code>dashDotStroked</code>
<code>thinThickSmallGap</code>	<code>threeDEmboss</code>
<code>thickThinSmallGap</code>	<code>threeDEngrave</code>
<code>thinThickThinSmallGap</code>	<code>outset</code>
<code>thinThickMediumGap</code>	<code>inset</code>

Chapter styles

<i>chapStyle</i>	
<code>Heading1</code>	<code>Heading6</code>
<code>Heading2</code>	<code>Heading7</code>
<code>Heading3</code>	<code>Heading8</code>
<code>Heading4</code>	<code>Heading9</code>
<code>Heading5</code>	

Colors

color, bgcolor, fgcolor, bgcolor

aliceblue	darkslategray	lightsalmon	palevioletred
antiquewhite	darkturquoise	lightseagreen	papayawhip
aqua	darkviolet	lightskyblue	peachpuff
aquamarine	deeppink	lightslategray	peru
azure	deepskyblue	lightsteelblue	pink
beige	dimgray	lightyellow	plum
bisque	dodgerblue	lime	powderblue
black	firebrick	limegreen	purple
blanchedalmond	floralwhite	linen	red
blue	forestgreen	magenta	rosybrown
blueviolet	fuchsia	maroon	royalblue
brown	gainsboro	mediumaquamarine	saddlebrown
burlywood	ghostwhite	mediumblue	salmon
cadetblue	gold	mediumorchid	sandybrown
chartreuse	goldenrod	mediumpurple	seagreen
chocolate	gray	mediumseagreen	seashell
coral	green	mediumslateblue	sienna
cornflowerblue	greenyellow	mediumspringgreen	silver
cornsilk	honeydew	mediumturquoise	skyblue
crimson	hotpink	mediumvioletred	slateblue
cyan	indianred	midnightblue	slategray
darkblue	indigo	mintcream	snow
darkcyan	ivory	mistyrose	springgreen
darkgoldenrod	khaki	moccasin	steelblue
darkgray	lavender	navajowhite	tan
darkgreen	lavenderblush	navy	teal
darkkhaki	lawngreen	oldlace	thistle
darkmagenta	lemonchiffon	olive	tomato
darkolivegreen	lightblue	olivedrab	turquoise
darkorange	lightcoral	orange	violet
darkorchid	lightcyan	orangered	wheat
darkred	lightgoldenrodyellow	orchid	white
darksalmon	lightgray	palegoldenrod	whitesmoke
darkseagreen	lightgreen	palegreen	yellow
darkslateblue	lightpink	paleturquoise	yellowgreen

Page number formats

pnformat

cardinal_text	lower_letter
decimal	lower_roman
decimal_enclosed_circle	none
decimal_enclosed_fullstop	ordinal_text
decimal_enclosed_paren	upper_letter
decimal_zero	upper_roman

Paragraph styles

pstyle

Title	Heading5
Subtitle	Heading6
Heading1	Heading7
Heading2	Heading8
Heading3	Heading9
Heading4	

Shading patterns

fpattern

nil	pct20
clear	pct25
solid	pct30
horzStripe	pct35
vertStripe	pct37
reverseDiagStripe	pct40
diagStripe	pct45
horzCross	pct50
diagCross	pct55
thinHorzStripe	pct60
thinVertStripe	pct62
thinReverseDiagStripe	pct65
thinDiagStripe	pct70
thinHorzCross	pct75
thinDiagCross	pct80
pct5	pct85
pct10	pct87
pct12	pct90
pct15	pct95

Underline patterns

upattern

none	dashLong
single	dashLongHeavy
words	dotDash
double	dashDotHeavy
thick	dotDotDash
dotted	dashDotDotHeavy
dottedHeavy	wave
dash	wavyHeavy
dashedHeavy	wavyDouble

Unsupported estimation commands

Command name	Available stored results documented in
bayesmh	[BAYES] bayesmh
bayes	[BAYES] bayes
boxcox	[R] boxcox
exlogistic	[R] exlogistic
expoisson	[R] expoisson
menl	[ME] menl
rocfit	[R] rocfit
varbasic	[TS] varbasic
xthtaylor	[XT] xthtaylor
xtpcse	[XT] xtpcse

Also see

[RPT] **putdocx intro** — Introduction to generating Office Open XML (.docx) files

[RPT] **putdocx begin** — Create an Office Open XML (.docx) file

[RPT] **putdocx pagebreak** — Add breaks to an Office Open XML (.docx) file

[RPT] **putdocx paragraph** — Add text or images to an Office Open XML (.docx) file

[RPT] **putdocx table** — Add tables to an Office Open XML (.docx) file

Description

putexcel writes Stata [expressions](#), [matrices](#), tables, images, and [returned results](#) to an Excel file. putexcel can also format the cells in the worksheet. This allows you to automate exporting and formatting of, for example, Stata estimation results. Excel 1997/2003 (.xls) files and Excel 2007/2010 and newer (.xlsx) files are supported.

putexcel set sets the Excel file to create, modify, or replace in subsequent putexcel commands. You must set the destination file before using any other putexcel commands. putexcel save closes the open file in memory and saves it to disk. putexcel clear clears the file information set by putexcel set. putexcel describe displays the file information set by putexcel set.

For an advanced syntax to simultaneously write multiple output types, see [\[RPT\] putexcel advanced](#).

Quick start

Declare the first sheet of myresults.xlsx to be the destination workbook for subsequent putexcel commands

```
putexcel set myresults
```

Same as above, but use a new sheet named Estimation Results and replace the existing workbook

```
putexcel set myresults, sheet("Estimation Results", replace)
```

Write the text “Coefficients” to cell B1

```
putexcel B1 = "Coefficients"
```

Add a table from the current collection to a new sheet named table2 beginning in cell B2

```
putexcel B2 = collect, sheet(table2)
```

Add variable names and estimated coefficients in the column under “Coefficients” after [regress](#), and format coefficients with two decimal places

```
matrix b = e(b)'  
putexcel A2 = matrix(b), rownames nformat(number_d2)
```

Format the header row of the table with a bottom border and bold text

```
putexcel (A1:B1), bold border(bottom)
```

Add a PNG of a [margins](#) plot saved to disk as mymargins.png where the upper-left corner is aligned with the upper-left corner of cell D2

```
marginsplot, name(mymargins)  
graph export mymargins.png, name(mymargins)  
putexcel D2 = image(mymargins.png)
```

Menu

File > Export > Results to Excel spreadsheet (*.xls;*.xlsx)

Syntax

Set workbook for export

```
putexcel set filename [ , set_options ]
```

Change worksheet settings

```
putexcel sheetset [ , sheetset_options ]
```

Write expression to Excel

```
putexcel ul_cell = exp [ , expression_options format_options ]
```

Export Stata matrix to Excel

```
putexcel ul_cell = matrix(matname) [ , matrix_options format_options ]
```

Export Stata graph, path diagram, or other image to Excel

```
putexcel ul_cell = image(filename)
```

Export returned results to Excel

```
putexcel ul_cell = returnset [ , colwise overwritefmt ]
```

Write formula to Excel

```
putexcel ul_cell = formula(formula) [ , overwritefmt ]
```

Write hyperlink to Excel

```
putexcel ul_cell = hyperlink(link, link_name)
```

Format cells

```
putexcel cellrange, overwritefmt format_options
```

Add the coefficient table from the last estimation command to Excel file

```
putexcel ul_cell = etable [ (#1 #2 ... #n) ]
```

Add a table from the current collection to Excel file

```
putexcel ul_cell = collect [ , collect_options ]
```


Close and save Excel file

putexcel save

Describe current export settings

putexcel describe

Clear current export settings

putexcel clear

ul_cell is a valid Excel upper-left cell specified using standard Excel notation, for example, A1 or D4.
cellrange is *ul_cell* or *ul_cell*:*lr_cell*, where *lr_cell* is a valid Excel lower-right cell, for example, A1, A1:D1, A1:A4, or A1:D4.

<i>set_options</i>	Description
open	open Excel file in memory
modify	modify Excel file
replace	overwrite Excel file
<u>sheet</u> (<i>sheetname</i> [, <i>replace</i>])	specify the worksheet to use; the default sheet name is Sheet1

<i>sheetset_options</i>	Description
<u>split</u> ([#] [, #])	split a worksheet at row # or column # or both
<u>header</u> ("text" [, <i>margin</i> (#)])	insert a header into a worksheet
<u>footer</u> ("text" [, <i>margin</i> (#)])	insert a footer into a worksheet
<u>gridon</u>	display the cell grid in Excel
<u>gridoff</u>	hide the cell grid in Excel
<u>hpagebreak</u> (#)	insert a page break at row #
<u>vpagebreak</u> (#)	insert a page break at column #
<u>namedrange</u> ("text" , <i>cellrange</i>)	add a named cell range to a worksheet

<i>expression_options</i>	Description
Main	
<u>overwritefmt</u>	overwrite existing cell formatting when exporting new content
<u>asdate</u>	convert Stata date (%td-formatted) <i>exp</i> to an Excel date
<u>asdatetime</u>	convert Stata datetime (%tc-formatted) <i>exp</i> to an Excel datetime
<u>asdatenum</u>	convert Stata date <i>exp</i> to an Excel date number, preserving the cell's format
<u>asdatetimenum</u>	convert Stata datetime <i>exp</i> to an Excel datetime number, preserving the cell's format

<i>matrix_options</i>	Description
Main	
<code>overwritefmt</code>	overwrite existing cell formatting when exporting new content
<code>names</code>	also write row names and column names for matrix <i>matname</i> ; may not be combined with <code>rownames</code> or <code>colnames</code>
<code>rownames</code>	also write matrix row names for matrix <i>matname</i> ; may not be combined with <code>names</code> or <code>colnames</code>
<code>colnames</code>	also write matrix column names for matrix <i>matname</i> ; may not be combined with <code>names</code> or <code>rownames</code>
<i>format_options</i>	Description
Number	
<code>nformat(excelnfmt)</code>	specify format for numbers
Alignment	
<code>left</code>	left-align text
<code>hcenter</code>	center text horizontally
<code>right</code>	right-align text
<code>top</code>	vertically align text with the top
<code>vcenter</code>	center text vertically
<code>bottom</code>	vertically align text with the bottom
<code>txtindent(#)</code>	indent text by # spaces; default is 0
<code>txtrotate(#)</code>	rotate text by # degrees; default is 0
<code>[no]txtwrap</code>	wrap text within each cell
<code>[no]shrinkfit</code>	shrink text to fit the cell width
<code>merge</code>	merge cells in <i>cellrange</i>
<code>unmerge</code>	separate merged cells identified by <i>ul_cell</i>
Font	
<code>font([fontname] [, size [, color]])</code>	specify font, font size, and font color
<code>[no]italic</code>	format text as italic
<code>[no]bold</code>	format text as bold
<code>[no]underline</code>	underline text in the specified cells
<code>[no]strikeout</code>	strikeout text in the specified cells
<code>script(sub super none)</code>	specify subscript or superscript formatting
Border	
<code>border(border [, style [, color]])</code>	specify horizontal and vertical cell border style
<code>dborder(direction [, style [, color]])</code>	specify diagonal cell border style
Fill	
<code>fpattern(pattern [, fgcolor [, bgcolor]])</code>	specify fill pattern for cells

<i>collect_options</i>	Description
<code>name(<i>cname</i>)</code>	use collection <i>cname</i>
<code>noisily</code>	show putexcel commands used for exporting
<code>dofile(<i>filename</i>[, <i>replace</i>])</code>	save putexcel commands used for exporting to the specified do-file

putexcel *ul_cell* = collect and *collect_options* do not appear in the dialog box.

Output types

exp writes a valid Stata expression to a cell; see [U] 13 Functions and expressions. Stata dates and datetimes differ from Excel dates and datetimes. To properly export date and datetime values, use `asdate` and `asdatetime`.

matrix(*matname*) writes the values from a Stata matrix to Excel. Stata determines where to place the data in Excel by default from the size of the matrix (the number of rows and columns) and the location you specified in *ul_cell*. By default, *ul_cell* contains the first element of *matname*, and matrix row names and column names are not written.

image(*filename*) writes a portable network graphics (.png), JPEG (.jpg), Windows metafile (.wmf), device-independent bitmap (.dib), enhanced metafile (.emf), or bitmap (.bmp) file to an Excel worksheet. The upper-left corner of the image is aligned with the upper-left corner of the specified *ul_cell*. The image is not resized. If *filename* contains spaces, it must be enclosed in double quotes.

returnset is a shortcut name that is used to identify a group of *return* values. It is intended primarily for use by programmers and by those who intend to do further processing of their exported results in Excel. *returnset* may be any one of the following:

<i>returnset</i>	
<code>escalars</code>	<code>escalarmnames</code>
<code>rscalars</code>	<code>rscalarnames</code>
<code>emacros</code>	<code>emacronames</code>
<code>rmacros</code>	<code>rmacronames</code>
<code>ematrices</code>	<code>ematrixnames</code>
<code>rmatrices</code>	<code>rmatrixnames</code>
<code>e*</code>	<code>enames</code>
<code>r*</code>	<code>rnames</code>

formula(*formula*) writes an Excel formula to the cell specified in *ul_cell*. *formula* may be any valid Excel formula. Stata does not validate formulas; the text is passed literally to Excel.

hyperlink(*link*, *link_name*) writes a hyperlink to the cell specified in *ul_cell*. *link* may be an external file, a cell, or a webpage. Stata does not validate the links; the text is passed literally to Excel. Examples:

```
putexcel A1 = hyperlink("[auto.xlsx]Sheet1!C2", "Result")
putexcel A1 = hyperlink("https://www.stata.com", "StataCorp")
putexcel A1 = hyperlink(".\auto.xlsx", "1978 automobile data")
```

`etable[(#1 #2 ... #n)]` adds an automatically generated table to an Excel file starting in *ul_cell*. The table may be derived from the coefficient table of the last estimation command, from the table of margins after the last `margins` command, or from the table of results from one or more models displayed by `estimates table`.

If the estimation command outputs $n > 1$ coefficient tables, the default is to add all tables and assign the corresponding table names *tablename1*, *tablename2*, ..., *tablename_n*. To specify which tables to add, supply the optional numlist to `etable`. For example, to add only the first and third tables from the estimation output, specify `etable(1 3)`. A few estimation commands do not support the `etable` output type. See [Unsupported estimation commands](#) in [RPT] [Appendix for putdocx](#) for a list of estimation commands that are not supported by `putexcel`.

`collect` adds a table from the current collection to an Excel file starting in *ul_cell*. This table may be created using `collect` or `table`. See [TABLES] [Intro](#) for more information on using `collect` to create a customized table from a collection of results from one or more Stata commands. See [R] [table intro](#) for information on using `table` to create tabulations, tables of summary statistics, tables of regression results, and more.

Options

Set

`open` permits `putexcel set` to open the Excel file in memory for modification. The Excel file is written to disk when `putexcel save` is issued.

`modify` permits `putexcel set` to modify an Excel file.

`replace` permits `putexcel set` to overwrite an existing Excel workbook. The workbook is overwritten when the first `putexcel` command is issued unless the `open` option is used.

`sheet(sheetname [, replace])` saves to the worksheet named *sheetname*. If there is no worksheet named *sheetname* in the workbook, then a new sheet named *sheetname* is created. If this option is not specified, `Sheet1` is used.

`replace` permits `putexcel set` to overwrite *sheetname* if it exists in the specified *filename*.

Sheet settings

`split([#] [, #])` splits a worksheet at row # or column # or both.

`header(" [text] " [, margin(#)])` inserts a header into a worksheet.

`footer(" [text] " [, margin(#)])` inserts a footer into a worksheet.

`gridon` shows the cell grid for a worksheet.

`gridoff` hides the cell grid for a worksheet.

`hpagebreak(#)` inserts a page break at row # for a worksheet.

`vpagebreak(#)` inserts a page break at column # for a worksheet.

`namedrange("text" , cellrange)` adds a named cell range to a worksheet.

`overwritefmt` causes `putexcel` to remove any existing cell formatting in the cell or cells to which it is writing new output. By default, all existing cell formatting is preserved. `overwritefmt`, when combined with a cell range, writes the cell format more efficiently.

`asdate` tells `putexcel` that the specified *exp* is a Stata `%td`-formatted date that should be converted to an Excel date with *m/d/yyyy* Excel date format.

This option has no effect if an *exp* is not specified as the output type.

`asdatetime` tells `putexcel` that the specified *exp* is a Stata `%tc`-formatted datetime that should be converted to an Excel datetime with *m/d/yyyy h:mm* Excel datetime format.

This option has no effect if an *exp* is not specified as the output type.

`asdatetimeum` tells `putexcel` that the specified *exp* is a Stata `%td`-formatted date that should be converted to an Excel date number, preserving the cell's format.

This option has no effect if an *exp* is not specified as the output type.

`asdatetimeum` tells `putexcel` that the specified *exp* is a Stata `%tc`-formatted datetime that should be converted to an Excel datetime number, preserving the cell's format.

This option has no effect if an *exp* is not specified as the output type.

`names` specifies that matrix row names and column names be written into the Excel worksheet along with the matrix values. If you specify `names`, then *ul_cell* will be blank, the cell to the right of it will contain the name of the first column, and the cell below it will contain the name of the first row. `names` may not be specified with `rownames` or `colnames`.

This option has no effect if `matrix()` is not specified as the output type.

`rownames` specifies that matrix row names be written into the Excel worksheet along with the matrix values. If you specify `rownames`, then *ul_cell* will contain the name of the first row. `rownames` may not be specified with `names` or `colnames`.

This option has no effect if `matrix()` is not specified as the output type.

`colnames` specifies that matrix column names be written into the Excel worksheet along with the matrix values. If you specify `colnames`, then *ul_cell* will contain the name of the first column. `colnames` may not be specified with `names` or `rownames`.

This option has no effect if `matrix()` is not specified as the output type.

`colwise` specifies that if a *returnset* is used, the values written to the Excel worksheet be written in consecutive columns. By default, the values are written in consecutive rows.

This option has no effect if a *returnset* is not specified as the output type.

Number

`nformat(excelfmt)` changes the numeric format of a cell range. Codes for commonly used formats are shown in the table of numeric formats in the [Appendix](#). However, any valid Excel format is permitted. Formats are formed from combinations of the following symbols.

Symbol	Description	Cell value	Fmt code	Cell displays
0	Digit placeholder (add zeros)	8.9	#.00	8.90
#	Digit placeholder (no zeros)	8.9	#.##	8.9
?	Digit placeholder (add space)	8.9	0.0?	8.9
.	Decimal point			
%	Percentage	.1	%	10%
,	Thousands separator	10000	#,###	10,000
E- E+ e- e+	Scientific format	12200000	0.00E+00	1.22E+07
\$-+/() :space	Display the symbol	12	(000)	(012)
\	Escape character	3	0\!	3!
*	Repeat character (fill in cell width)	3	3*	3xxxxx
_	Skip width of next character	-1.2	_0.0	1.2
"text"	Display text in quotes	1.23	0.00 "a"	1.23 a
@	Text placeholder	b	"a"@ "c"	abc

Formats that contain spaces must be enclosed in double quotes.

Alignment

`left` sets the specified cells to have contents left-aligned within the cell. `left` may not be combined with `right` or `hcenter`. Right-alignment is the Excel default for numeric values and need not be specified when outputting numbers.

`hcenter` sets the specified cells to have contents horizontally centered within the cell. `hcenter` may not be combined with `left` or `right`.

`right` sets the specified cells to have contents right-aligned within the cell. `right` may not be combined with `left` or `hcenter`. Left-alignment is the Excel default for text and need not be specified when outputting strings.

`top` sets the specified cells to have contents vertically aligned with the top of the cell. `top` may not be combined with `bottom` or `vcenter`.

`vcenter` sets the specified cells to have contents vertically aligned with the center of the cell. `vcenter` may not be combined with `top` or `bottom`.

`bottom` sets the specified cells to have contents vertically aligned with the bottom of the cell. `bottom` may not be combined with `top` or `vcenter`.

`txtindent(#)` sets the text indention in each cell in a cell range. `#` must be an integer between 0 and 15.

`txtrotate(#)` sets the text rotation in each cell in a cell range. `#` must be an integer between 0 and 180 or equal to 255. `txtrotate(0)` is equal to no rotation and is the default. `txtrotate(255)` specifies vertical text. Values 1–90 rotate the text counterclockwise 1 to 90 degrees. Values 91–180 rotate the text clockwise 1 to 90 degrees.

`txtwrap` and `nottxtwrap` specify whether the text is to be wrapped in a cell or within each cell in a range of cells. The default is no wrapping. `nottxtwrap` has an effect only if the cell or cells were previously formatted to wrap. `txtwrap` may not be specified with `shrinkfit`.

`shrinkfit` and `noshrinkfit` specify whether the text is to be shrunk to fit in the cell width of a cell or in each cell of a range of cells. The default is no shrinking. `noshrinkfit` has an effect only if the cell or cells were previously formatted to shrink text to fit. `shrinkfit` may not be specified with `txtwrap`.

`merge` tells Excel to merge cells in the specified cell range. `merge` may be combined with `left`, `right`, `hcenter`, `top`, `bottom`, and `vcenter` to format the merged cell. Merging cells that contain data in each cell will result in the upper-leftmost data being kept.

Once you have merged cells, you can refer to the merged cell by using any single cell from the specified *cellrange*. For example, if you specified a *cellrange* of A1:B2, you could refer to the merged cell using A1, B1, A2, or B2.

`unmerge` tells Excel to unmerge previously merged cells. When using `unmerge`, you only need to use a single cell from the merged cell in the previously specified *cellrange*.

Font

`font([fontname] [, size [, color]])` sets the font, font size, and font color for each cell in a cell range. The font size and font color may be specified individually without specifying *fontname*. Use `font("", size)` to specify font size only. Use `font("", "", color)` to specify font color only. For both cases, the default font will be used. If `font()` is not specified, the Excel defaults are preserved.

fontname may be any valid Excel font. If *fontname* includes spaces, then it must be enclosed in double quotes. What constitutes a valid Excel font is determined by the version of Excel that is installed on the user's computer.

size is a numeric value that represents any valid Excel font size. The default is 12.

color may be one of the colors listed in the table of colors in the [Appendix](#) or may be a valid RGB value in the form "### #### ####". If no *color* is specified, then Excel workbook defaults are used.

`italic` and `noitalic` specify whether to italicize or unitalicize the text in a cell or range of cells. The default is for text to be unitalicized. `noitalic` has an effect only if the cell or cells were previously italicized.

`bold` and `nobold` specify whether to bold or unbold the text in a cell or range of cells. The default is for text to be unbold. `nobold` has an effect only if the cell or cells were previously formatted as bold.

`underline` and `nounderline` specify whether to underline the text or remove the underline from the text in a cell or range of cells. The default is for text not to be underlined. `nounderline` has an effect only if the cell or cells previously contained underlined text.

`strikeout` and `nostrikeout` specify whether to strikeout the text or remove the strikeout from the text in a cell or range of cells. The default is for text not to have a strikeout mark. `nostrikeout` has an effect only if the cell or cells previously had a strikeout mark.

`script(sub | super | none)` changes the script style of the cell. `script(sub)` makes all text in a cell or range of cells a subscript. `script(super)` makes all text in a cell or range of cells a superscript. `script(none)` removes all subscript or superscript formatting from a cell or range of cells. Specifying `script(none)` has an effect only if the cell or cells were previously formatted as subscript or superscript.

Border

`border(border [, style [, color]])` sets the cell border, style, and color for a cell or range of cells.

border may be `all`, `left`, `right`, `top`, or `bottom`.

style is a keyword specifying the look of the border. The most common styles are `thin`, `medium`, `thick`, and `double`. The default is `thin`. For a complete list of border styles, see the [Appendix](#). To remove an existing border, specify `none` as the *style*.

color may be one of the colors listed in the table of colors in the [Appendix](#) or may be a valid RGB value in the form "`### #`". If no *color* is specified, then Excel workbook defaults are used.

`dborder(direction [, style [, color]])` sets the cell diagonal border direction, style, and color for a cell or range of cells.

direction may be `down`, `up`, or `both`. `down` draws a line from the upper-left corner of the cell to the lower-right corner of the cell or, for a range of cells, from the upper-left corner of *ul_cell* to the lower-right corner of *lr_cell*. `up` draws a line from the lower-left corner of the cell to the upper-right corner of the cell or, for a range of cells, from the lower-left corner of the area defined by *ul_cell*:*lr_cell* to the upper-right corner.

style is a keyword specifying the look of the border. The most common styles are `thin`, `medium`, `thick`, and `double`. The default is `thin`. For a complete list of border styles, see the [Appendix](#). To remove an existing border, specify `none` as the *style*.

color may be one of the colors listed in the table of colors in the [Appendix](#) or may be a valid RGB value in the form "`### #`". If no *color* is specified, then Excel workbook defaults are used.

Fill

`fpattern(pattern [, fgcolor [, bgcolor]])` sets the fill pattern, foreground color, and background color for a cell or range of cells.

pattern is a keyword specifying the fill pattern. The most common fill patterns are `solid` for a solid color (determined by *fgcolor*), `gray25` for 25% gray scale, `gray50` for 50% gray scale, and `gray75` for 75% gray scale. A complete list of fill patterns is shown in the [Appendix](#). To remove an existing fill pattern from the cell or cells, specify `none` as the *pattern*.

fgcolor specifies the foreground color. The default foreground color is `black`. *fgcolor* may be any of the colors listed in the table of colors in the [Appendix](#) or may be a valid RGB value in the form "`### #`".

bgcolor specifies the background color. *bgcolor* may be any of the colors listed in the table of colors in the [Appendix](#) or may be a valid RGB value in the form "`### #`". If no *bgcolor* is specified, then Excel workbook defaults are used.

The following options are available when exporting a collection:

`name(cname)` specifies a collection *cname* from which to export the table. By default, the table is taken from the current collection.

`noisily` specifies that the `putexcel` commands used to export to the workbook be displayed.

`dofile(filename[, replace])` specifies that `putexcel` save to *filename* the commands used to export to the workbook. If *filename* already exists, it can be overwritten by specifying `replace`. If *filename* is specified without an extension, `.do` is assumed.

Remarks and examples

Remarks are presented under the following headings:

Introduction
Writing expressions and formatting cells
Exporting summary statistics to Excel
Exporting estimation results
Exporting a table from a collection
Exporting graphs and other images

Introduction

The `putexcel` command is a means of directly controlling the layout of an Excel file. As such, `putexcel` is designed to mimic the options and functionality of Excel, and many options of `putexcel` are simply pass-through arguments to Excel itself. In what follows, we provide documentation of how to use the `putexcel` command. However, for many options, such as the specification of valid numeric formats, font names, and font sizes, users are encouraged to also consult the help for their specific version of Excel.

`putexcel` may be used with Excel 1997/2003 (`.xls`) and Excel 2007/2010 and newer (`.xlsx`). `putexcel` looks at the file extension `.xls` or `.xlsx` to determine which Excel format to write. It is supported on Windows, Mac, and Linux.

`putexcel` also has an advanced syntax that allows you to write multiple types of output to different cells or cell ranges at a time; see [RPT] [putexcel advanced](#).

▷ Example 1: Setting the workbook and sheet

Before we can write to an Excel workbook using `putexcel`, we need to tell Stata what the destination is. We do this using the `putexcel set` command. For the next several examples, we will use an Excel file named `myresults.xlsx` and a sheet named `Descriptive`.

```
. putexcel set myresults.xlsx, sheet(Descriptive)
```

If we had not specified the sheet name, `putexcel set` would have defaulted to using the first sheet in the workbook.



Writing expressions and formatting cells

Although there are many uses for `putexcel`, one of the primary reasons to use the command is to keep track of the results of analyses in a single location for later use. The next several examples show how to export results to a single location for easy sharing and to create formatted tables that can be placed in a paper or poster.

Suppose we are analyzing data about the number of times young adults visited a news website. These data are contained in `website.dta` and described in detail in [example 1](#) of [R] [ivpoisson](#).

```
. use https://www.stata-press.com/data/r19/website
(Visits to website)

. describe

Contains data from https://www.stata-press.com/data/r19/website.dta
Observations:      500      Visits to website
Variables:         6      11 Feb 2024 14:45
```

Variable name	Storage type	Display format	Value label	Variable label
visits	byte	%8.0g		Visits to website
female	byte	%8.0g		Female
ad	byte	%8.0g		Advertisements
time	double	%10.0g		Time on internet (hrs.)
phone	double	%10.0g		Time on phone (hrs.)
frfam	double	%10.0g		Time with friends and out of town family (hrs.)

Sorted by:

We want to create a formatted table of summary statistics for men and women.

► Example 2: Write expression to cell

We start by writing column headers to the first row of our worksheet.

```
. putexcel A1 = "Variable"
file myresults.xlsx saved

. putexcel B1 = "Men"
file myresults.xlsx saved

. putexcel C1 = "Women"
file myresults.xlsx saved
```

Each of these commands opens the Excel workbook, writes the text to the specified cell, and then closes the workbook. We also use expressions to write out specific results (see [example 4](#)).



► Example 3: Format a range of cells

We may also want to make the column headings bold and add a border underneath. We can format a range of cells by specifying a cell range and the appropriate formatting options.

```
. putexcel A1:C1, bold border(bottom)
file myresults.xlsx saved
```

We also could have specified the bold and border() options each time we exported the column heading in [example 2](#).

```
. putexcel A1 = "Variable", bold border(bottom)
file myresults.xlsx saved

. putexcel B1 = "Men", bold border(bottom)
file myresults.xlsx saved

. putexcel C1 = "Women", bold border(bottom)
file myresults.xlsx saved
```

Whether you apply formatting at the time you write to Excel or for a range will likely depend on the number of options you have and the number of cells affected by common formatting options.



□ Technical note

When formatting many cells within a workbook, make sure to write cell formats efficiently using cell ranges. If you do not, you can overload the Excel workbook with too many formats, causing the Excel workbook to become large and, in extreme cases, the cell formatting to stop working. Take the following example:

```
putexcel set test.xlsx
putexcel A1 = "Sex"
putexcel A1, bold
putexcel A1, italic
putexcel A1, font("Arial", 16, "black")

putexcel B1 = "Age"
putexcel B1, bold
putexcel B1, italic
putexcel B1, font("Arial", 16, "black")

putexcel C1 = "Race"
putexcel C1, bold
putexcel C1, italic
putexcel C1, font("Arial", 16, "black")
```

Creating the Excel workbook with the commands above will create nine format entries for the worksheet. A more efficient way to write the cell formats is

```
putexcel set test.xlsx
putexcel A1 = "Sex"
putexcel B1 = "Age"
putexcel C1 = "Race"

putexcel A1:C1, bold italic font("Arial", 16, "black")
```

Now the Excel workbook contains only one format entry for the worksheet. Although the nine format entries created by the previous set of `putexcel` commands are unlikely to cause a problem, your Excel workbook will become large if you format hundreds of cells across multiple worksheets in this manner. Regardless of the number of cells you are formatting, you should always try to write cell formats efficiently to reduce your Excel workbook size.



Exporting summary statistics to Excel

We can use `putexcel` to write summary statistics to an Excel worksheet. Summary statistics can be obtained in two ways. The `table` command computes summary statistics, creates a table of the results, and creates a collection that can be inserted into an Excel file. Alternatively, summary statistics can be computed using another command such as `summarize` or `tabstat`. After running these commands, the available summary statistics are determined by [returned results](#). The returned results are documented in the *Stored results* section of the command's manual entry or help. You can also see what is returned by typing `return list` or `ereturn list`.

▷ Example 4: Export selected statistics

To export a specific statistic from returned results, we use the expression output type. For example, we might want a table that has the number of observations for men and women followed by means for each variable, which we show in [example 5](#).

We can obtain counts in many ways in Stata. Here we use the `summarize` command and restrict the command to males (`female==0`). We use `visits`, but any continuous variable without missing values would have worked just as well. `r(N)` stores the number of observations.

```
. summarize visits if female==0
```

Variable	Obs	Mean	Std. dev.	Min	Max
visits	257	5.105058	3.893185	1	27

```
. putexcel B2 = 'r(N)'
file myresults.xlsx saved
```

A more direct way to obtain the number of observations is with the `count` command.

```
. count if female==1
243
. putexcel C2 = 'r(N)'
file myresults.xlsx saved
```

Notice that we typed `'r(N)'` instead of `"r(N)"`. The `' '` tell Stata to fill in the numeric value associated with `r(N)` instead of exporting the text, known as macro substitution; see [P] [macro](#). If we wanted to treat the contents of `r(N)`, or any other return value, like a string, we could have typed `"'r(N)'"`.



► Example 5: Export frequency tables

You can use `putexcel` in Stata to create tables in Excel by using the `matrix()` output type. Suppose we want to create a table of means for each variable for each value of `female`. We can use `tabstat` with the `save` option and then check the return values with `return list` to determine what values to output.

```
. tabstat visits ad time phone frfam, by(female) save
Summary statistics: Mean
Group variable: female (Female)
```

female	visits	ad	time	phone	frfam
0	5.105058	2.175097	3.222412	3.164086	3.65428
1	4.946502	2.098765	3.161276	3.155391	3.676708
Total	5.028	2.138	3.1927	3.15986	3.66518

```
. return list
macros:
      r(name2) : "1"
      r(name1) : "0"

matrices:
      r(Stat2) : 1 x 5
      r(Stat1) : 1 x 5
      r(StatTotal) : 1 x 5
```

First, we transpose the row vectors `r(Stat1)` and `r(Stat2)`, which contain the means, so that the values are written in a column under each heading. We want the variable names to be included, so for males, which is the first matrix we output, we include the `rownames` option. Because we want the values in B3 and the row names of the matrix in A3, we specify A3. We use `nformat(number_d2)` to format the means with two decimal places. For females, we do not need to specify the names again, so we just specify the cell where we want the data to be written.

```
. qui tabstat visits ad time phone frfam, by(female) save
. matrix male = r(Stat1)'
. matrix female = r(Stat2)'
. putexcel A3 = matrix(male), rownames nformat(number_d2)
```

```
file myresults.xlsx saved
. putexcel C3 = matrix(female), nformat(number_d2)
file myresults.xlsx saved
```

The above commands give a final table of frequencies and means that looks like this:

	A	B	C
1	Variable	Men	Women
2		257	243
3	visits	5.11	4.95
4	ad	2.18	2.10
5	time	3.22	3.16
6	phone	3.16	3.16
7	frfam	3.65	3.68

In building this table, we have demonstrated how to add text, expressions, and matrices to an Excel file individually. Alternatively, we can compute the summary statistics with a single table command and format the results using `collect`. We demonstrate this approach in [example 8](#).

◀

Exporting estimation results

A similar approach to that used in [example 5](#) can be used to export estimation results. Suppose we want to fit a linear regression model of `visits` as a function of `ad`, `female`, and `time` using `regress`, and we want to export formatted results.

```
. regress visits ad female time
```

Source	SS	df	MS	Number of obs = 500		
Model	5710.6792	3	1903.55973	F(3, 496)	=	346.75
Residual	2722.9288	496	5.4897758	Prob > F	=	0.0000
				R-squared	=	0.6771
				Adj R-squared	=	0.6752
Total	8433.608	499	16.901018	Root MSE	=	2.343

visits	Coefficient	Std. err.	t	P> t	[95% conf. interval]	
ad	.7996179	.0516591	15.48	0.000	.6981203	.9011156
female	-.0467997	.2096816	-0.22	0.823	-.4587734	.3651739
time	.82962	.0436601	19.00	0.000	.7438385	.9154014
_cons	.6924339	.2007914	3.45	0.001	.2979273	1.086941

► Example 6: Export point estimates and sample size

We start by using `putexcel set` again to create a new worksheet for our regression results. Because we are working with the same workbook that we specified previously, we specify the `modify` option.

```
. putexcel set myresults.xlsx, sheet(Estimation) modify
```

We want to give our coefficients the title “Coef.” in the table we create, so we use an expression to write this to cell B1. We create a new matrix for our coefficients as the transpose of the values of e-class matrix `e(b)`. We use column A row 2 as the starting location for the matrix row labels, and we use column B row 2 as the starting location for the coefficients; to do this, we only need to specify the upper-left cell (A2) and the `rownames` option. To make our table more readable, we format the coefficient estimates with two decimal places by using `nformat()` and add a bottom border under the estimates.

```
. putexcel B1 = "Coef."
file myresults.xlsx saved

. matrix b = e(b)'

. putexcel A2 = matrix(b), rownames nformat(number_d2)
file myresults.xlsx saved

. putexcel A5:B5, border(bottom)
file myresults.xlsx saved
```

We then add a right-aligned and italicized “N=” in column A row 6 and the sample size from e-class scalar `e(N)` in column B row 6. We use a medium border under this row instead of the default thin border to indicate that the table is complete.

```
. putexcel A6 = "N=", italic right border(bottom, medium)
file myresults.xlsx saved

. putexcel B6 = matrix(e(N)), nformat(number_sep) border(bottom, medium)
file myresults.xlsx saved
```

The above commands create a table that looks like this:

	A	B	C
1		Coef.	
2	ad	0.80	
3	female	-0.05	
4	time	0.83	
5	_cons	0.69	
6	N=	500	
7			

If you are going to export more-complex tables or many objects, you can use the advanced syntax of `putexcel`; see [\[RPT\] putexcel advanced](#). You might also create a customized table with the `collect` or `table` command, which we demonstrate in the next section.



Exporting a table from a collection

The `collect` suite of commands allows you to collect results from Stata commands and create customized tables with these results. You can arrange the results in different ways, modify the labels for the table, format numbers within the table, and make other style changes. With `putexcel`, you can easily

export these customized tables to your Excel files, as we demonstrate below. In these examples, we assume some familiarity with the `table` and `collect` commands. We recommend that you see [\[R\] table intro](#) for information on `table` and [\[TABLES\] Intro](#) for more information on `collect` to learn more about creating and customizing tables before incorporating them into your Excel file.

► Example 7: Export a customized table with multiple results

Suppose that we now want to compare the results from the linear regression we fit in the previous example with the results from a model with an additional covariate. We will collect the results from each model and then create a customized table with both sets of results; this table will replace the contents on the Estimation sheet:

```
. putexcel set myresults.xlsx, sheet(Estimation, replace) modify
```

We begin by clearing out any style specifications for the collection. When you collect results from a Stata command, the collection will have a default style. For example, a border will be added between the row headers and the results. For this table, we would like to clear all of those specifications and begin with an empty style. Then we prefix each estimation command with `collect:`, and we suppress the output with `quietly`. Once we have collected the results, we create our table by specifying the layout with `collect layout`. We set the rows to correspond to the variable names, which are contained in the `colname` dimension. The columns will be determined by the model, accessed through the `cmdset` dimension, and by the statistics, accessed through the `result` dimension. Many results are collected from each regression model, but suppose that we only want to report the coefficients (`_r_b`) and confidence intervals (`_r_ci`).

```
. collect style clear
. quietly: collect: regress visits ad female time
. quietly: collect: regress visits ad female time phone
. quietly: collect layout (colname) (cmdset#result[_r_b _r_ci])
```

If we do not make any style specifications, we will have the title for each dimension displayed, resulting in a very wide table. So we used the `quietly` prefix to suppress the preview that `collect layout` displays by default. Below, we hide the title for all dimensions in the table, format all cells with numeric content to only display three digits after the decimal, and specify that upper and lower bounds of confidence intervals be separated by a comma and enclosed in square brackets. We also request that row headers display the variable names, the values of the dimension `colname`, rather than the variable labels. Then we report the current layout:

```
. collect style header, title(hide)
. collect style cell, nformat(%7.3f)
. collect style cell result[_r_ci], cidelimiter(,) sformat("[%s]")
. collect style header colname, level(value)
. collect layout
Collection: default
  Rows: colname
  Columns: cmdset#result[_r_b _r_ci]
Table 1: 5 x 4
```

	1		2	
	Coefficient	95% CI	Coefficient	95% CI
ad	0.800	[0.698, 0.901]	0.795	[0.679, 0.911]
female	-0.047	[-0.459, 0.365]	-0.047	[-0.460, 0.365]
time	0.830	[0.744, 0.915]	0.827	[0.737, 0.917]
phone			0.010	[-0.111, 0.132]
_cons	0.692	[0.298, 1.087]	0.677	[0.246, 1.109]

We could export this table to our Excel file now, but first let's clean up the labels. The values 1 and 2 represent the order of the `collect` commands we issued, and they are levels of the dimension `cmdset`. Below, we change the 1 to `Reduced` and the 2 to `Full`. Also, notice that the values are repeated for each column that corresponds to that model, so we specify that these duplicate column headers be placed in the center of all the cells they span and that they be centered horizontally. Additionally, we change the label for the coefficients to simply `B`:

```
. collect label levels cmdset 1 "Reduced" 2 "Full", modify
. collect style column, dups(center)
. collect label levels result _r_b "B", modify
. collect layout
Collection: default
  Rows: colname
  Columns: cmdset#result[_r_b _r_ci]
Table 1: 5 x 4
```

	Reduced			Full		
	B	95% CI		B	95% CI	
ad	0.800	[0.698, 0.901]		0.795	[0.679, 0.911]	
female	-0.047	[-0.459, 0.365]		-0.047	[-0.460, 0.365]	
time	0.830	[0.744, 0.915]		0.827	[0.737, 0.917]	
phone				0.010	[-0.111, 0.132]	
_cons	0.692	[0.298, 1.087]		0.677	[0.246, 1.109]	

So that all our contents are aligned, below we also center-align our results. The last thing we will do is add some borders to the table. The dimension `border_block` divides the table into four blocks: column-header, item (which contains the results), corner (top left area), and the row-header. We add borders to the top and bottom of the `item` and `row-header` blocks.

```
. collect style cell result, halign(center)
. collect style cell border_block[item], border(top) border(bottom)
. collect style cell border_block[row-header], border(top) border(bottom)
```

Now that we are done customizing our table, we export it to our Excel file with the upper-left cell aligned with the upper-left corner of cell A1:

```
. putexcel A1 = collect
(collection default posted to putexcel)
```


Now the Estimation sheet contains the following:

	A	B	C	D	E	F
1		Reduced		Full		
2		B	95% CI	B	95% CI	
3	ad	0.800	[0.698, 0.901]	0.795	[0.679, 0.911]	
4	female	-0.047	[-0.459, 0.365]	-0.047	[-0.460, 0.365]	
5	time	0.830	[0.744, 0.915]	0.827	[0.737, 0.917]	
6	phone			0.010	[-0.111, 0.132]	
7	_cons	0.692	[0.298, 1.087]	0.677	[0.246, 1.109]	

Exporting the collection only required a single command. However, Stata did a lot of work in the background to export the collection. If you are curious, you can use the `noisily` option to see all the commands that were used to export the collection.



➤ Example 8: Export a customized table of summary statistics

In [example 5](#), we built a table of summary statistics from returned results produced by `summarize`, `count`, and `tabstat`. We could instead compute all of these summary statistics at once using `table`. An advantage of `table` is that it creates a collection with its results. If you are happy with the table, you can simply export the table to Excel using `putexcel`'s `collect` export type. However, you may prefer to further customize the table before export. To do this, you can use the `collect` commands.

To demonstrate, we re-create the table from [example 5](#) using a series of `table` and `collect` commands. To begin, we compute the frequencies and means for males and females. Here we specify that we want statistics (`result`) and variables (`var`) as our row dimensions, while we want gender (`female`) on our columns. We also format our results to include two decimal places for the means.

```
. table (result var) (female), statistic(freq)
> statistic(mean visits ad time phone frfam) nototals
> nformat(%5.2f mean)
```

	Female	
	0	1
Frequency	257	243
Mean		
Visits to website	5.11	4.95
Advertisements	2.18	2.10
Time on internet (hrs.)	3.22	3.16
Time on phone (hrs.)	3.16	3.16
Time with friends and out of town family (hrs.)	3.65	3.68

To customize this table, we first suppress the titles of our `result` dimension to remove the `Frequency` and `Mean` headers. For conciseness, we will show the variable names instead of the variable labels. In addition, we can hide the title of `Female`, and instead we can set level labels of “Men” and “Women” to appear in the column headers. Finally, we will remove all borders.

```

. collect style header, title(hide)
. collect style header var, level(value)
. collect style header result, level(hide)
. collect label levels female 0 "Men" 1 "Women"
. collect style cell, border(,pattern(nil))
. collect preview
      Men    Women
      257     243
visits 5.11    4.95
ad      2.18    2.10
time    3.22    3.16
phone   3.16    3.16
frfam   3.65    3.68

```

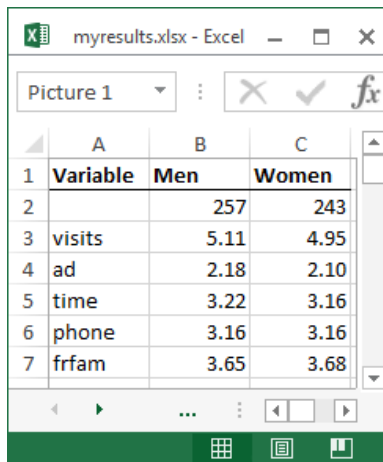
To fully re-create [example 5](#), we must add Variable to the top left cell and then specify that the names across the top are bold with a border underneath. To demonstrate another workflow, we will export our collection from table and then use putexcel's formatting options.

```

. putexcel set myresults.xlsx, sheet(Descriptive, replace) modify
. putexcel A1 = collect
(collection Table posted to putexcel)
. putexcel A1 = "Variable"
file myresults.xlsx saved
. putexcel A1:C1, bold border(bottom)
file myresults.xlsx saved

```

We have once again created



	A	B	C
1	Variable	Men	Women
2		257	243
3	visits	5.11	4.95
4	ad	2.18	2.10
5	time	3.22	3.16
6	phone	3.16	3.16
7	frfam	3.65	3.68



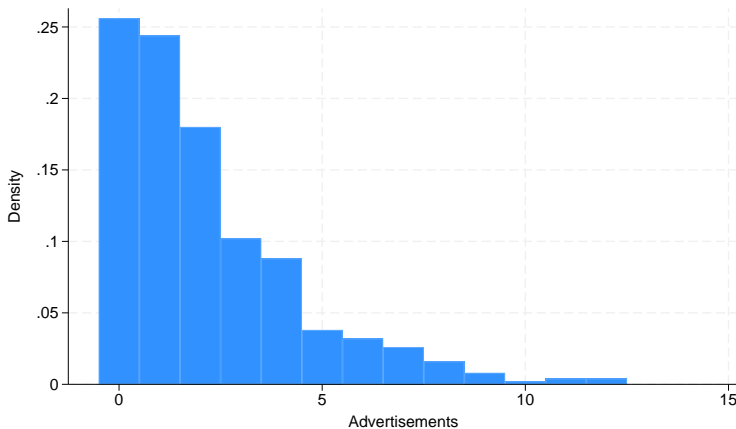
Exporting graphs and other images

You can export PNG, JPEG, and other image formats to Excel with putexcel. To export a Stata graph, you must first use graph export to convert the Stata graph to one of the supported image formats.

► Example 9: Export Stata graph

We may want to add a histogram about the number of advertisements viewed to our set of descriptive results. We can use the `histogram` command and then `graph export` to create a PNG file of our histogram.

```
. histogram ad, discrete
(start=0, width=1)
. graph export ads.png
file ads.png saved as PNG format
```



Then we output the graph to Excel with the `image()` output type.

```
. putexcel E2 = image(ads.png)
file myresults.xlsx saved
```

This adds the graph with the upper-left corner of the graph aligned in the upper-left corner of cell E2. Graphs are not resized by `putexcel`, but you can change the size with the `graph export` command; see [\[G-2\] graph export](#).



□ Technical note

See the technical notes [Excel data size limits](#) and [Dates and times](#) in [\[D\] import excel](#).



Appendix

Codes for numeric formats

Code	Example
number	1000
number_d2	1000.00
number_sep	100,000
number_sep_d2	100,000.00
number_sep_negbra	(1,000)
number_sep_negbrared	(1,000)
number_d2_sep_negbra	(1,000.00)
number_d2_sep_negbrared	(1,000.00)
currency_negbra	(\$4000)
currency_negbrared	(\$4000)
currency_d2_negbra	(\$4000.00)
currency_d2_negbrared	(\$4000.00)
account	5,000
accountcur	\$ 5,000
account_d2	5,000.00
account_d2_cur	\$ 5,000.00
percent	75%
percent_d2	75.00%
scientific_d2	10.00E+1
fraction_onedig	10 1/2
fraction_twodig	10 23/95
date	3/18/2007
date_d_mon_yy	18-Mar-07
date_d_mon	18-Mar
date_mon_yy	Mar-07
time_hmm_AM	8:30 AM
time_HMMSS_AM	8:30:00 AM
time_HMM	8:30
time_HMMSS	8:30:00
time_MMSS	30:55
time_HOMMSS	20:30:55
time_MMSS0	30:55.0
date_time	3/18/2007 8:30
text	this is text

Colors

color

aliceblue	deeppink
antiquewhite	deepskyblue
aqua	dimgray
aquamarine	dodgerblue
azure	firebrick
beige	floralwhite
bisque	forestgreen
black	fuchsia
blanchedalmond	gainsboro
blue	ghostwhite
blueviolet	gold
brown	goldenrod
burlywood	gray
cadetblue	green
chartreuse	greenyellow
chocolate	honeydew
coral	hotpink
cornflowerblue	indianred
cornsilk	indigo
crimson	ivory
cyan	khaki
darkblue	lavender
darkcyan	lavenderblush
darkgoldenrod	lawngreen
darkgray	lemonchiffon
darkgreen	lightblue
darkkhaki	lightcoral
darkmagenta	lightcyan
darkolivegreen	lightgoldenrodyellow
darkorange	lightgray
darkorchid	lightgreen
darkred	lightpink
darksalmon	lightsalmon
darkseagreen	lightseagreen
darkslateblue	lightskyblue
darkslategray	lightslategray
darkturquoise	lightsteelblue
darkviolet	lightyellow

color, continued

lime	peru
limegreen	pink
linen	plum
magenta	powerblue
maroon	purple
mediumaquamarine	red
mediumblue	rosybrown
mediumorchid	royalblue
mediumpurple	saddlebrown
mediumseagreen	salmon
mediumslateblue	sandybrown
mediumspringgreen	seagreen
mediumturquoise	seashell
mediumvioletred	sienna
midnightblue	silver
mintcream	skyblue
mistyrose	slateblue
moccasin	snow
navajowhite	springgreen
navy	steelblue
oldlace	tan
olive	teal
olivedrab	thistle
orange	tomato
orangered	turquoise
orchid	violet
palegoldenrod	wheat
palegreen	white
paleturquoise	whitesmoke
palevioletred	yellow
papayawhip	yellowgreen
peachpuff	

Border styles

style

none
thin
medium
dashed
dotted
thick
double
hair
medium_dashed
dash_dot
medium_dash_dot
dash_dot_dot
medium_dash_dot_dot
slant_dash_dot

Background patterns

pattern

none
solid
gray50
gray75
gray25
horstripe
verstripe
diagstripe
revdiagstripe
diagcrosshatch
thinhorstripe
thinverstripe
thindia stripe
thinrevdiagstripe
thinhorcrosshatch
thindiacrosshatch
thickdiacrosshatch
gray12p5
gray6p25

Stored results

putexcel collect stores the following in `s()`:

Scalars

<code>s(collection)</code>	name of collection
<code>s(dofile)</code>	name of the new do-file

References

- Crow, K. 2013. Export tables to Excel. *The Stata Blog: Not Elsewhere Classified*. <https://blog.stata.com/2013/09/25/export-tables-to-excel/>.
- . 2014. Retaining an Excel cell's format when using putexcel. *The Stata Blog: Not Elsewhere Classified*. <https://blog.stata.com/2014/02/04/retaining-an-excel-cells-format-when-using-putexcel/>.
- . 2018. Export tabulation results to Excel—update. *The Stata Blog: Not Elsewhere Classified*. <https://blog.stata.com/2018/06/07/export-tabulation-results-to-excel-update/>.
- Gallup, J. L. 2012. A new system for formatting estimation tables. *Stata Journal* 12: 3–28.
- Huber, C. 2017a. Creating Excel tables with putexcel, part 1: Introduction and formatting. *The Stata Blog: Not Elsewhere Classified*. <https://blog.stata.com/2017/01/10/creating-excel-tables-with-putexcel-part-1-introduction-and-formatting/>.
- . 2017b. Creating Excel tables with putexcel, part 2: Macro, picture, matrix, and formula expressions. *The Stata Blog: Not Elsewhere Classified*. <https://blog.stata.com/2017/01/24/creating-excel-tables-with-putexcel-part-2-macro-picture-matrix-and-formula-expressions/>.
- Quintó, L. 2012. HTML output in Stata. *Stata Journal* 12: 702–717.

Also see

- [RPT] **putexcel advanced** — Export results to an Excel file using advanced syntax
- [RPT] **putdocx intro** — Introduction to generating Office Open XML (.docx) files
- [RPT] **putpdf intro** — Introduction to generating PDF files
- [D] **import excel** — Import and export Excel files
- [M-5] **_docx*()** — Generate Office Open XML (.docx) file
- [M-5] **Pdf*()** — Create a PDF file
- [M-5] **xl()** — Excel file I/O class
- [R] **table intro** — Introduction to tables of frequencies, summaries, and command results
- [TABLES] **Intro** — Introduction

[Description
Options](#)

[Quick start
Remarks and examples](#)

[Menu
References](#)

[Syntax
Also see](#)

Description

`putexcel` with the advanced syntax may be used to simultaneously write Stata [expressions](#), [matrices](#), tables, images, and [returned results](#) to an Excel file. It may also be used to format existing contents of cells in a worksheet. This syntax is intended for use by programmers of commands that call `putexcel` in the background and by other advanced users. Excel 1997/2003 (.xls) files and Excel 2007/2010 and newer (.xlsx) files are supported.

`putexcel set` sets the Excel file to create, modify, or replace in subsequent `putexcel` commands. You must set the destination file before using any other `putexcel` commands. `putexcel save` closes the open file in memory and saves it to disk. `putexcel clear` clears the file information set by `putexcel set`. `putexcel describe` displays the file information set by `putexcel set`.

For a simplified version of the syntax, see [\[RPT\] putexcel](#).

Quick start

Declare the first sheet of `myresults.xlsx` to be the destination workbook for subsequent `putexcel` commands

```
putexcel set myresults
```

Same as above, but use a new sheet named `Estimation Results` and replace the existing workbook

```
putexcel set myresults, sheet("Estimation Results", replace)
```

Write estimation results in `e(b)` to column B, starting in row 2, with the results formatted to have two decimal places, matrix row names in column A, and the title “Coefs.” in cell B1

```
matrix b=e(b)'  
putexcel B1="Coefs." A2=matrix(b), rownames nformat(number_d2)
```

Add a thin border under cells A1 to B1 and italicize the text

```
putexcel (A1:B1), border(bottom) italic
```

Write “Some text that is too long to fit” to cell D1 and set the text to wrap within the cell

```
putexcel D1="Some text that is too long to fit", txtwrap
```

Merge D1 with D2, E1, and E2, and horizontally and vertically center

```
putexcel (D1:E2), merge hcenter vcenter
```

Menu

File > Export > Results to Excel spreadsheet (*.xls;*.xlsx)

Syntax

Set workbook for export

```
putexcel set filename [ , set_options ]
```

Specify formatting and output

```
putexcel spec1 [spec2 [...]] [ , export_options format_options ]
```

Close and save current Excel file

```
putexcel save
```

Describe current export settings

```
putexcel describe
```

Clear current export settings

```
putexcel clear
```

spec may be *ul_cell* or *cellrange* of the form *ul_cell*:*lr_cell* if no output is to be written or may be one of the following **output types**:

ul_cell = *exp*

ul_cell:*lr_cell* = *exp*

ul_cell = `matrix(matname)`

ul_cell = `image(filename)`

ul_cell = `returnset`

ul_cell = `formula(formula)`

ul_cell = `hyperlink(link, link_name)`

ul_cell = `etable[(#1 #2 ... #n)]`

ul_cell = `collect`

ul_cell is a valid Excel upper-left cell specified using standard Excel notation, and *lr_cell* is a valid Excel lower-right cell. If you specify *ul_cell* as the output location multiple times, the rightmost specification is the one written to the Excel file.

<i>set_options</i>	Description
<code>open</code>	open Excel file in memory
<code>modify</code>	modify Excel file
<code>replace</code>	overwrite Excel file
<code>sheet(sheetname[, replace])</code>	specify the worksheet to use; the default sheet name is Sheet1

<i>export_options</i>	Description
Main	
<code>overwritefmt</code>	overwrite existing cell formatting when exporting new content
<code>asdate</code>	convert Stata date (%td-formatted) <i>exp</i> to an Excel date
<code>asdatetime</code>	convert Stata datetime (%tc-formatted) <i>exp</i> to an Excel datetime
<code>asdatenum</code>	convert Stata date <i>exp</i> to an Excel date number, preserving the cell's format
<code>asdatetimenum</code>	convert Stata datetime <i>exp</i> to an Excel datetime number, preserving the cell's format
<code>names</code>	also write row names and column names for matrix <i>name</i> ; may not be combined with <code>rownames</code> or <code>colnames</code>
<code>rownames</code>	also write matrix row names for matrix <i>name</i> ; may not be combined with <code>names</code> or <code>colnames</code>
<code>colnames</code>	also write matrix column names for matrix <i>name</i> ; may not be combined with <code>names</code> or <code>rownames</code>
<code>colwise</code>	write results in <i>returnset</i> to consecutive columns instead of rows

<i>format_options</i>	Description
Number	
<code>nformat(<i>excelnfmt</i>)</code>	specify format for numbers
Alignment	
<code>left</code>	left-align text
<code>hcenter</code>	center text horizontally
<code>right</code>	right-align text
<code>top</code>	vertically align text with the top
<code>vcenter</code>	center text vertically
<code>bottom</code>	vertically align text with the bottom
<code>txtindent(#)</code>	indent text by # spaces; default is 0
<code>txtrotate(#)</code>	rotate text by # degrees; default is 0
<code>[no]txtwrap</code>	wrap text within each cell
<code>[no]shrinkfit</code>	shrink text to fit the cell width
<code>merge</code>	merge cells in <i>cellrange</i>
<code>unmerge</code>	separate merged cells identified by <i>ul_cell</i>
Font	
<code>font([<i>fontname</i>] [, <i>size</i> [, <i>color</i>]])</code>	specify font, font size, and font color
<code>[no]italic</code>	format text as italic
<code>[no]bold</code>	format text as bold
<code>[no]underline</code>	underline text in the specified cells
<code>[no]strikeout</code>	strikeout text in the specified cells
<code>script(sub super none)</code>	specify subscript or superscript formatting
Border	
<code>border(<i>border</i> [, <i>style</i> [, <i>color</i>]])</code>	specify horizontal and vertical cell border style
<code>dborder(<i>direction</i> [, <i>style</i> [, <i>color</i>]])</code>	specify diagonal cell border style
Fill	
<code>fpattern(<i>pattern</i> [, <i>fgcolor</i> [, <i>bgcolor</i>]])</code>	specify fill pattern for cells

Output types

`exp` writes a valid Stata expression to a cell; see [\[U\] 13 Functions and expressions](#). Stata dates and datetimes differ from Excel dates and datetimes. To properly export date and datetime values, use `asdate` and `asdatetime`.

`matrix(matname)` writes the values from a Stata matrix to Excel. Stata determines where to place the data in Excel by default from the size of the matrix (the number of rows and columns) and the location you specified in *ul_cell*. By default, *ul_cell* contains the first element of *matname*, and matrix row names and column names are not written.

`image(filename)` writes a portable network graphics (.png), JPEG (.jpg), Windows metafile (.wmf), device-independent bitmap (.dib), enhanced metafile (.emf), or bitmap (.bmp) file to an Excel worksheet. The upper-left corner of the image is aligned with the upper-left corner of the specified *ul_cell*. The image is not resized. If *filename* contains spaces, it must be enclosed in double quotes.

returnset is a shortcut name that is used to identify a group of [return](#) values. *returnset* may be any one of the following:

<i>returnset</i>	
<u>escalars</u>	<u>escalarmnames</u>
<u>rscalars</u>	<u>rscalarnames</u>
<u>emacros</u>	<u>emacronames</u>
<u>rmacros</u>	<u>rmacronames</u>
<u>ematrices</u>	<u>ematrixnames</u>
<u>rmatrices</u>	<u>rmatrixnames</u>
<u>e*</u>	<u>enames</u>
<u>r*</u>	<u>rnames</u>

`formula(formula)` writes an Excel formula to the cell specified in *ul_cell*. *formula* may be any valid Excel formula. Stata does not validate formulas; the text is passed literally to Excel.

`hyperlink(link, link_name)` writes a hyperlink to the cell specified in *ul_cell*. *link* may be an external file, a cell, or a webpage. Stata does not validate the links; the text is passed literally to Excel. Examples:

```
putexcel A1 = hyperlink("[auto.xlsx]Sheet1!C2", "Result")
putexcel A1 = hyperlink("https://www.stata.com", "StataCorp")
putexcel A1 = hyperlink(".\auto.xlsx", "1978 automobile data")
```

`etable[(#1 #2 ... #n)]` adds an automatically generated table to an Excel file starting in *ul_cell*. The table may be derived from the coefficient table of the last estimation command, from the table of margins after the last [margins](#) command, or from the table of results from one or more models displayed by [estimates table](#).

If the estimation command outputs $n > 1$ coefficient tables, the default is to add all tables and assign the corresponding table names *tablename1*, *tablename2*, ..., *tablename_n*. To specify which tables to add, supply the optional numlist to `etable`. For example, to add only the first and third tables from the estimation output, specify `etable(1 3)`. A few estimation commands do not support the `etable` output type. See [Unsupported estimation commands](#) in [\[RPT\] Appendix for putdocx](#) for a list of estimation commands that are not supported by `putexcel`.

`collect` adds a table from the current collection to an Excel file starting in *ul_cell*. This table may be created using `collect` or `table`. See [\[TABLES\] Intro](#) for more information on using `collect` to create a customized table from a collection of results from one or more Stata commands. See [\[R\] table intro](#) for information on using `table` to create tabulations, tables of summary statistics, tables of regression results, and more.

Options

Set

`open` permits `putexcel set` to open the Excel file in memory for modification. The Excel file is written to disk when `putexcel save` is issued.

`modify` permits `putexcel set` to modify an Excel file.

`replace` permits `putexcel set` to overwrite an existing Excel workbook. The workbook is overwritten when the first `putexcel` command is issued unless the `open` option is used.

`sheet(sheetname [, replace])` saves to the worksheet named *sheetname*. If there is no worksheet named *sheetname* in the workbook, then a new sheet named *sheetname* is created. If this option is not specified, `Sheet1` is used.

`replace` permits `putexcel` set to overwrite *sheetname* if it exists in the specified *filename*.

Main

`overwritefmt` causes `putexcel` to remove any existing cell formatting in the cell or cells to which it is writing new output. By default, all existing cell formatting is preserved. `overwritefmt`, when combined with a cell range, writes the cell format more efficiently.

`asdate` tells `putexcel` that the specified *exp* is a Stata `%td`-formatted date that should be converted to an Excel date with `m/d/yyyy` Excel date format.

This option has no effect if an *exp* is not specified as one of the output types.

`asdatetime` tells `putexcel` that the specified *exp* is a Stata `%tc`-formatted datetime that should be converted to an Excel datetime with `m/d/yyyy h:mm` Excel datetime format.

This option has no effect if an *exp* is not specified as one of the output types.

`asdatetimeum` tells `putexcel` that the specified *exp* is a Stata `%td`-formatted date that should be converted to an Excel date number, preserving the cell's format.

This option has no effect if an *exp* is not specified as one of the output types.

`asdatetimeum` tells `putexcel` that the specified *exp* is a Stata `%tc`-formatted datetime that should be converted to an Excel datetime number, preserving the cell's format.

This option has no effect if an *exp* is not specified as one of the output types.

`names` specifies that matrix row names and column names be written into the Excel worksheet along with the matrix values. If you specify `names`, then `ul_cell` will be blank, the cell to the right of it will contain the name of the first column, and the cell below it will contain the name of the first row. `names` may not be specified with `rownames` or `colnames`.

This option has no effect if `matrix()` is not specified as one of the output types.

`rownames` specifies that matrix row names be written into the Excel worksheet along with the matrix values. If you specify `rownames`, then `ul_cell` will contain the name of the first row. `rownames` may not be specified with `names` or `colnames`.

This option has no effect if `matrix()` is not specified as one of the output types.

`colnames` specifies that matrix column names be written into the Excel worksheet along with the matrix values. If you specify `colnames`, then `ul_cell` will contain the name of the first column. `colnames` may not be specified with `names` or `rownames`.

This option has no effect if `matrix()` is not specified as one of the output types.

`colwise` specifies that if a *returnset* is used, the values written to the Excel worksheet be written in consecutive columns. By default, the values are written in consecutive rows.

This option has no effect if a *returnset* is not specified as one of the output types.

Number

`nformat(excelfmt)` changes the numeric format of a cell range. Codes for commonly used formats are shown in the table of numeric formats in the [Appendix](#). However, any valid Excel format is permitted. Formats are formed from combinations of the following symbols.

Symbol	Description	Cell value	Fmt code	Cell displays
0	Digit placeholder (add zeros)	8.9	#.00	8.90
#	Digit placeholder (no zeros)	8.9	#.##	8.9
?	Digit placeholder (add space)	8.9	0.0?	8.9
.	Decimal point			
%	Percentage	.1	%	10%
,	Thousands separator	10000	#,###	10,000
E- E+ e- e+	Scientific format	12200000	0.00E+00	1.22E+07
\$-+/() : space	Display the symbol	12	(000)	(012)
\	Escape character	3	0\!	3!
*	Repeat character (fill in cell width)	3	3*	3xxxxx
—	Skip width of next character	−1.2	_0.0	1.2
"text"	Display text in quotes	1.23	0.00 "a"	1.23 a
@	Text placeholder	b	"a"@ "c"	abc

Formats that contain spaces must be enclosed in double quotes.

Alignment

`left` sets the specified cells to have contents left-aligned within the cell. `left` may not be combined with `right` or `hcenter`. Right-alignment is the Excel default for numeric values and need not be specified when outputting numbers.

`hcenter` sets the specified cells to have contents horizontally centered within the cell. `hcenter` may not be combined with `left` or `right`.

`right` sets the specified cells to have contents right-aligned within the cell. `right` may not be combined with `left` or `hcenter`. Left-alignment is the Excel default for text and need not be specified when outputting strings.

`top` sets the specified cells to have contents vertically aligned with the top of the cell. `top` may not be combined with `bottom` or `vcenter`.

`vcenter` sets the specified cells to have contents vertically aligned with the center of the cell. `vcenter` may not be combined with `top` or `bottom`.

`bottom` sets the specified cells to have contents vertically aligned with the bottom of the cell. `bottom` may not be combined with `top` or `vcenter`.

`txtindent(#)` sets the text indentation in each cell in a cell range. `#` must be an integer between 0 and 15.

`txtrotate(#)` sets the text rotation in each cell in a cell range. `#` must be an integer between 0 and 180 or equal to 255. `txtrotate(0)` is equal to no rotation and is the default. `txtrotate(255)` specifies vertical text. Values 1–90 rotate the text counterclockwise 1 to 90 degrees. Values 91–180 rotate the text clockwise 1 to 90 degrees.

`txtwrap` and `nottxtwrap` specify whether the text is to be wrapped in a cell or within each cell in a range of cells. The default is no wrapping. `nottxtwrap` has an effect only if the cell or cells were previously formatted to wrap. `txtwrap` may not be specified with `shrinkfit`.

`shrinkfit` and `noshrinkfit` specify whether the text is to be shrunk to fit in the cell width of a cell or in each cell of a range of cells. The default is no shrinking. `noshrinkfit` has an effect only if the cell or cells were previously formatted to shrink text to fit. `shrinkfit` may not be specified with `txtwrap`.

`merge` tells Excel to merge cells in the specified cell range. `merge` may be combined with `left`, `right`, `hcenter`, `top`, `bottom`, and `vcenter` to format the merged cell. Merging cells that contain data in each cell will result in the upper-leftmost data being kept.

Once you have merged cells, you can refer to the merged cell by using any single cell from the specified *cellrange*. For example, if you specified a *cellrange* of A1:B2, you could refer to the merged cell using A1, B1, A2, or B2.

`unmerge` tells Excel to unmerge previously merged cells. When using `unmerge`, you only need to use a single cell from the merged cell in the previously specified *cellrange*.

Font

`font([fontname] [, size [, color]])` sets the font, font size, and font color for each cell in a cell range. The font size and font color may be specified individually without specifying *fontname*. Use `font("", size)` to specify font size only. Use `font("", "", color)` to specify font color only. For both cases, the default font will be used. If `font()` is not specified, the Excel defaults are preserved.

fontname may be any valid Excel font. If *fontname* includes spaces, then it must be enclosed in double quotes. What constitutes a valid Excel font is determined by the version of Excel that is installed on the user's computer.

size is a numeric value that represents any valid Excel font size. The default is 12.

color may be a valid RGB value in the form "### ##" or may be one of the colors listed in the table of colors in the [Appendix in \[RPT\] putexcel](#). If no *color* is specified, then Excel workbook defaults are used.

`italic` and `noitalic` specify whether to italicize or unitalicize the text in a cell or range of cells. The default is for text to be unitalicized. `noitalic` has an effect only if the cell or cells were previously italicized.

`bold` and `nobold` specify whether to bold or unbold the text in a cell or range of cells. The default is for text to be unbold. `nobold` has an effect only if the cell or cells were previously formatted as bold.

`underline` and `nounderline` specify whether to underline the text or remove the underline from the text in a cell or range of cells. The default is for text not to be underlined. `nounderline` has an effect only if the cell or cells previously contained underlined text.

`strikeout` and `nostrikeout` specify whether to strikeout the text or remove the strikeout from the text in a cell or range of cells. The default is for text not to have a strikeout mark. `nostrikeout` has an effect only if the cell or cells previously had a strikeout mark.

`script(sub | super | none)` changes the script style of the cell. `script(sub)` makes all text in a cell or range of cells a subscript. `script(super)` makes all text in a cell or range of cells a superscript. `script(none)` removes all subscript or superscript formatting from a cell or range of cells. Specifying `script(none)` has an effect only if the cell or cells were previously formatted as subscript or superscript.

Border

`border`(*border* [, *style* [, *color*]]) sets the cell border, style, and color for a cell or range of cells.

border may be `all`, `left`, `right`, `top`, or `bottom`.

style is a keyword specifying the look of the border. The most common styles are `thin`, `medium`, `thick`, and `double`. The default is `thin`. For a complete list of border styles, see the [Appendix](#) in [RPT] **putexcel**. To remove an existing border, specify `none` as the *style*.

color may be a valid RGB value in the form "`### #`" or may be one of the colors listed in the table of colors in the [Appendix](#) in [RPT] **putexcel**. If no *color* is specified, then Excel workbook defaults are used.

`dborder`(*direction* [, *style* [, *color*]]) sets the cell diagonal border direction, style, and color for a cell or range of cells.

direction may be `down`, `up`, or `both`. `down` draws a line from the upper-left corner of the cell to the lower-right corner of the cell or, for a range of cells, from the upper-left corner of *ul_cell* to the lower-right corner of *lr_cell*. `up` draws a line from the lower-left corner of the cell to the upper-right corner of the cell or, for a range of cells, from the lower-left corner of the area defined by *ul_cell*:*lr_cell* to the upper-right corner.

style is a keyword specifying the look of the border. The most common styles are `thin`, `medium`, `thick`, and `double`. The default is `thin`. For a complete list of border styles, see the [Appendix](#) in [RPT] **putexcel**. To remove an existing border, specify `none` as the *style*.

color may be a valid RGB value in the form "`### #`" or may be one of the colors listed in the table of colors in the [Appendix](#) in [RPT] **putexcel**. If no *color* is specified, then Excel workbook defaults are used.

Fill

`fpattern`(*pattern* [, *fgcolor* [, *bcolor*]]) sets the fill pattern, foreground color, and background color for a cell or range of cells.

pattern is a keyword specifying the fill pattern. The most common fill patterns are `solid` for a solid color (determined by *fgcolor*), `gray25` for 25% gray scale, `gray50` for 50% gray scale, and `gray75` for 75% gray scale. A complete list of fill patterns is shown in the [Appendix](#) of [RPT] **putexcel**. To remove an existing fill pattern from the cell or cells, specify `none` as the *pattern*.

fgcolor specifies the foreground color. The default foreground color is `black`. *fgcolor* may be a valid RGB value in the form "`### #`" or may be any of the colors listed in the table of colors in the [Appendix](#) in [RPT] **putexcel**.

bcolor specifies the background color. *bcolor* may be a valid RGB value in the form "`### #`" or may be any of the colors listed in the table of colors in the [Appendix](#) in [RPT] **putexcel**. If no *bcolor* is specified, then Excel workbook defaults are used.

Remarks and examples

If you have not already read [Remarks and examples](#) in [RPT] **putexcel**, please do so now. The examples here build on the examples shown there.

Remarks are presented under the following headings:

[Writing expressions and formatting cells](#)
[Using formulas](#)
[Exporting estimation results](#)

Writing expressions and formatting cells

Before we can write to an Excel workbook using `putexcel`, we need to tell Stata what the destination is. We do this using the `putexcel set` command. For the next several examples, we will use an Excel file named `myresults2.xlsx`. We will begin with a sheet named `Descriptive`.

```
. putexcel set myresults2.xlsx, sheet(Descriptive)
```

If we had not specified the sheet name, `putexcel` would have defaulted to using the first sheet in the workbook.

► Example 1: Write multiple expressions and format cells simultaneously

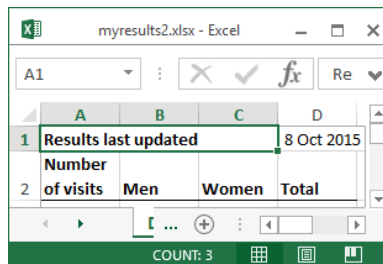
Suppose we want to write the same headers and format them as shown in [example 2](#) and [example 3](#) of [\[RPT\] putexcel](#). Using the advanced syntax of `putexcel`, we can do this with a single `putexcel` command.

```
. putexcel A1="Variable" B1="Men" C1="Women", bold border(bottom)
file myresults2.xlsx saved
```

Rather than beginning with the table headers, we could improve record keeping by including the date we generated the results. To do this, we can use the system parameter, or c-class return value, `c(current_date)`, which will add a string date in the format *dd Mon yyyy*; see [\[P\] creturn](#). We also merge cells A1 and B1 for aesthetic reasons. Notice that we type `"'c(current_date)'"` instead of `"c(current_date)"`. The `'` indicate macro substitution; see [\[P\] macro](#). We specify `replace` as a suboption to `sheet()` in a new `putexcel set` command to overwrite our previous output. Because the workbook `myresults2.xlsx` already exists, we also need to `replace`, or `modify`, the workbook.

```
. putexcel set myresults2.xlsx, replace sheet(Descriptive, replace)
note: file will be replaced when the first putexcel command is issued.
. putexcel A1 = "Results last updated" D1 = "'c(current_date)'"
file myresults2.xlsx saved
. putexcel (A1:C1), merge
file myresults2.xlsx saved
. putexcel A2="Variable" B2="Men" C2="Women", bold border(bottom)
file myresults2.xlsx saved
```

The above commands give an Excel file with a header row that looks like this:



	A	B	C	D
1	Results last updated			8 Oct 2015
2	Number of visits	Men	Women	Total

Using formulas

Writing Excel formulas is useful when you want to perform calculations using the output of Stata commands. Formulas are passed verbatim to Excel, so you must use the correct Excel function (not Stata function) for what you want to accomplish. You may find it helpful to experiment in Excel first and then copy the formula over to your [do-file](#) to keep a record of your work and in case you need to run your analysis again later.

► Example 2: Adding a total row and total column

In [example 4](#) of [\[RPT\] putexcel](#), we obtained the number of females and number of males in the `website` dataset and wrote these out to Excel. Suppose instead that we want the number of males and females at each number of visits and the total number of visits.

We can change the heading for the first column from “Variable” to “Number of visits” and add a column for “Total”. We specify the `txtwrap` option with our `putexcel` command so that the long text “Number of visits” wraps within cell A2.

We use the `matcell()` option with [tabulate](#) to save the cell frequencies to a matrix and the `matrow()` option to save the row values from the table.

```
. use https://www.stata-press.com/data/r19/website
(Visits to website)
. putexcel A2="Number of visits" D2="Total", txtwrap bold border(bottom)
file myresults2.xlsx saved
```

```
. tabulate visits female, matrow(nvisits) matcell(freq)
```

Visits to website	Female		
	0	1	Total
1	18	17	35
2	58	42	100
3	32	46	78
4	35	25	60
5	24	40	64
6	21	17	38
7	21	21	42
8	12	7	19
9	8	10	18
10	8	5	13
11	3	5	8
12	4	1	5
13	2	2	4
14	4	0	4
15	1	1	2
16	1	1	2
18	1	0	1
20	2	1	3
22	1	0	1
23	0	1	1
27	1	0	1
51	0	1	1
Total	257	243	500

```
. putexcel A3=matrix(nvisits) B3=matrix(freq)
file myresults2.xlsx saved
```

Totals, however, are not saved. By using formulas in Excel, we can add the row and column totals. Our results will begin on row 3, so we will want to begin calculating row totals here. For example, in Excel the row total for males (column B) and females (column C) in row 3 is $D3=B3+C3$. The fastest way to write this formula multiple times for each row is to use `forvalues`; see [P] [forvalues](#).

```
. forvalues i=3/24 {
  2.     putexcel D`i'=formula(B`i'+C`i')
  3. }
file myresults2.xlsx saved
(output omitted)
```

We also want column totals. For this, it is easier to use Excel's `SUM()` function. We supply this as the `formula()` output type and add a border above the total row to set it apart from the rest of the table.

```
. putexcel A25="Total" B25=formula(SUM(B3:B24)) C25=formula(SUM(C3:C24))
> D25=formula(SUM(D3:D24)), bold border(top)
file myresults2.xlsx saved
```

Note that while this example demonstrates how you can easily take advantage of Excel's formulas, you can add results from a tabulation that includes totals in a simpler method. See [R] [table twoway](#) to learn about creating tabulations that store results in a collection. With results from `table`, you can use `putexcel`'s `collect` output type to include the full table in an Excel document. See [example 7](#) and [example 8](#) in [RPT] [putexcel](#) for examples of `putexcel`'s `collect` output type.

◀

Exporting estimation results

We start by using `putexcel` `set` again to create a new worksheet for our regression results, modifying our existing workbook.

```
. putexcel set myresults2.xlsx, sheet(Estimation) modify
```

► Example 3: Export point estimates and formatted confidence intervals

We continue from [example 6](#) of [RPT] [putexcel](#), where we gave the coefficients the title “Coef.”. Here we add a column named “C.I.” for the confidence interval, which we center in cells C1 and D1.

```
. putexcel B1="Coef." C1="C.I."
file myresults2.xlsx saved
. putexcel (C1:D1), merge hcenter
file myresults2.xlsx saved
```

We fit the same model that we did in [example 6](#) of [RPT] [putexcel](#). We copy the `r-class` return `r(table)`, which contains the values that were returned by the command in the estimation results table, into a new matrix named `table`.

```
. quietly regress visits ad female time
. matrix table = r(table)
. matrix list table
table[9,4]
      ad      female      time      _cons
b      .79961794   -.04679974   .82961995   .69243389
se      .05165911   .2096816    .04366007   .20079144
t       15.47874    -.22319432   19.001801   3.448523
pvalue   2.235e-44   .82347616    6.794e-61   .00061166
ll       .69812028   -.45877341   .74383847   .29792725
ul       .90111561   .36517393    .91540143   1.0869405
df        496        496        496        496
crit     1.9647583   1.9647583    1.9647583   1.9647583
eform        0        0        0        0
```

We then select row 1 for the coefficients (b), row 5 for the lower limit of the confidence interval (ll), and row 6 for the upper limit of the confidence interval (ul) into separate vectors. We take the transpose to ensure that our results for each variable are in a row rather than in a column when we write them out. We specify `nformat(number_d2)` for the coefficients, but we will specify a custom format for the confidence interval. We also add the `rownames` option so that the variable names are written out along with the coefficient estimates.

```
. matrix b = table[1, 1...]'
. matrix ll = table[5, 1...]'
. matrix ul = table[6, 1...]'
. putexcel A2=matrix(b), rownames nformat(number_d2)
file myresults2.xlsx saved
```

We want our confidence interval to be displayed as (0.74 to 0.92), taking `time`, for example. That is, the numbers should have a 0 before the decimal and be rounded to two decimal places. The lower and upper limits should be separated by the word “to”, and the confidence interval should be enclosed in parentheses.

We specify our format in two steps. For the lower limit, we ensure that the negative sign will display inside the parentheses by specifying separate formats for positive and negative numbers. The code for positive numbers is specified before the semicolon. The code for negative numbers is specified after the semicolon. We add the word “to” so that it is printed between the lower and upper limit values. For the upper limit, the negative sign will display in front of the number by default, so we only have to specify one format.

```
. putexcel C2=matrix(ll), nformat("(0.00 to;(-0.00 to)") right
file myresults2.xlsx saved
. putexcel D2=matrix(ul), nformat("0.00)") left
file myresults2.xlsx saved
```

The above commands give a table that looks like this:

	Coef.	C.I.
ad	0.80	(0.70 to 0.90)
female	-0.05	(-0.46 to 0.37)
time	0.83	(0.74 to 0.92)
_cons	0.69	(0.30 to 1.09)

This example demonstrates how to use `putexcel`'s formatting options in combination with specifying the output that is to appear in the cells. However, if your goal is to create a regression table and customize the format of the confidence interval, the `collect` and `table` commands provide another, sometimes more direct, way to do this. See [example 7](#) in [\[RPT\] putexcel](#) for an example of creating a regression table with customized confidence intervals using `collect` and exporting it using `putexcel`'s `collect` output type.

◀

► Example 4: Export SEM estimates and path diagram

Continuing [example 3](#), suppose we also fit a corresponding linear regression model using `sem` and want to export the coefficient estimates, confidence intervals, and the path diagram. We use the same general approach as in [example 3](#) but with some modification. First, we specify the range of columns from table to be 1 through 4 when we create our `b`, `ll`, and `ul` vectors to exclude the variance.

```
. quietly sem (visits <- ad female time)
. matrix table = r(table)
. matrix list table
table[9,5]
      visits:      visits:      visits:      visits:  var(e.vis~):
      ad      female      time      _cons      _cons
b      .79961794  -.04679974  .82961995  .69243389  5.4458576
se      .05145206  .20884119  .04348508  .19998666  .34442628
z      15.541029  -.22409249  19.078268  3.4624004  .b
pvalue  1.830e-54  .82268533  3.827e-81  .00053538  .b
ll      .69877376  -.45612096  .74439077  .30046724  4.810958
ul      .90046212  .36252147  .91484914  1.0844005  6.1645445
df      .          .          .          .          .
crit    1.959964  1.959964  1.959964  1.959964  1.959964
eform   0          0          0          0          0
. matrix b = table[1, 1..4]'
. matrix ll = table[5, 1..4]'
. matrix ul = table[6, 1..4]'
```

If we wanted that term as well, we could select it separately. Because we do not need to specify row names again, we also specify the `ul_cell` for `b` in reference to its values, not the labels this time. We use the same numeric format for `ll` and `ul`.

```
. putexcel E2=matrix(b), nformat(number_d2)
file myresults2.xlsx saved

. putexcel F2=matrix(ll), nformat("(0.00 to;(-0.00 to)") right
file myresults2.xlsx saved

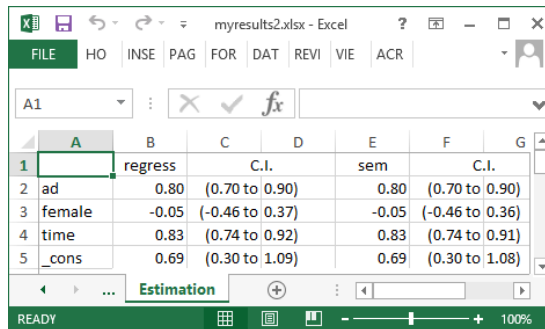
. putexcel G2=matrix(ul), nformat("0.00)") left
file myresults2.xlsx saved
```

Because we are adding estimation results, we change “Coef.” to “regress” and add a column heading for “sem”. As before, we include a merged column heading for the confidence interval. We center all column headings by using the `hcenter` option.

```
. putexcel B1="regress" E1="sem" F1="C.I.", hcenter
file myresults2.xlsx saved

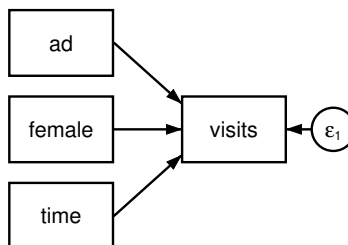
. putexcel (F1:G1), merge
file myresults2.xlsx saved
```

Our table from [example 3](#) now includes additional columns for our sem results:



	regress	C.I.	sem	C.I.
ad	0.80	(0.70 to 0.90)	0.80	(0.70 to 0.90)
female	-0.05	(-0.46 to 0.37)	-0.05	(-0.46 to 0.36)
time	0.83	(0.74 to 0.92)	0.83	(0.74 to 0.91)
_cons	0.69	(0.30 to 1.09)	0.69	(0.30 to 1.08)

We might also want to export the path diagram for our model so that we can view it while looking at our results.



The SEM Builder will allow you to save your path diagram as a PNG file, in addition to other file types; PNGs can be exported to Excel. We name our path diagram `visits_sem` and save it as a PNG, and we then output the file to Excel with the `image()` output type.

```
. putexcel B6 = image(visits_sem.png)
file myresults2.xlsx saved
```

This adds the path diagram with the upper-left corner aligned in the upper-left corner of cell B6.

□ Technical note

See the technical notes *Excel data size limits* and *Dates and times* in [D] **import excel**.



References

- Crow, K. 2013. Export tables to Excel. *The Stata Blog: Not Elsewhere Classified*. <https://blog.stata.com/2013/09/25/export-tables-to-excel/>.
- . 2018. Export tabulation results to Excel—update. *The Stata Blog: Not Elsewhere Classified*. <https://blog.stata.com/2018/06/07/export-tabulation-results-to-excel-update/>.
- Gallup, J. L. 2012. *A new system for formatting estimation tables*. *Stata Journal* 12: 3–28.
- Huber, C. 2017. Creating Excel tables with putexcel, part 3: Writing custom reports for arbitrary variables. *The Stata Blog: Not Elsewhere Classified*. <https://blog.stata.com/2017/04/06/creating-excel-tables-with-putexcel-part-3-writing-custom-reports-for-arbitrary-variables/>.
- Quintó, L. 2012. *HTML output in Stata*. *Stata Journal* 12: 702–717.

Also see

- [RPT] **putexcel** — Export results to an Excel file
- [RPT] **putdocx intro** — Introduction to generating Office Open XML (.docx) files
- [RPT] **putpdf intro** — Introduction to generating PDF files
- [D] **import excel** — Import and export Excel files
- [M-5] **_docx*()** — Generate Office Open XML (.docx) file
- [M-5] **Pdf*()** — Create a PDF file
- [M-5] **xl()** — Excel file I/O class
- [P] **postfile** — Post results in Stata dataset
- [P] **return** — Return stored results
- [R] **table intro** — Introduction to tables of frequencies, summaries, and command results
- [TABLES] **Intro** — Introduction

Description

The putpdf suite of commands creates PDF documents that include text, formatted images, and tables of Stata estimation results and summary statistics. The following commands are used to create, format, add content to, and save PDF documents.

Create and save PDF files (see [RPT] [putpdf begin](#))

putpdf begin	Creates a PDF file for export
putpdf describe	Describes contents of the active PDF file
putpdf save	Saves and closes the PDF file
putpdf clear	Closes the PDF file without saving the changes

Insert page breaks in a PDF file (see [RPT] [putpdf pagebreak](#))

putpdf pagebreak	Adds a page break to the document
putpdf sectionbreak	Adds a new section to the document

Add paragraph with text and images (see [RPT] [putpdf paragraph](#))

putpdf paragraph	Adds a new paragraph to the active document
putpdf text	Adds text to the active paragraph
putpdf image	Appends an image to the active paragraph

Add tables to a PDF file (see [RPT] [putpdf table](#))

putpdf table	Creates a new table in the PDF file containing estimation results, summary statistics, or data in memory
--------------	--

Add a table from a collection to a PDF file (see [RPT] [putpdf collect](#))

putpdf collect	Adds a customized table created by collect or table to the PDF file
----------------	---

In this manual entry, we show you how to use the putpdf commands by walking you through an example that creates a simple report as a PDF file.

Remarks and examples

Remarks are presented under the following headings:

[Introduction](#)

[Create a PDF file](#)

[Add a paragraph with text](#)

[Add an image to a paragraph](#)

[Add table of estimation results](#)

Introduction

putpdf is a suite of commands used to write paragraphs, images, and tables to a PDF file. This allows you to create PDF files that include Stata results and graphs. With putpdf, you can also format the text, tables, and images in the document you create.

To get started with the putpdf commands, it is best to see them in action. Here we demonstrate how to create a PDF file, include text, add a graph, and incorporate an estimation table all from within Stata.

This example shows the basic tools you need to create your own document. However, this is only a starting point. You may want to create more extensive and more customized documents, and putpdf allows you to do that. We save the details of customizing text, tables, and images for the individual entries of the commands listed [above](#).

Create a PDF file

To demonstrate, we create a report on 1978 automobiles using `auto.dta`.

```
. use https://www.stata-press.com/data/r19/auto
(1978 automobile data)
```

Before we can add any content to the report, we first need to create an active `.pdf` document in memory. We do this with the `putpdf begin` command.

```
. putpdf begin
```

Because we did not include any options with `putpdf begin`, the document created uses the `letter` page size and the portrait orientation.

Add a paragraph with text

Now that the document is created, we can add other objects such as paragraphs, images, and tables to it. We begin by adding a title to our report using the Courier font with size 20 and centering it.

```
. putpdf paragraph, font("Courier",20) halign(center)
. putpdf text ("Report on 1978 automobiles")
```

We added strings to this paragraph, but text can include any valid Stata expression. Below, we add text with summary statistics for our data by referring directly to the results stored after `summarize`. We type `return list` and see that `r(N)`, `r(mean)`, and `r(max)` store the number of automobiles, the average MPG, and the maximum MPG among those automobiles.

```
. summarize mpg
```

Variable	Obs	Mean	Std. dev.	Min	Max
mpg	74	21.2973	5.785503	12	41

```
. return list
```

```
scalars:
```

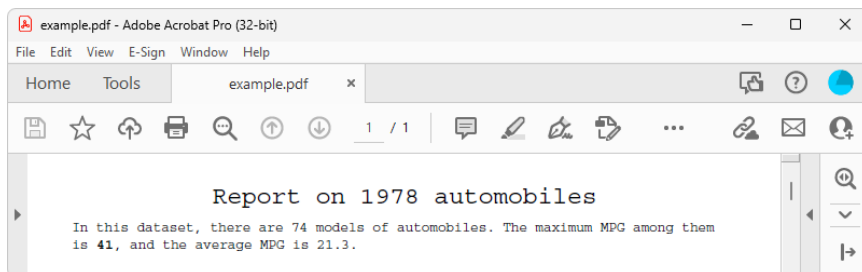
```

      r(N) = 74
    r(sum_w) = 74
    r(mean) = 21.2972972972973
    r(Var) = 33.47204738985561
      r(sd) = 5.785503209735141
    r(min) = 12
    r(max) = 41
    r(sum) = 1576
```

We add a new paragraph and then refer to these scalars directly as we add text.

```
. putpdf paragraph, font("Courier")
. putpdf text ("In this dataset, there are 'r(N)'")
. putpdf text (" models of automobiles. The maximum MPG among them is ")
. putpdf text (r(max)), bold
. putpdf text (" , and the average MPG is ")
. putpdf text (r(mean)), nformat("%.1f")
. putpdf text ("."), linebreak(2)
```

We formatted `r(max)` as bold and `r(mean)` with one decimal place. We also inserted two line breaks at the end of our text to create some space between this content and the next element we export. If we were to save our document now, it would contain the following:



Add an image to a paragraph

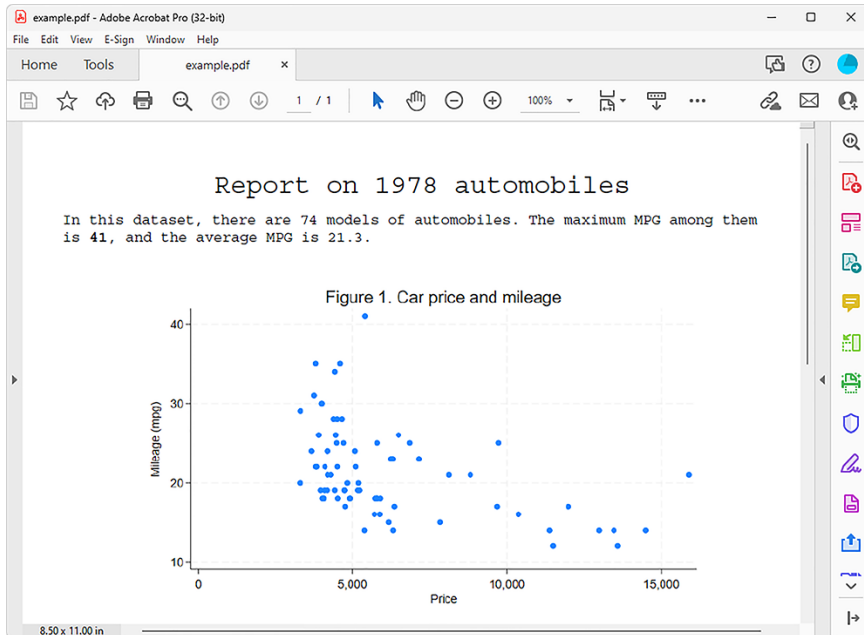
Next we add a scatterplot to our document, showing how mileage (mpg) correlates with the price (price) of cars. We use `scatter` to create our scatterplot, specifying a title for the graph. We must convert the graph to one of the supported image formats, `.png` or `.jpg`. We save it as a `.png` file with `graph export`.

```
. scatter mpg price, title("Figure 1. Car price and mileage")
. graph export auto.png
file auto.png saved as PNG format
```

Now we use `putpdf image` to append it to the active paragraph. To center the image, we specify the alignment of the paragraph. We resize the image by setting the width at 6 inches and add a line break after the image.

```
. putpdf paragraph, halign(center)
. putpdf image auto.png, linebreak width(6)
```

If we were to save the document now, it would contain the following:



Add table of estimation results

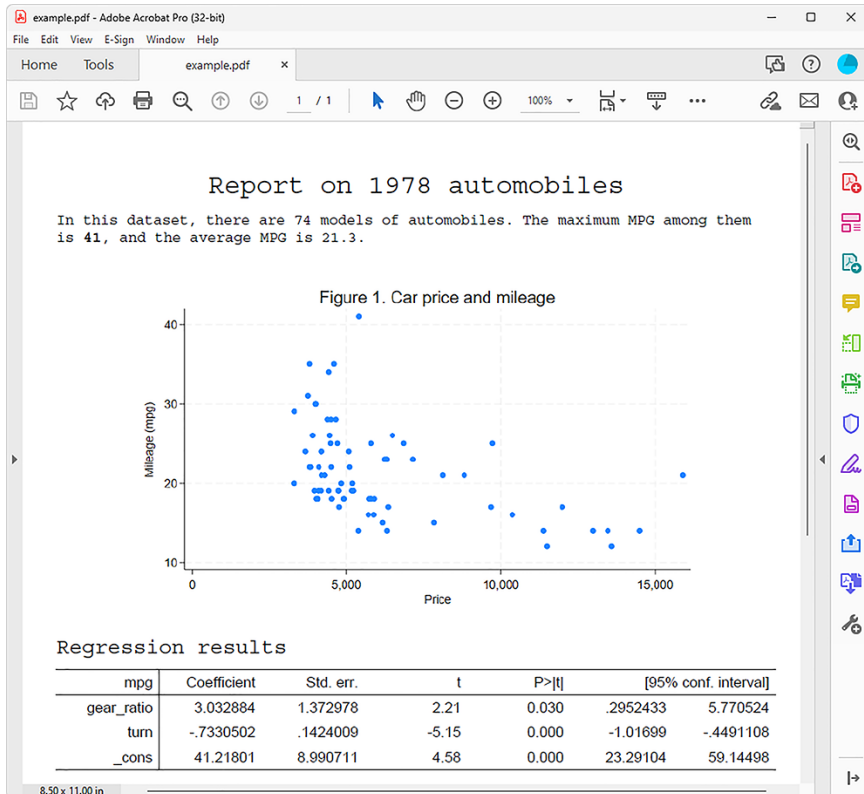
Next we will add a table with regression results after modeling fuel efficiency as a function of the car's gear ratio (`gear_ratio`) and turn radius (`turn`). We export a table named `tbl1` with all the statistics shown in the regression output below. This is done effortlessly by specifying the `etable` output type with `putpdf table`.

```
. putpdf paragraph, font("Courier",16)
. putpdf text ("Regression results")
. regress mpg gear_ratio turn, noheader
```

mpg	Coefficient	Std. err.	t	P> t	[95% conf. interval]	
gear_ratio	3.032884	1.372978	2.21	0.030	.2952433	5.770524
turn	-.7330502	.1424009	-5.15	0.000	-1.01699	-.4491108
_cons	41.21801	8.990711	4.58	0.000	23.29104	59.14498

```
. putpdf table tbl1 = etable
. putpdf save example.pdf
successfully created "C:/mypath/example.pdf"
```

After saving our document, `example.pdf` appears as follows:



We will consider this our final report. However, you will likely want to create PDF files with more content and perhaps more customization. See [RPT] [putpdf begin](#) for information on formatting the document as a whole, including specifying page size, page layout, and font. See [RPT] [putpdf paragraph](#) for information on adding text to a document; modifying the style, font, alignment, and other formatting of a paragraph; and customizing the size and location of an image. See [RPT] [putpdf table](#) for information on creating tables from stored results, matrices, data, and even images and for information on customizing these tables. See [RPT] [putpdf collect](#) for information on adding customized tables created with the collect suite of commands to a document. Finally, see [RPT] [putpdf pagebreak](#) for information on adding page breaks and section breaks to your document.

Also see

[RPT] [putpdf begin](#) — Create a PDF file

[RPT] [putpdf collect](#) — Add a table from a collection to a PDF file

[RPT] [putpdf pagebreak](#) — Add breaks to a PDF file

[RPT] [putpdf paragraph](#) — Add text or images to a PDF file

[RPT] [putpdf table](#) — Add tables to a PDF file

Description

`putpdf begin` creates a PDF file. This is the active document that the remaining `putpdf` commands modify.

`putpdf describe` describes the active PDF file.

`putpdf save` saves and closes the PDF file.

`putpdf clear` closes the PDF file without saving.

Quick start

Create a document in memory onto which subsequent contents are added

```
putpdf begin
```

Same as above, but with 1.5 inch margins on the left and right

```
putpdf begin, margin(left,1.5) margin(right,1.5)
```

Save the document in memory to disk as `myfile.pdf`

```
putpdf save myfile.pdf
```

Same as above, but overwrite `myfile.pdf` if it already exists

```
putpdf save myfile.pdf, replace
```

Close the document in memory without saving the changes

```
putpdf clear
```

Syntax

Create document for export

```
putpdf begin [ , begin_options ]
```

Describe active document

```
putpdf describe
```

Save and close document

```
putpdf save filename [ , replace nomsg ]
```

Close without saving

```
putpdf clear
```

<i>begin_options</i>	Description
<code>pagesize(<i>psize</i>)</code>	set document page size
<code>landscape</code>	change document orientation to landscape
<code>font(<i>fspec</i>)</code>	set font, font size, and font color for the document
<code>halign(<i>hvalue</i>)</code>	set horizontal alignment for the document
<code>margin(<i>type</i>, #[<i>unit</i>])</code>	set page margins for the document
<code>bgcolor(<i>color</i>)</code>	set background color

collect is allowed with `putpdf describe`; see [U] 11.1.10 Prefix commands.

Options

Options are presented under the following headings:

Options for putpdf begin

Options for putpdf save

Options for putpdf begin

`pagesize(psize)` sets the page size of the document. *psize* may be letter, legal, A3, A4, A5, B4, or B5. The default is `pagesize(letter)`.

`landscape` changes the document orientation from portrait (the default) to landscape.

`font(fontname [, size [, color]])` sets the font, font size, and font color for the document.

fontname may be any supported font installed on the user's computer. Base 14 fonts, Type 1 fonts (*.pfa* or *.pfb*), TrueType fonts (*.ttf* or *.ttc*), and OpenType fonts (*.otf*) are supported. TrueType and OpenType fonts that cannot be embedded may not be used. If *fontname* includes spaces, then it must be enclosed in double quotes. The default font is Helvetica.

size is a numeric value that represents font size measured in points. The default is 11.

color sets the text color. *color* may be one of the colors listed in [Colors](#) of [\[RPT\] Appendix for putpdf](#); a valid RGB value in the form `### ###`, for example, 171 248 103; or a valid RRGGBB hex value in the form `#####`, for example, ABF867.

The font size and font color may be specified individually without specifying *fontname*. Use `font("", size)` to specify font size only. Use `font("", "", color)` to specify font color only. For both cases, the default font will be used.

`halign(hvalue)` sets the horizontal alignment of the document within the paragraphs, images, and tables. *hvalue* may be left, right, or center. The default is `halign(left)`.

`margin(type, #[unit])` sets the page margins of the document. This option may be specified multiple times in a single command to account for different margin settings.

type identifies the location of the margin inside the document. *type* may be top, left, bottom, right, or all.

unit may be in (inch), pt (point), cm (centimeter), or twip (twentieth of a point). An inch is equivalent to 72 points, 2.54 centimeters, or 1440 twips. The default is in.

`bgcolor(color)` sets the background color for the document. *color* may be one of the colors listed in [Colors](#) of [\[RPT\] Appendix for putpdf](#); a valid RGB value in the form `### ###`, for example, 171 248 103; or a valid RRGGBB hex value in the form `#####`, for example, ABF867.

Options for putpdf save

`replace` specifies to overwrite *filename*, if it exists, with the contents of the document in memory.

`nomsg` suppresses the message that contains a link to *filename*.

Remarks and examples

Remarks are presented under the following headings:

[Creating and formatting a PDF file](#)

[Describing the document](#)

[Saving or clearing the PDF file](#)

Creating and formatting a PDF file

Before we can write to a PDF file using `putpdf`, we need to create an active PDF document in memory by using the `putpdf begin` command. We can simply type

```
. putpdf begin
```

to create a document. We could now add content to this document. For information on adding text or images to the document, see [\[RPT\] putpdf paragraph](#). For information on adding tables to the document, see [\[RPT\] putpdf table](#).

Because we did not include any options in the above `putpdf begin` command, it creates a letter-size document with pages in portrait orientation. We can specify other formats for the document as a whole by using the options available with `putpdf begin`. We can specify the page size, page orientation, and font properties for the document.

For example, below we create a legal-sized PDF file in memory with the font Courier.

```
. putpdf begin, pagesize(legal) font("Courier")
```


The page size and orientation specified with `putpdf begin` will remain in effect until a [section break](#) is added. On the other hand, the font specification can be modified for every [paragraph](#) and addition of [text](#).

Describing the document

To view a document, we must first save the file. However, if we have been adding text, images, and tables to a document, we can describe the contents of the active document without saving it.

```
. putpdf describe
```

This reports the number of paragraphs and tables that have been added to the document.

Saving or clearing the PDF file

When we have finished adding content to our document, we can save it under a given filename, say, `myfile.pdf`.

```
. putpdf save myfile.pdf
```

If the file already exists in the saving directory, we will need to specify the `replace` option, which will overwrite the existing contents in the file.

Suppose we have mistakenly added an image with the wrong dimensions. Rather than saving the document in this situation, we can clear the active document from memory without saving it by typing

```
. putpdf clear
```

This command clears the document in memory and automatically closes the document without saving.

References

- Jann, B. 2016. [Creating L^AT_EX documents from within Stata using texdoc](#). *Stata Journal* 16: 245–263.
- Rodríguez, G. 2017. [Literate data analysis with Stata and Markdown](#). *Stata Journal* 17: 600–618.
- Weinreb, M. D., and J. Trinitapoli. 2022. [printcase: A command for visualizing single observations](#). *Stata Journal* 22: 958–968.

Also see

- [RPT] [putpdf intro](#) — Introduction to generating PDF files
- [RPT] [putpdf collect](#) — Add a table from a collection to a PDF file
- [RPT] [putpdf pagebreak](#) — Add breaks to a PDF file
- [RPT] [putpdf paragraph](#) — Add text or images to a PDF file
- [RPT] [putpdf table](#) — Add tables to a PDF file
- [RPT] [Appendix for putpdf](#) — Appendix for putpdf entries

Description

putpdf collect allows you to export a customized table from a collection to a table in the active PDF file. A collection contains a set of results that have been collected from one or more Stata commands using the collect: prefix or the collect get command. With the suite of collect commands, you can specify the layout and style of your table and customize the table.

Unlike putpdf table, which allows you to create tables and modify them as needed, putpdf collect is designed for exporting a table that you have already finalized using the collect suite of commands. To learn more about creating customized tables, see [TABLES] Intro.

putpdf collect also allows you to include tables created by table and etable in your report. The table command is a powerful command for producing tabulations, tables of summary statistics, tables of regression results, and more. The etable command creates tables with the active estimation results, results from margins, and results stored with estimates store. table and etable are unique in that they automatically create a collection. Their results can be further styled using the collect commands, or they can be included in your report as is with putpdf collect. See [R] table intro and [R] etable for more information on these commands.

Quick start

Create a table in the document using items from the current collection

```
putpdf collect
```

Same as above, and display the putpdf commands used to export to the PDF file

```
putpdf collect, noisily
```

Same as above, but instead of displaying the commands, save them to the file myfile.do

```
putpdf collect, dofile(myfile.do)
```

Syntax

```
putpdf collect [ , options ]
```

options	Description
name(<i>cname</i>)	use collection <i>cname</i>
<u>me</u> ntable	keep table in memory rather than add it to document
noisily	show the putpdf commands used to export to the PDF file
dofile(<i>filename</i> [, replace])	save the putpdf commands used for exporting to the named do-file
<u>ta</u> blename(<i>tablename</i>)	specify a name for the table

Options

`name(cname)` specifies a collection *cname* from which to export the customized table. By default, the customized table is taken from the current collection.

`memtable` specifies that the table be created and held in memory instead of being added to the active document. By default, the table is added to the document. This option is useful if the table is intended to be added to a cell of another table or to be used multiple times later.

`noisily` specifies that putpdf collect show the putpdf commands used to export to the PDF file.

`dofile(filename[, replace])` specifies that putpdf collect save to *filename* the putpdf commands used to export to the PDF file. If *filename* already exists, it can be overwritten by specifying *replace*. If *filename* is specified without an extension, `.do` is assumed.

`tablename(tablename)` specifies a name for the table. By naming the table, you can make further edits with `putpdf table`. The name must be a valid name according to Stata's naming conventions; see [\[U\] 11.3 Naming conventions](#).

If the current collection contains multiple tables, the table names will contain the prefix *tablename* and an integer as the suffix. For example, if you specify `tablename(myreg)` and the collection contains three tables, the table names will be `myreg1`, `myreg2`, and `myreg3`. Also, note that a name is a sequence of 1 to 32 letters, digits, and underscores. If you are exporting multiple tables, consider using a shorter sequence to account for the integer suffix.

Remarks and examples

Remarks are presented under the following headings:

[Introduction](#)

[Export a table with items from a collection](#)

Introduction

`putpdf collect` exports a table from a collection to the active PDF file. A collection is a set of results that have been obtained from one or more Stata commands with the `collect` suite of commands, from the `table` command, or from the `etable` command.

To create a table with the `collect` commands, you first collect results from one or more Stata commands. Then you create a table by specifying a layout that determines which results are to be included in the table and how they are to be arranged. For example, you can have the rows correspond to variables and the columns correspond to the statistics, or vice versa. You can also modify the labels in the table, format the results, add borders, change the font style, and make other styling changes to the table. Once you have finalized your table, you can add it to your active PDF file with `putpdf collect`.

It is possible to have many collections in memory. However, there is only one current collection, the one you are working with. The current collection is the one that will be exported with `putpdf collect`, but you can export a table from another collection by specifying the `name()` option.

A table that is conveniently added to a PDF file using `putpdf collect` could equivalently be added using a series of `putpdf table` commands. To see what those commands look like, use the `noisily` option with `putpdf collect`. This long list will quickly fill up your Results window, so you may want to store those commands in a do-file instead by using the `dofile()` option with `putpdf collect`. For

reproducibility purposes, you may choose to include your `collect` and `putpdf collect` commands in your do-file, or you may instead incorporate the commands from the do-file created by the `dofile()` option.

While the `collect` commands provide many generic styling options that will be applied to the table you export to your PDF file, the suite also includes `collect style putpdf`, which provides styling options specifically for tables exported to the PDF format. You can use `collect style putpdf` to specify the alignment of the table on the page, specify the table indentation, and more.

In the next section, we will show you how to collect results from a Stata command, customize the table with those results, and export the table to a PDF file. In these examples, we assume some familiarity with the `table` and `collect` commands. We recommend that you see [\[R\] table intro](#) for information on `table` and [\[TABLES\] Intro](#) for more information on `collect` to learn more about creating and customizing tables before incorporating them into your PDF file.

Export a table with items from a collection

In this example, we will use data from the Second National Health and Nutrition Examination Survey (NHANES II) ([McDowell et al. 1981](#)). Suppose we want to model the occurrence of high blood pressure (`highbp`) and then create a table in our PDF file that lists the estimated probabilities for different groups of people. Below, we begin by loading the data and fitting the logistic regression model:

```
. use https://www.stata-press.com/data/r19/nhanes2, clear
. logistic highbp i.sex i.race i.agegrp

Logistic regression                                Number of obs = 10,351
                                                    LR chi2(8)      = 1599.71
                                                    Prob > chi2     = 0.0000
Log likelihood = -6250.912                        Pseudo R2      = 0.1134
```

highbp	Odds ratio	Std. err.	z	P> z	[95% conf. interval]	
sex						
Female	.6463459	.0279512	-10.09	0.000	.59382	.703518
race						
Black	1.673928	.1179769	7.31	0.000	1.457958	1.921891
Other	1.321562	.2075301	1.78	0.076	.971449	1.797857
agegrp						
30-39	1.957614	.1543556	8.52	0.000	1.677301	2.284775
40-49	3.316726	.2668563	14.90	0.000	2.832851	3.88325
50-59	6.117762	.4864457	22.78	0.000	5.234924	7.149485
60-69	7.239449	.4920962	29.12	0.000	6.336446	8.271139
70+	10.59802	.9377584	26.68	0.000	8.910598	12.605
_cons	.2316432	.013944	-24.30	0.000	.205864	.2606506

Note: `_cons` estimates baseline odds.

Then before collecting any results, we clear out all styles, including the default style. When you collect results from a Stata command, the collection will have a default style. For example, a border will be added between the row headers and the results. For this table, we would like to clear all those specifications and begin with an empty style. The next step is to use `margins` to obtain the expected probabilities of having high blood pressure. We prefix the `margins` command with `collect:` to collect the statistics that are computed.

```
. collect style clear
. collect: margins sex race agegrp
Predictive margins                                Number of obs = 10,351
Model VCE: OIM
Expression: Pr(highbp), predict()
```

	Delta-method					
	Margin	std. err.	z	P> z	[95% conf. interval]	
sex						
Male	.4708373	.0065793	71.56	0.000	.457942	.4837326
Female	.3794041	.0061263	61.93	0.000	.3673968	.3914114
race						
White	.4104461	.0047768	85.93	0.000	.4010838	.4198084
Black	.5189607	.0139876	37.10	0.000	.4915455	.546376
Other	.468975	.0328331	14.28	0.000	.4046234	.5333267
agegrp						
20-29	.1668675	.0076677	21.76	0.000	.151839	.1818959
30-39	.2798711	.0110569	25.31	0.000	.2581999	.3015423
40-49	.3949581	.0135814	29.08	0.000	.3683391	.4215771
50-59	.5436194	.0137415	39.56	0.000	.5166866	.5705522
60-69	.5842869	.0091338	63.97	0.000	.5663851	.6021888
70+	.6716042	.0148123	45.34	0.000	.6425726	.7006357

Now these results are stored in a collection, and we can arrange them in a few different ways with collect layout. Below, we specify that we want a single table, where the rows correspond to the variable names (colname) and the columns correspond to the statistics (result). While many statistics are collected from the margins command, we only want to include the probabilities and confidence intervals in our table. Each statistic is a level, or component, of the dimension result. And we can include levels of a dimension by specifying them within brackets next to the dimension name. `_r_b` represents the probabilities, and `_r_ci` represents the confidence intervals.

```
. collect layout (colname) (result[_r_b _r_ci])
Collection: default
Rows: colname
Columns: result[_r_b _r_ci]
Table 1: 11 x 2
```

			Result		Result
			Coefficient		95% CI
Covariate names and column names	Sex	Male	.4708373	.457942	.4837326
Covariate names and column names	Sex	Female	.3794041	.3673968	.3914114
Covariate names and column names	Race	White	.4104461	.4010838	.4198084
Covariate names and column names	Race	Black	.5189607	.4915455	.546376
Covariate names and column names	Race	Other	.468975	.4046234	.5333267
Covariate names and column names	Age group	20-29	.1668675	.151839	.1818959
Covariate names and column names	Age group	30-39	.2798711	.2581999	.3015423
Covariate names and column names	Age group	40-49	.3949581	.3683391	.4215771
Covariate names and column names	Age group	50-59	.5436194	.5166866	.5705522
Covariate names and column names	Age group	60-69	.5842869	.5663851	.6021888
Covariate names and column names	Age group	70+	.6716042	.6425726	.7006357

If we were to export the collection right now, the table above is what we would see in the PDF file. But collect has many tools for customizing this table, so let's clean up the headers and format the results before exporting.

Our table reports probabilities, not coefficients, so below we use `collect label levels` to modify the label for the level `_r_b` accordingly. Also, we have two dimensions in this table, `result` and `colname`. You may have noticed the repeating titles for both of these dimensions; we can remove them with `collect style header`:

```
. collect label levels result _r_b "Prob.", modify
. collect style header, title(hide)
```

With `collect style cell`, we can make changes to a single cell, multiple levels of a dimension, or all cells in a dimension. First, we use the `cidelimiter()` option to specify that we want to use a comma to separate the upper and lower bounds of each confidence interval. We also use the `sformat()` option to place square brackets around the confidence intervals. We only need to apply these options to the level `_r_ci` of the dimension `result`. Then, for all cells with numeric content, we specify that only three digits should be displayed after the decimal.

```
. collect style cell result[_r_ci], cidelimiter(", ") sformat("%s")
. collect style cell, nformat(%5.3f)
. collect layout
```

```
Collection: default
```

```
  Rows: colname
```

```
Columns: result[_r_b _r_ci]
```

```
Table 1: 11 x 2
```

		Prob.	95% CI
Sex	Male	0.471	[0.458, 0.484]
Sex	Female	0.379	[0.367, 0.391]
Race	White	0.410	[0.401, 0.420]
Race	Black	0.519	[0.492, 0.546]
Race	Other	0.469	[0.405, 0.533]
Age group	20–29	0.167	[0.152, 0.182]
Age group	30–39	0.280	[0.258, 0.302]
Age group	40–49	0.395	[0.368, 0.422]
Age group	50–59	0.544	[0.517, 0.571]
Age group	60–69	0.584	[0.566, 0.602]
Age group	70+	0.672	[0.643, 0.701]

When we type `collect layout` without any arguments, we get a report of the current layout. This collection looks nice, but there are just a few minor things we can do to make it complete. First, notice that the row headers `Sex`, `Race`, and `Age group` are repeated for all the levels of the factor variables. We can use `collect style row` to hide all but the first of the duplicate row headers. Then, we add some borders. `border_block` is another dimension of the table and it divides the table into four blocks: row header, column header, corner (top left area), and item (which contains the results).

```
. collect style row split, dups(first)
. collect style cell border_block[row-header], border(top) border(bottom)
. collect style cell border_block[column-header], border(top) border(bottom)
. collect style cell border_block[corner], border(top)
. collect style cell border_block[item], border(bottom)
```

If you find yourself modifying borders or making other style changes in the same way often enough, you do not need to type these commands every time you create a table. Instead, you can save the style specifications with `collect style save`. Then you can apply that style to your tables with `collect style use`.

We are almost ready to export our polished collection to a PDF file. The last thing we will do is set the width of the table to 80% of the default table width; we can make this change with `collect style putpdf`. Then we will create a document in memory and export the collection.

```
. collect style putpdf, width(80%)
. putpdf begin
. putpdf collect
(collection default posted to putpdf)
. putpdf save report, replace
successfully created "C:/mypath/report.pdf"
```

After saving our work, `report.pdf` contains the following:



Stored results

`putpdf collect` stores the following in `s()`:

```
Macros
      s(collection)      name of collection
      s(dofile)         name of the new do-file
```

References

Huber, C. 2021. Customizable tables in Stata 17, part 6: Tables for multiple regression models. *The Stata Blog: Not Elsewhere Classified*. <https://blog.stata.com/2021/09/02/customizable-tables-in-stata-17-part-6-tables-for-multiple-regression-models/>.

McDowell, A., A. Engel, J. T. Massey, and K. Maurer. 1981. "Plan and operation of the Second National Health and Nutrition Examination Survey, 1976–1980". In *Vital and Health Statistics*, ser. 1, no. 15. Hyattsville, MD: National Center for Health Statistics.

Also see

[RPT] **putpdf intro** — Introduction to generating PDF files

[RPT] **putpdf begin** — Create a PDF file

[RPT] **putpdf pagebreak** — Add breaks to a PDF file

[RPT] **putpdf paragraph** — Add text or images to a PDF file

[RPT] **putpdf table** — Add tables to a PDF file

[R] **etable** — Create a table of estimation results

[R] **table intro** — Introduction to tables of frequencies, summaries, and command results

[TABLES] **Intro** — Introduction

[TABLES] **collect style putpdf** — Collection styles for putpdf

Description

putpdf pagebreak adds a page break to the document, placing subsequent content on the next page of the document.

putpdf sectionbreak adds a new section to the active document that starts on the next page. It lets you vary the page size, orientation, margins, and other properties of the pages within a single document. This formatting of sections is most useful when you want to mix portrait and landscape layouts.

Quick start

Add a page break to the document

```
putpdf pagebreak
```

Begin a new section with a landscape layout in the document

```
putpdf sectionbreak, landscape
```

Same as above, and specify 1 inch margins on the left and right side of the page

```
putpdf sectionbreak, landscape margin(left,1) margin(right,1)
```

Syntax

Add page break to document

```
putpdf pagebreak
```

Add section break to document

```
putpdf sectionbreak [ , section_options ]
```

<i>section_options</i>	Description
<code>pagesize(<i>psize</i>)</code>	set page size of section
<code>landscape</code>	set section orientation to landscape
<code>font(<i>fspec</i>)</code>	set font, font size, and font color
<code>halign(<i>hvalue</i>)</code>	set horizontal alignment of section
<code>margin(<i>type</i>, #[<i>unit</i>])</code>	set page margins of section
<code>bgcolor(<i>color</i>)</code>	set background color

Options

`pagesize(psize)` sets the page size of the section. *psize* may be `letter`, `legal`, `A3`, `A4`, `A5`, `B4`, or `B5`. The default is `pagesize(letter)`.

`landscape` changes the section orientation from portrait (the default) to landscape.

`font(fontname [, size [, color]])` sets the font, font size, and font color for the document.

fontname may be any supported font installed on the user's computer. Base 14 fonts, Type 1 fonts (`.pfa` or `.pfb`), TrueType fonts (`.ttf` or `.ttc`), and OpenType fonts (`.otf`) are supported. TrueType and OpenType fonts that cannot be embedded may not be used. If *fontname* includes spaces, then it must be enclosed in double quotes. The default font is Helvetica.

size is a numeric value that represents font size measured in points. The default is 11.

color sets the text color. *color* may be one of the colors listed in [Colors of \[RPT\] Appendix for putpdf](#); a valid RGB value in the form `### ##`, for example, `171 248 103`; or a valid RRGGBB hex value in the form `#####`, for example, `ABF867`.

The font size and font color may be specified individually without specifying *fontname*. Use `font("", size)` to specify font size only. Use `font("", "", color)` to specify font color only. For both cases, the default font will be used.

`halign(hvalue)` sets the horizontal alignment of the paragraphs, images, and tables within the section. *hvalue* may be `left`, `right`, or `center`. The default is `halign(left)`.

`margin(type, #[unit])` sets the page margins of the section. This option may be specified multiple times in a single command to account for different margin settings.

type identifies the location of the margin inside the document. *type* may be `top`, `left`, `bottom`, `right`, or `all`.

unit may be `in` (inch), `pt` (point), `cm` (centimeter), or `twip` (twentieth of a point). An inch is equivalent to 72 points, 2.54 centimeters, or 1440 twips. The default is `in`.

`bghcolor(color)` sets the background color for the document. *color* may be one of the colors listed in [Colors of \[RPT\] Appendix for putpdf](#); a valid RGB value in the form `### ##`, for example, `171 248 103`; or a valid RRGGBB hex value in the form `#####`, for example, `ABF867`.

Remarks and examples

The `putpdf pagebreak` and `putpdf sectionbreak` commands are useful for organizing your PDF document. Whether you wish to insert page breaks for each new section in your file or you want to format different segments of your document differently, you can do this with these two commands.

To begin all subsequently added content on the next page in the active document, use the `putpdf pagebreak` command. This may be useful, for instance, when you do not want tables or paragraphs split across pages.

You can vary the page layout for sections of your document by using `putpdf sectionbreak`. When inserting a section break, the content that follows will be placed on the following page and formatted according to the options you specify with `putpdf sectionbreak`. You can modify the page size, orientation, font, and other features for each section. This is particularly useful if you have content in your

document that varies in dimensions because you can specify a different layout for each element. For example, you may want a portrait layout for a long estimation table and a landscape layout for a wide image.

► Example 1: Add section break to PDF file

Suppose we have created a PDF file with the default portrait layout and added some content.

```
putpdf begin
putpdf paragraph
...
```

We now wish to add a wide image that does not fit properly in the confines of the current layout. We can begin a new section with a `landscape` layout and specify quarter-inch margins on the left and right side of the page.

```
putpdf sectionbreak, landscape margin(left,0.25) margin(right,0.25)
```

We can now export the image created in [example 2](#) of [\[RPT\] putpdf paragraph](#) with a much larger width.

```
putpdf paragraph
putpdf image scatter.png, width(11) height(7)
```

To continue adding content on the following page, retaining the `landscape` layout and small margins, we type

```
putpdf pagebreak
```

Inserting a page break simply places the following content on the next page. Any formatting options specified with `putpdf sectionbreak` will still be applied to the following pages.

To conclude this section and resume our document with a portrait layout and the default margin size, we type

```
putpdf sectionbreak
```



Also see

[\[RPT\] putpdf intro](#) — Introduction to generating PDF files

[\[RPT\] putpdf begin](#) — Create a PDF file

[\[RPT\] putpdf collect](#) — Add a table from a collection to a PDF file

[\[RPT\] putpdf paragraph](#) — Add text or images to a PDF file

[\[RPT\] putpdf table](#) — Add tables to a PDF file

[\[RPT\] Appendix for putpdf](#) — Appendix for putpdf entries

[Description](#)[Remarks and examples](#)[Quick start](#)[Also see](#)[Syntax](#)[Options](#)

Description

`putpdf paragraph` adds a new paragraph to the active document. The newly created paragraph becomes the active paragraph. All subsequent text or images will be appended to the active paragraph.

`putpdf text` adds content to the paragraph created by `putpdf paragraph`. The text may be a plain text string or any valid Stata expression (see [\[U\] 13 Functions and expressions](#)).

`putpdf image` embeds a portable network graphics (.png) or JPEG (.jpg) file in the paragraph.

Quick start

Add paragraph to the document

```
putpdf paragraph
```

Same as above, but with a half-inch standard paragraph indentation

```
putpdf paragraph, indent(para,0.5)
```

Append the text “This is paragraph text” to the active paragraph and format the text as bold

```
putpdf text ("This is paragraph text"), bold
```

Add a PNG image saved as `image1` to the paragraph above, with a height and width of 4 inches

```
putpdf image image1.png, height(4) width(4)
```

Add a PNG image saved as `image2` and center the image

```
putpdf paragraph, halign(center)
```

```
putpdf image image2.png
```

Syntax

Add paragraph to document

```
putpdf paragraph [ , paragraph_options ]
```

Add text to paragraph

```
putpdf text (exp) [ , text_options ]
```

Add image to paragraph

```
putpdf image filename [ , image_options ]
```

filename is the full path to the image file or the relative path from the current working directory.

<i>paragraph_options</i>	Description
<code>font(<i>fspec</i>)</code>	set font, font size, and font color
<code>halign(<i>hvalue</i>)</code>	set paragraph alignment
<code>valign(<i>vvalue</i>)</code>	set vertical alignment of characters on each line
<code>indent(<i>indenttype</i>, #[<i>unit</i>])</code>	set paragraph indentation
<code>spacing(<i>position</i>, #[<i>unit</i>])</code>	set spacing between lines of text
<code>bgcolor(<i>color</i>)</code>	set background color

<i>text_options</i>	Description
<code>nformat(<i>%fmt</i>)</code>	specify numeric format for text
<code>font(<i>fspec</i>)</code>	set font, font size, and font color
<code>bold</code>	format text as bold
<code>italic</code>	format text as italic
<code>script(sub super)</code>	set subscript or superscript formatting of text
<code><u>strikeout</u></code>	strike out text
<code><u>underline</u></code>	underline text
<code>bgcolor(<i>color</i>)</code>	set background color
<code>linebreak[(#)]</code>	add line breaks after text
<code><u>allcaps</u></code>	format text as all caps

<i>image_options</i>	Description
<code>width(#[<i>unit</i>])</code>	set image width
<code>height(#[<i>unit</i>])</code>	set image height
<code>linebreak[(#)]</code>	add line breaks after image

fspec is

fontname [, *size* [, *color*]]

fontname may be any supported font installed on the user's computer. Base 14 fonts, Type 1 fonts (.pfa or .pfb), TrueType fonts (.ttf or .ttc), and OpenType fonts (.otf) are supported. TrueType and OpenType fonts that cannot be embedded may not be used. If *fontname* includes spaces, then it must be enclosed in double quotes. The default font is Helvetica.

size is a numeric value that represents font size measured in points. The default is 11.

color sets the text color.

unit may be in (inch), pt (point), cm (centimeter), or twip (twentieth of a point). An inch is equivalent to 72 points, 2.54 centimeters, or 1440 twips. The default is in.

color may be one of the colors listed in [Colors](#) of [\[RPT\] Appendix for putpdf](#); a valid RGB value in the form ### ##, for example, 171 248 103; or a valid RRGGBB hex value in the form #####, for example, ABF867.

Options

Options are presented under the following headings:

[Options for putpdf paragraph](#)

[Options for putpdf text](#)

[Options for putpdf image](#)

Options for putpdf paragraph

`font(fontname [, size [, color]])` sets the font, font size, and font color for the text within the paragraph. The font size and font color may be specified individually without specifying *fontname*. Use `font("", size)` to specify font size only. Use `font("", "", color)` to specify font color only. For both cases, the default font will be used.

Specifying `font()` with `putpdf paragraph` overrides font settings specified with `putpdf begin`.

`halign(hvalue)` sets the horizontal alignment of the text within the paragraph. *hvalue* may be left, right, center, justified, or distribute. distribute and justified justify text between the left and right margins equally, but distribute also changes the spacing between words and characters. The default is `halign(left)`.

`valign(vvalue)` sets the vertical alignment of the characters on each line when the paragraph contains characters of varying size. *vvalue* may be baseline, bottom, center, or top. The default is `valign(baseline)`.

`indent(indenttype, #[unit])` specifies that the paragraph be indented by # *units*. *indenttype* may be left, right, or para. left and right indent # *units* from the left or the right, respectively. para uses standard paragraph indentation and indents the first line by # inches unless another *unit* is specified. This option may be specified multiple times in a single command to accommodate different indentation settings.

`spacing(position, #[unit])` sets the spacing between lines of text. *position* may be `before`, `after`, or `line`. `before` specifies the space before the first line of the current paragraph, `after` specifies the space after the last line of the current paragraph, and `line` specifies the space between lines within the current paragraph. This option may be specified multiple times in a single command to accommodate different spacing settings.

`bgcolor(color)` sets the background color for the paragraph.

Specifying `bgcolor()` with `putpdf paragraph` overrides background color specifications from `putpdf begin`.

Options for putpdf text

`nformat(%fmt)` specifies the numeric format of the text when the content of the new text appended to the paragraph is a numeric value. This setting has no effect when the content is a string.

`font(fontname [, size [, color]])` sets the font, font size, and font color for the new text within the active paragraph. The font size and font color may be specified individually without specifying *fontname*. Use `font("", size)` to specify font size only. Use `font("", "", color)` to specify font color only. For both cases, the default font will be used.

Specifying `font()` with `putpdf text` overrides all other font settings, including those specified with `putpdf begin` and `putpdf paragraph`.

`bold` specifies that the new text in the active paragraph be formatted as bold.

`italic` specifies that the new text in the active paragraph be formatted as italic.

`script(sub | super)` changes the script style of the new text. `script(sub)` makes the text a subscript. `script(super)` makes the text a superscript.

`strikeout` specifies that the new text in the active paragraph have a strikeout mark.

`underline` specifies that the new text in the active paragraph be underlined.

`bgcolor(color)` sets the background color for the active paragraph.

Specifying `bgcolor()` with `putpdf text` overrides background color specifications from `putpdf begin` and `putpdf paragraph`.

`linebreak[(#)]` specifies that one or *#* line breaks be added after the new text.

`allcaps` specifies that all letters of the new text in the active paragraph be capitalized.

Options for putpdf image

`width(#[unit])` sets the width of the image. If the width is larger than the body width of the document, then the body width is used. If `width()` is not specified, then the default size is used; the default is determined by the image information and the body width of the document.

`height(#[unit])` sets the height of the image. If `height()` is not specified, then the height of the image is determined by the width and the aspect ratio of the image.

`linebreak[(#)]` specifies that one or *#* line breaks be added after the new image.

Remarks and examples

Once you have created a PDF file in memory with `putpdf begin`, you can add text and images by appending them to an active paragraph. You can control the formatting for the whole paragraph, such as font properties and alignment, with options for `putpdf paragraph`. You can further customize each addition of text with the options available with `putpdf text`. Text can include any valid Stata expression, such as a string, algebraic expression, or a direct reference to a stored result.

Remarks are presented under the following headings:

Adding a paragraph

Adding an image to the document

Adding a paragraph

Before we can add text or an image to a document, we must first begin a new paragraph by using `putpdf paragraph`. The current paragraph remains active until you add a new paragraph, a table, a section break, or a page break.

► Example 1: Add a paragraph with text

Suppose we want to write a description of `auto.dta` to `example.pdf`. We first open the dataset:

```
. use https://www.stata-press.com/data/r19/auto
(1978 automobile data)
```

We begin by adding a paragraph for the title of our report:

```
. putpdf begin
. putpdf paragraph, halign(center) font(,16)
. putpdf text ("Auto report"), bold underline
```

By specifying a font size without the actual font, we have simply incremented the size of the default font in our document. We could similarly change just the color of the font without specifying anything else.

We use `summarize` to compute summary statistics on the price of cars. We can use the [stored results](#) in the text that we write.

```
. summarize price, detail
```

Price			
Percentiles		Smallest	
1%	3291	3291	
5%	3748	3299	
10%	3895	3667	Obs 74
25%	4195	3748	Sum of wgt. 74
50%	5006.5		Mean 6165.257
		Largest	Std. dev. 2949.496
75%	6342	13466	
90%	11385	13594	Variance 8699526
95%	13466	14500	Skewness 1.653434
99%	15906	15906	Kurtosis 4.819188


```
. return list
scalars:
      r(N) = 74
    r(sum_w) = 74
    r(mean) = 6165.256756756757
    r(Var) = 8699525.97426879
    r(sd) = 2949.495884768919
  r(skewness) = 1.653433511704859
  r(kurtosis) = 4.819187528464004
    r(sum) = 456229
    r(min) = 3291
    r(max) = 15906
    r(p1) = 3291
    r(p5) = 3748
    r(p10) = 3895
    r(p25) = 4195
    r(p50) = 5006.5
    r(p75) = 6342
    r(p90) = 11385
    r(p95) = 13466
    r(p99) = 15906
```

With the scalar names in hand, we can begin describing the prices of cars in our data. However, we first create a new paragraph because we do not want the following text to be center-aligned.

```
. putpdf paragraph, indent(para,0.5)
. putpdf text ("The average car price in this dataset is $")
. putpdf text ("`r(mean)'"', nformat("%5.2f")
. putpdf text (" . The 25th percentile is `$r(p25)', and the 75th percentile
> is `$r(p75)'".")
```

We request the standard paragraph indentation using the `para` option and specify a half-inch indentation. We also format the average car price to two decimal places.

◀

Adding an image to the document

Like text, images are appended to active paragraphs, and thus their alignment within the document is controlled by the options specified with `putpdf paragraph`. Images must be stored in one of the supported formats: portable network graphics (`.png`) or JPEG (`.jpg`) file.

▷ Example 2: Export an image to a .pdf file

Now that we know some basic information on car pricing, we can see how the price of cars correlates with their *weight* and *mileage*. We visualize this correlation with a scatterplot matrix, using `graph matrix`:

```
graph matrix weight mpg price, half
graph export scatter.png
```

Because Stata graphs use the `.gph` extension, we use `graph export` to save the scatterplot matrix as `scatter.png`. Now we can append it to a paragraph and specify the dimensions of the image.

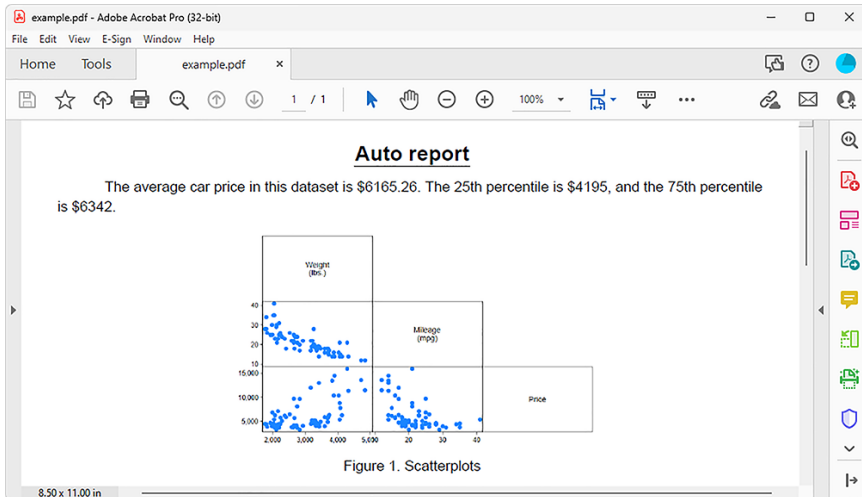
```
putpdf paragraph, halign(center)
putpdf image scatter.png, width(4) linebreak
putpdf text ("Figure 1. Scatterplots")
```

We center the image by specifying the horizontal alignment of the paragraph we are appending it to. To place a caption below the image, rather than beside it, we add a linebreak.

Now that we have added our descriptive statistics and a related graph, we can save and close our document:

```
putpdf save example.pdf
```

The document appears as follows:



◀

Also see

[RPT] [putpdf intro](#) — Introduction to generating PDF files

[RPT] [putpdf begin](#) — Create a PDF file

[RPT] [putpdf collect](#) — Add a table from a collection to a PDF file

[RPT] [putpdf pagebreak](#) — Add breaks to a PDF file

[RPT] [putpdf table](#) — Add tables to a PDF file

[RPT] [Appendix for putpdf](#) — Appendix for putpdf entries

Description

`putpdf table` creates a new table in the active PDF file. Tables may be created from several output types, including the data in memory, matrices, and estimation results.

`putpdf describe` describes a table within the active PDF file.

`set pdf_maxtable` allows you to set the maximum number of tables allowed in `putpdf`.

Quick start

Add a table named `tbl1` with three rows and four columns to the document

```
putpdf table tbl1 = (3,4)
```

Same as above, but with the title “Table 1”

```
putpdf table tbl1 = (3,4), title("Table 1")
```

Insert the image in `myimg.png` in the second row and second column of table `tbl1`

```
putpdf table tbl1(2,2) = image(myimg.png)
```

Set the content of the cell on the second row and first column to be “Image 1” and center the text

```
putpdf table tbl1(2,1) = ("Image 1"), halign(center)
```

Add a table named `tbl2` with regression results after `regress`

```
putpdf table tbl2 = etable
```

Format the content on all rows of columns two through four of `tbl2` to have two decimal places

```
putpdf table tbl2(.,2/4), nformat(%5.2f)
```

Add a table with the contents matrix `mymat`, including the row names and column names of the matrix

```
putpdf table tbl3= matrix(mymat), rownames colnames
```

Syntax

Add table to document

```
putpdf table tablename = (nrows, ncols) [ , table_options ]
putpdf table tablename = data(varlist) [ if ] [ in ] [ , varnames obsno
table_options ]
putpdf table tablename = matrix(matname) [ , nformat(%fmt) rownames colnames
table_options ]
putpdf table tablename = mata(matname) [ , nformat(%fmt) table_options ]
putpdf table tablename = etable[ (#1 #2 ... #n) ] [ , table_options ]
putpdf table tablename = returnset [ , table_options ]
```

Add content to cell

```
putpdf table tablename(i, j) = (exp) [ , cell_options ]
putpdf table tablename(i, j) = image(filename) [ , cell_options ]
putpdf table tablename(i, j) = table(mem_tablename) [ , cell_options ]
```

Alter table layout

```
putpdf table tablename(i, .), row_col_options
putpdf table tablename(., j), row_col_options
```

Customize format of cells or table

```
putpdf table tablename(i, j), cell_options
putpdf table tablename(numlisti, .), cell_fmt_options
putpdf table tablename(., numlistj), cell_fmt_options
putpdf table tablename(numlisti, numlistj), cell_fmt_options
putpdf table tablename(., .), cell_fmt_options
```

Describe table

```
putpdf describe tablename
```

Set the maximum number of tables

```
set pdf_maxtable # [ , permanently ]
```

is any number between 1 and 10,000; the default is 500.

tablename specifies the name of a new table. The name must be a valid name according to Stata’s naming conventions; see [U] 11.3 Naming conventions.

<i>table_options</i>	Description
<u>mem</u> table	keep table in memory rather than add it to document
width(<i>#</i> [<i>unit</i> %] <i>matname</i>)	set table width
halign(<i>hvalue</i>)	set table horizontal alignment
indent(<i>#</i> [<i>unit</i>])	set table indentation
spacing(<i>position</i> , <i>#</i> [<i>unit</i>])	set spacing before or after table
<u>border</u> (<i>bspec</i>)	set pattern and color for border
title(<i>string</i> [, <i>cell_fmt_options</i>])	add a title to the table
note(<i>string</i> [, <i>cell_fmt_options</i>])	add a note to the table

<i>cell_options</i>	Description
append	append objects to current content of cell
rowspan(<i>#</i>)	merge cells vertically
colspan(<i>#</i>)	merge cells horizontally
span(<i>#</i> ₁ , <i>#</i> ₂)	merge cells both horizontally and vertically
linebreak[(<i>#</i>)]	add line breaks into the cell
<i>cell_fmt_options</i>	options that control the look of cell contents

<i>row_col_options</i>	Description
<u>nosplit</u>	prevent row from breaking across pages
addrows(<i>#</i> [, before after])	add # rows in specified location
addcols(<i>#</i> [, before after])	add # columns in specified location
drop	drop specified row or column
<i>cell_fmt_options</i>	options that control the look of cell contents

<i>cell_fmt_options</i>	Description
margin(<i>type</i> , <i>#</i> [<i>unit</i>])	set margins
halign(<i>hvalue</i>)	set horizontal alignment
valign(<i>vvalue</i>)	set vertical alignment
<u>border</u> (<i>bspec</i>)	set pattern and color for border
bgcolor(<i>color</i>)	set background color
<u>nformat</u> (<i>%fmt</i>)	specify numeric format for cell text
font(<i>fspec</i>)	set font, font size, and font color
bold	format text as bold
italic	format text as italic
* <u>script</u> (sub super)	set subscript or superscript formatting of text
<u>strikeout</u>	strikeout text
<u>underline</u>	underline text
<u>allcaps</u>	format text as all caps

* May only be specified when formatting a single cell.

fspec is

fontname [, *size* [, *color*]]

fontname may be any supported font installed on the user's computer. Base 14 fonts, Type 1 fonts (.pfa or .pfb), TrueType fonts (.ttf or .ttc), and OpenType fonts (.otf) are supported. TrueType and OpenType fonts that cannot be embedded may not be used. If *fontname* includes spaces, then it must be enclosed in double quotes. The default font is Helvetica.

size is a numeric value that represents font size measured in points. The default is 11.

color sets the text color.

bspec is

bordername [, *bpattern* [, *bcolor*]]

bordername specifies the location of the border.

bpattern is a keyword specifying the look of the border. Possible patterns are `nil` and `single`. The default is `single`. To remove an existing border, specify `nil` as the *bpattern*.

bcolor specifies the border color.

unit may be `in` (inch), `pt` (point), `cm` (centimeter), or `twip` (twentieth of a point). An inch is equivalent to 72 points, 2.54 centimeters, or 1440 twips. The default is `in`.

color and *bcolor* may be one of the colors listed in [Colors](#) of [\[RPT\] Appendix for putpdf](#); a valid RGB value in the form `#### #### ####`, for example, 171 248 103; or a valid RRGGBB hex value in the form `#####`, for example, ABF867.

Output types for tables

The following output types are supported when creating a new table using `putpdf table tablename`:

`(nrows, ncols)` creates an empty table with *nrows* rows and *ncols* columns. A maximum of 50 columns in a table is allowed.

`data(varlist)` adds the current Stata dataset in memory as a table to the active document. *varlist* contains a list of the variable names from the current dataset in memory.

`matrix(matname)` adds a [matrix](#) called *matname* as a table to the active document.

`mata(matname)` adds a Mata [matrix](#) called *matname* as a table to the active document.

`etable[(#1 #2 ... #n)]` adds an automatically generated table to the active document. The table may be derived from the coefficient table of the last estimation command, from the table of margins after the last [margins](#) command, or from the table of results from one or more models displayed by [estimates table](#).

If the estimation command outputs $n > 1$ coefficient tables, the default is to add all tables and assign the corresponding table names *tablename1*, *tablename2*, ..., *tablename_n*. To specify which tables to add, supply the optional numlist to `etable`. For example, to add only the first and third tables from the estimation output, specify `etable(1 3)`. A few estimation commands do not support the `etable` output type. See [Unsupported estimation commands](#) of [\[RPT\] Appendix for putpdf](#) for a list of estimation commands that are not supported by `putpdf`.

returnset exports a group of Stata [return](#) values to a table in the active document. It is intended primarily for use by programmers and by those who want to do further processing of their exported results in the active document. *returnset* may be one of the following:

<i>returnset</i>	Description
<i>escalars</i>	all ereturned scalars
<i>rscalars</i>	all returned scalars
<i>emacros</i>	all ereturned macros
<i>rmacros</i>	all returned macros
<i>ematrices</i>	all ereturned matrices
<i>rmatrices</i>	all returned matrices
<i>e*</i>	all ereturned scalars, macros, and matrices
<i>r*</i>	all returned scalars, macros, and matrices

The following output types are supported when adding content to an existing table using `putpdf table tablename(i, j)`:

`(exp)` writes a valid Stata expression to a cell; see [\[U\] 13 Functions and expressions](#).

`image filename` adds a portable network graphics (.png) or JPEG (.jpg) file to the table cell. *filename* is the path to the image file. It may be either the full path or the relative path from the current working directory.

`table(mem_tablename)` adds a previously created table, identified by *mem_tablename*, to the table cell.

The following combinations of `tablename(numlisti, numlistj)` (see [\[U\] 11.1.8 numlist](#) for valid specifications) can be used to format a cell or range of cells in an existing table:

`tablename(i, j)` specifies the cell on the *i*th row and *j*th column.

`tablename(i, .)` and `tablename(numlisti, .)` specify all cells on the *i*th row or on the rows identified by *numlist_{*i*}*.

`tablename(. , j)` and `tablename(. , numlistj)` specify all cells in the *j*th column or in the columns identified by *numlist_{*j*}*.

`tablename(. , .)` specifies the whole table.

Options

Options are presented under the following headings:

`table_options`
`cell_options`
`row_col_options`
`cell_fmt_options`
`Option for set pdf_maxtable`

table_options

`memtable` specifies that the table be created and held in memory instead of being added to the active document. By default, the table is added to the document immediately after it is created. This option is useful if the table is intended to be added to a cell of another table or to be used multiple times later.

`width(#[unit | %] | matname)` sets the table width. `width(#)`, `width(#unit)`, or `width(%)` may be specified with `width(matname)`.

`width(#[unit | %])` sets the width based on a specified value. `#` may be an absolute width or a percentage of the default table width, which is determined by the page width of the document. For example, `width(50%)` sets the table width to 50% of the default table width. The default is `width(100%)`.

When specifying the table width as a percentage, it cannot be greater than 100%.

`width(matname)` sets the table width based on the dimensions specified in the Stata matrix *matname*, which has contents in the form of $(\#_1, \#_2, \dots, \#_n)$ to denote the percentage of the default table width for each column. *n* is the number of columns of the table, and the sum of $\#_1$ to $\#_n$ must be equal to 100.

`halign(hvalue)` sets the horizontal alignment of the table within the page. *hvalue* may be `left`, `right`, or `center`. The default is `halign(left)`.

`indent(#[unit])` specifies the table indentation from the left margin of the current document.

`spacing(position, #[unit])` sets the spacing before or after the table. *position* may be `before` or `after`. `before` specifies the space before the top of the current table, and `after` specifies the space after the bottom of the current table. This option may be specified multiple times in a single command to account for different space settings.

`border(bordername [, bpattern [, bcolor]])` adds a single border in the location specified by *bordername*, which may be `start`, `end`, `top`, `bottom`, `insideH` (inside horizontal borders), `insideV` (inside vertical borders), or `all`. Optionally, you may change the pattern and color for the border by specifying *bpattern* and *bcolor*.

This option may be specified multiple times in a single command to accommodate different border settings. If multiple `border()` options are specified, they are applied in the order specified from left to right.

`varnames` specifies that the variable names be included as the first row in the table when the table is created using the dataset in memory. By default, only the data values are added to the table.

`obsno` specifies that the observation numbers be included as the first column in the table when the table is created using the dataset in memory. By default, only the data values are added to the table.

`nformat(%fmt)` specifies the numeric format to be applied to the source values when creating the table from a Stata or Mata matrix. The default is `nformat(%12.0g)`.

`rownames` specifies that the row names of the Stata matrix be included as the first column in the table. By default, only the matrix values are added to the table.

`colnames` specifies that the column names of the Stata matrix be included as the first row in the table. By default, only the matrix values are added to the table.

`title(string [, cell_fmt_options])` inserts a row without borders above the current table. The added row spans all the columns of the table and contains *string* as text. The added row shifts all other table contents down by one row. You should account for this when referencing table cells in subsequent commands.

This option may be specified multiple times in a single command to add titles on new lines within the same cell. Title text is inserted in the order it was specified from left to right. Each title can be customized using *cell_fmt_options*.

`note(string[, cell_fmt_options])` inserts a row without borders to the bottom of the table. The added row spans all the columns of the table. This option may be specified multiple times in a single command to add notes on new lines within the same cell. Note that text is inserted in the order it was specified from left to right. Each note can be customized using *cell_fmt_options*.

cell_options

`append` specifies that the new content for the cell be appended to the current content of the cell. If `append` is not specified, then the current content of the cell is replaced by the new content. Unlike with the `putdocx` command, this option with `putpdf` is used only for appending a new string to the cell when the original cell content is also a string.

`rowspan(#)` sets the specified cell to span vertically `#` cells downward. If the span exceeds the total number of rows in the table, the span stops at the last row.

`colspan(#)` sets the specified cell to span horizontally `#` cells to the right. If the span exceeds the total number of columns in the table, the span stops at the last column.

`span(#1, #2)` sets the specified cell to span `#1` cells downward and span `#2` cells to the right.

`linebreak[(#)]` specifies that one or `#` line breaks be added after the text within the cell.

row_col_options

`nosplit` specifies that row *i* not split across pages. When a table row is displayed, a page break may fall within the contents of a cell on the row, causing the contents of that cell to be displayed across two pages. `nosplit` prevents this behavior. If the entire row cannot fit on the current page, the row will be moved to the start of the next page.

`addrows([#, before|after])` adds `#` rows to the current table before or after row *i*. If `before` is specified, the rows are added before the specified row. By default, rows are added after the specified row.

`addcols([#, before|after])` adds `#` columns to the current table to the right or the left of column *j*. If `before` is specified, the columns are added to the left of the specified column. By default, the columns are added after, or to the right of, the specified column.

`drop` deletes row *i* or column *j* from the table.

cell_fmt_options

`margin(type, #[unit])` sets the margins inside the specified cell or of all cells in the specified row, column, or range. *type* may be `top`, `left`, `bottom`, or `right`, which identify the top margin, left margin, bottom margin, or right margin of the cell, respectively. This option may be specified multiple times in a single command to account for different margin settings.

`halign(hvalue)` sets the horizontal alignment of the specified cell or of all cells in the specified row, column, or range. *hvalue* may be `left`, `right`, or `center`. The default is `halign(left)`.

`valign(vvalue)` sets the vertical alignment of the specified cell or of all cells in the specified row, column, or range. *vvalue* may be `top`, `bottom`, or `center`. The default is `valign(top)`.

`border(bordername [, bpattern [, bcolor]])` adds a single border to the specified cell or to all cells in the specified row, column, or range in the given location. *bordername* may be `start`, `end`, `top`, `bottom`, or `all`. Optionally, you may change the pattern and color for the border by specifying *bpattern* and *bcolor*.

This option may be specified multiple times in a single command to accommodate different border settings. If multiple `border()` options are specified, they are applied in the order specified from left to right.

`bgcolor(color)` sets the background color for the specified cell or for all cells in the specified row, column, or range.

`nformat(%fmt)` applies the Stata numeric format *%fmt* to the text within the specified cell or within all cells in the specified row, column, or range. This setting only applies when the content of the cell is a numeric value.

`font(fontname [, size [, color]])` sets the font, font size, and font color for the text within the specified cell or within all cells in the specified row, column, or range. The font size and font color may be specified individually without specifying *fontname*. Use `font("", size)` to specify font size only. Use `font("", "", color)` to specify font color only. For both cases, the default font will be used.

`bold` applies bold formatting to the text within the specified cell or within all cells in the specified row, column, or range.

`italic` applies italic formatting to the text within the specified cell or within all cells in the specified row, column, or range.

`script(sub|super)` changes the script style of the text. `script(sub)` makes the text a subscript. `script(super)` makes the text a superscript. `script()` may only be specified when formatting a single cell.

`strikeout` adds a strikeout mark to the current text within the specified cell or within all cells in the specified row, column, or range.

`underline` adds an underline to the current text within the specified cell or within all cells in the specified row, column, or range.

`allcaps` uses capital letters for all letters of the current text within the specified cell or within all cells in the specified row, column, or range.

Option for `set pdf_maxtable`

`permanently` specifies that, in addition to making the change right now, the settings be remembered and become the default settings when you invoke Stata.

Remarks and examples

The suite of `putpdf` commands makes it easy to export summary statistics, estimation results, data, and images in neatly formatted tables. There are different output types available to export a whole coefficient table, matrix, or dataset in a single step. Alternatively, you can create a table by specifying the dimensions and then gradually inserting contents, such as text, images, and stored results. When you create a table you specify a name for it, which allows you to make further modifications to its contents. You can customize each cell, or apply a specific formatting to a range of cells with row and column indexes.

Remaining remarks are presented under the following headings:

- Creating basic tables*
- Exporting summary statistics*
- Exporting estimation results*
- Creating advanced tables*

Creating basic tables

When exporting only a few statistics and results, you can create a table from scratch—first specifying the size of the table and then adding content one cell at a time. With this method, it is easy to control the location of each element within the table.

► Example 1: Export a basic table

Suppose we want to export a table with just the mean, minimum, and maximum miles per gallon for 1978 automobiles to a .pdf file. First, we load the dataset and create a document in memory.

```
. sysuse auto, clear
. putpdf begin
```

Now we create our table with the necessary dimensions. We are exporting three statistics that will appear in one column. We need to allot another column to label these statistics. Therefore, we create a table named `mpgstats` with three rows and two columns, and we add a title. By default, the table width is set at 100%, but because our content is rather short, we set the width to 40% of the default. Otherwise, the table would take up a large portion of our document and look very empty.

```
. putpdf table mpgstats = (3,2), title(MPG summary statistics) width(40%)
. putpdf describe mpgstats
```

Table name	mpgstats
No. of rows	4
No. of cols	2

When we describe our table, it reports four rows instead of the three we specified. When we add a title, it is included as the first row of the table without any borders. With our table in place, we now run `summarize mpg` and view the list of stored results:

```
. summarize mpg
```

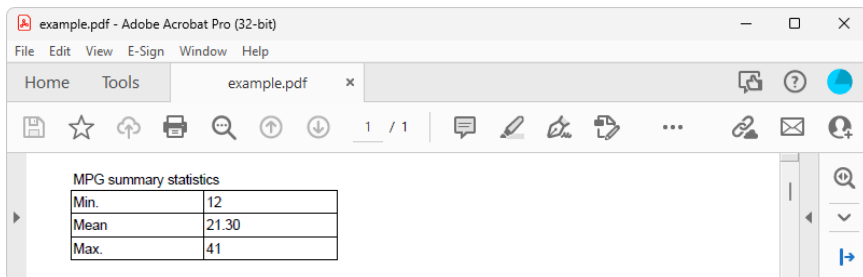
Variable	Obs	Mean	Std. dev.	Min	Max
mpg	74	21.2973	5.785503	12	41

```
. return list
scalars:
      r(N) = 74
    r(sum_w) = 74
    r(mean) = 21.2972972972973
    r(Var) = 33.47204738985561
    r(sd) = 5.785503209735141
    r(min) = 12
    r(max) = 41
    r(sum) = 1576
```

All the statistics we want are stored in `rclass` scalars. We can directly refer to stored results with `putpdf table`. We begin filling in the table by putting the label for the minimum in the second row and first column, `mpgstats(2,1)`. Then we add the value of the minimum in the cell next to it, `mpgstats(2,2)`, by referring to the `r(min)` scalar shown above. Similarly, we add the mean on row 3 and the maximum on row 4.

```
. putpdf table mpgstats(2,1) = ("Min.")
. putpdf table mpgstats(2,2) = (r(min))
. putpdf table mpgstats(3,1) = ("Mean")
. putpdf table mpgstats(3,2) = (r(mean)), nformat(%5.2f)
. putpdf table mpgstats(4,1) = ("Max.")
. putpdf table mpgstats(4,2) = (r(max))
. putpdf save example.pdf
successfully created "C:/mypath/example.pdf"
```

We format the mean of `mpg` to report only two digits after the decimal. After saving our work, `example.pdf` contains the following:



Exporting summary statistics

In the example above, we exported just a few summary statistics by filling in the content of each cell in a table. That method would be tedious if we wanted to export several results. Another option is to create a dataset of summary statistics and export the entire dataset to a table, as shown in the example below.

➤ Example 2: Export a dataset of summary statistics

Suppose that now we want to report the summary statistics separately for foreign and domestic automobiles. We create a new active document, and then we use the `statsby` command to collect the number of observations along with the minimum, mean, and maximum of `mpg` for each group. Because `statsby` creates a new dataset that overwrites the dataset in memory, we need to preserve the dataset and then restore it after we have finished exporting the data.

```

. putpdf begin
. preserve
. statsby Obs=r(N) Min=r(min) Mean=r(mean) Max=r(max), by(foreign):
> summarize mpg
(running summarize on estimation sample)

    Command: summarize mpg
      Obs: r(N)
      Min: r(min)
      Mean: r(mean)
      Max: r(max)
      By: foreign

Statsby groups:
..

```

We want the variable names to serve as column titles, so we rename `foreign` to `Origin`. Then, we export the data in memory as a table by specifying in `data()` the variable names `Origin`, `Obs`, `Min`, `Mean`, and `Max`. In this case, we use the table name `tbl1`. The order of the variable names in the list determines the column order in the table.

```

. rename foreign Origin
. putpdf table tbl1 = data("Origin Obs Min Mean Max"), varnames
> border(start, nil) border(insideV, nil) border(end, nil)

```

By default, the exported table includes single borders around all cells. We use the `border()` option to remove all vertical borders from the table.

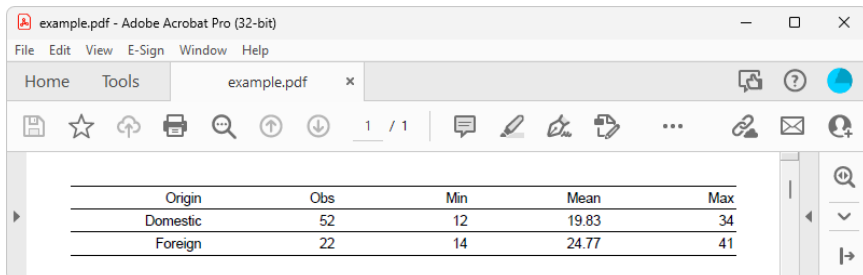
We further customize the table by setting the text for all cells of the table to be right-aligned instead of the default left-alignment. Because we would like all cells in the table to be right-aligned, we specify “.” for both the row and column specification.

```

. putpdf table tbl1(.,.), halign(right)
. putpdf table tbl1(2/3,4), nformat(%5.2f)
. putpdf save example, replace
successfully created "C:/mypath/example.pdf"

```

We save our work, replacing the existing `example.pdf` file with the contents of `tbl1`:



Origin	Obs	Min	Mean	Max
Domestic	52	12	19.83	34
Foreign	22	14	24.77	41

Afterward, we restore the dataset.

```

. restore

```

Exporting estimation results

One of the primary uses of `putpdf table` is to export estimation results. Suppose we fit a linear regression model of `mpg` as a function of the car's gear ratio (`gear_ratio`), turning radius (`turn`), and whether the car is of foreign origin (`foreign`) using `regress`.

```
. regress mpg gear_ratio turn foreign, noheader
```

mpg	Coefficient	Std. err.	t	P> t	[95% conf. interval]	
gear_ratio	4.855506	1.522481	3.19	0.002	1.819013	7.891999
turn	-.8364698	.1440204	-5.81	0.000	-1.123709	-.5492302
foreign	-3.503218	1.433526	-2.44	0.017	-6.362296	-.6441404
_cons	40.865	8.692731	4.70	0.000	23.52789	58.2021

We want to add these regression results to the document. For most estimation commands, we can use the `etable` output type to add the elements of the displayed coefficient table. To export all columns of the default `regress` output, we need type only

```
. putpdf table reg = etable
```

But we need not stop there. In [example 3](#), we select a subset of the results and format them.

► Example 3: Export selected estimation results

Suppose we want to export only the point estimates and confidence intervals from the table above; we can also use `putpdf table` to remove the components that we do not want.

First, we create a new active document and add a table, `tbl2`, containing the estimation results from `regress`. We specify the `width` option to ensure that the table occupies the full page width of the document. Next, we want to remove the third through the fifth columns. To drop the fifth column, we specify `tbl2(.,5)` followed by the `drop` option. We work from right to left, dropping column five before four, because in this manner, the column numbers to the left of the newly dropped column do not change.

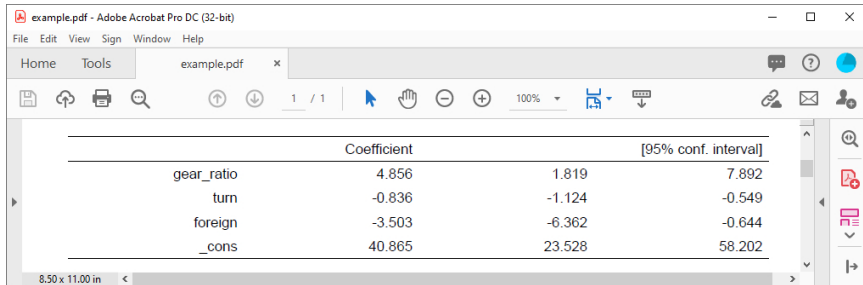
```
. putpdf begin
. putpdf table tbl2 = etable, width(100%)
. putpdf table tbl2(.,5), drop //drop p-value column
. putpdf table tbl2(.,4), drop //drop t column
. putpdf table tbl2(.,3), drop //drop SE column
```

Equivalently, we could have dropped the third column three times, because each time we drop it, the previous fourth column becomes the third.

Now that we have only the statistics we want, we can format our table by removing the border on the right side of the first column. To do this, we specify `nil` as the border pattern for the right border. We also format all estimates, in what will now be columns two through four, to have three decimal places by specifying the column indexes as a range. Finally, we erase the text “mpg” from the header for the first column and save our work.

```
. putpdf table tbl2(.,1), border(right, nil)
. putpdf table tbl2(.,2/4), nformat(%9.3f)
. putpdf table tbl2(1,1) = ("") // erase the content of first cell "mpg"
. putpdf save example, replace
successfully created "C:/mypath/example.pdf"
```

Our final table appears in `example.pdf` as follows:



The screenshot shows a PDF viewer window titled 'example.pdf - Adobe Acrobat Pro DC (32-bit)'. The table displayed is a regression results table with the following data:

	Coefficient	[95% conf. interval]	
gear_ratio	4.856	1.819	7.892
turn	-0.836	-1.124	-0.549
foreign	-3.503	-6.362	-0.644
_cons	40.865	23.528	58.202

In this example, because we wanted to use the same number of decimals for all estimates in our table and because we are using the `etable` output type, we could have preemptively set the format with our `regress` command. The modified version of the command is

```
. regress mpg gear_ratio turn foreign, noheader cformat(%9.3f)
```

This avoids the need to issue a separate formatting command.

The `etable` output type also works after `estimates table`, and you may find it easier to build a table of selected estimates prospectively. See [example 3](#) in [\[R\] estimates table](#) for an illustration.

Creating advanced tables

Sometimes, we need to create highly customized tables with complex layouts. `putpdf table` has many features that will allow you to create and incorporate sophisticated tables in your documents.

First, we demonstrate how to create a table from a matrix. While this method works with any Stata matrix, we will demonstrate by using a matrix of stored results from `regress`. See [\[U\] 14 Matrix expressions](#) for information on working with Stata matrices.

Exporting a matrix may be helpful when working with one of the few estimation commands that do not support the `etable` output type. After running any of these commands, matrices of stored results can be exported. For a list of estimation commands that do not support the `etable` output type, see [Unsupported estimation commands](#) of [\[RPT\] Appendix for putpdf](#).

In the following examples, we demonstrate how to use `putpdf table` to export tabulation results and create a customized regression table. These examples demonstrate advanced uses of `putpdf table`, which are often helpful. However, for tabulations and regression tables, you can also customize results using `table` or `collect` and export the results to your PDF file using `putpdf collect`. See [\[RPT\] putpdf collect](#) for more information on this often simpler solution.

▷ Example 4: Export selected estimation results from a matrix

To illustrate the basic use of matrix manipulation of stored results to create an estimation table, we re-create the simple estimation table from [example 3](#). The displayed results returned by `regress` are stored in the matrix `r(table)`, which can be viewed by typing `matrix list`.

```
. matrix list r(table)
r(table)[9,4]
      gear_ratio      turn      foreign      _cons
b    4.8555057    -0.83646975   -3.5032183   40.864996
se    1.5224812     .14402036    1.4335262    8.6927313
t     3.1892057    -5.8079965    -2.4437769    4.7010537
pvalue .0021348    1.704e-07    .01705791    .00001258
ll    1.8190127   -1.1237093    -6.3622962   23.527891
ul    7.8919987   -0.5492302    -6.4414044   58.202102
df          70          70          70          70
crit    1.9944371    1.9944371    1.9944371    1.9944371
eform          0          0          0          0
```

First, we create the matrix `rtable` as the transpose of `r(table)` because we want to see the variable names in rows. The point estimates and confidence intervals in the regression table can be extracted from the matrix `rtable`. We can extract columns 1, 5, and 6 from `rtable`, combine them, and assign them to another new matrix, `r_table`.

```
. matrix rtable = r(table)'
. matrix r_table = rtable[1...,1], rtable[1...,5..6]
```

Then we create a new active document and export `r_table` as a table with the name `tbl3`.

```
. putpdf begin
. putpdf table tbl3 = matrix(r_table), nformat(%9.3f) rownames colnames
> border(start, nil) border(insideH, nil) border(insideV, nil) border(end, nil)
```

In this table, all values imported from the matrix have been formatted as `%9.3f`. In addition, the row and column names of the matrix `r_table` are included as the first column and first row of the table. We keep only the top and bottom borders of the table by specifying `nil` for the leading edge border (`start`), trailing edge border (`end`), inside horizontal edges border (`insideH`), and inside vertical edges border (`insideV`).

The column names (`b`, `ll`, and `ul`) from the matrix may not be exactly what we want; we can modify them by customizing the corresponding cells. We can reset the contents and the horizontal alignment of the cells on the first row to give the columns new titles.

```
. putpdf table tbl3(1,2) = ("Coef."), halign(right)
. putpdf table tbl3(1,3) = ("[95% conf. interval]"), halign(right) colspan(2)
```

Afterward, we add back the bottom border of the first row and right-align the cells on the rest of the rows by specifying the row range as two through five.

```
. putpdf table tbl3(1,.), border(bottom)
. putpdf table tbl3(2/5,.), halign(right)
. putpdf save example, replace
successfully created "C:/mypath/example.pdf"
```

Our final table will be identical to that shown in [example 3](#).



► Example 5: Export tabulation results

Matrices are an alternative method for exporting results from estimation commands that do not support the `etable` output type, but we can also use them to export the results from nonestimation commands. For example, if we want to export the cross-tabulation of repair records and car origins, we can export these data as a matrix. We begin by adding a value label to the `rep78` variable.


```
. label define repairs 1 "Poor" 2 "Fair" 3 "Average" 4 "Good" 5 "Excellent"
. label values rep78 repairs
```

Next we tabulate our variables of interest, storing the frequencies in a matrix named `repairs`. We then create a new active document and add the matrix contents as a table, `tbl4`, adding a column at the end for the totals.

```
. tabulate foreign rep78, matcell(repairs)
```

Car origin	Repair record 1978					Total
	Poor	Fair	Average	Good	Excellent	
Domestic	2	8	27	9	2	48
Foreign	0	0	3	9	9	21
Total	2	8	30	18	11	69

```
. putpdf begin
. putpdf table tbl4 = matrix(repairs), rownames colnames
. putpdf table tbl4(.,6), addcols(1)
. putpdf table tbl4(1,7) = ("Total")
```

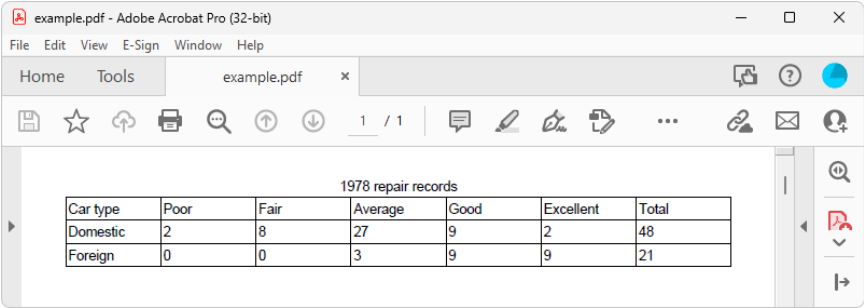
We have exported the tabulations, but we still need to label the columns and rows. We begin by looping over the two values of `foreign`, 0 and 1. We use the extended macro function `label` to store the labels for each category in the macro `'r1b1'`. Within that loop, we also loop over the five columns for repair records and label the columns similarly. To calculate the row totals, we create a macro `freq'row'_val` that points to the frequency of a given cell. We store the cumulative frequencies in the macro `cumul_freq'row'`, which is incremented in each run of the loop.

```
. forvalues i=0/1 {
2.   local r1b1: label (foreign) 'i'
3.   local row = 'i' + 2
4.   putpdf table tbl4('row',1) = ("'"r1b1'"')
5.
.   forvalues j=1/5 {
6.       local clb1: label (rep78) 'j'
7.       local cstart = 'j' + 1
8.       putpdf table tbl4(1,'cstart') = ("'"clb1'"')
9.       local freq'row'_val = repairs['i'+1,'j']
10.      local cumul_freq'row' = 'cumul_freq'row' + 'freq'row'_val'
11.      putpdf table tbl4('row',7) = ("'"cumul_freq'row'"')
12.      }
13. }
```

Once all the frequencies have been exported, we insert a row at the top of the table to provide a title. Additionally, we provide a label for the first column.

```
. putpdf table tbl4(1,.), addrows(1,before)
. putpdf table tbl4(1,1)= ("1978 repair records"), colspan(7) halign(center)
> border(left,nil) border(top,nil) border(right,nil)
. putpdf table tbl4(2,1)= ("Car type")
. putpdf save example, replace
successfully created "C:/mypath/example.pdf"
```

The table appears in our document as follows:



In the example above, we exported the cross-tabulation of `foreign` and `rep78` using the matrix of stored results. To complete our table, we added a column for the totals and a row for the title. Similarly, in your work you may need to add components to a table after the initial export of results. In the example below, we demonstrate how to create a table by gradually adding components.

► Example 6: Create a table dynamically

Below, we use Census data recording the deathrate (`drate`) and median age (`medage`) for each state. The data also record four regions of the country in which each state is located, NE, N Cntrl, South, and West. We fit a linear regression model and store the transpose of rows 1, 5, and 6 in column matrices named `b`, `ll`, and `ul`.

```
. use https://www.stata-press.com/data/r19/census9
(1980 Census data by state)
. regress drate i.region medage [aw=pop], noheader
(sum of wgt is 225,907,472)
```

drate	Coefficient	Std. err.	t	P> t	[95% conf. interval]	
region						
N Cntrl	.3138738	2.456431	0.13	0.899	-4.633632	5.26138
South	-1.438452	2.320244	-0.62	0.538	-6.111663	3.234758
West	-10.90629	2.681349	-4.07	0.000	-16.30681	-5.505777
medage	4.283183	.5393329	7.94	0.000	3.196911	5.369455
_cons	-39.14727	17.23613	-2.27	0.028	-73.86262	-4.431915

```
. matrix rtable = r(table)
. matrix b = rtable[1,1...]'
. matrix ll = rtable[5,1...]'
. matrix ul = rtable[6,1...]'
```

Our goal is to collect the point estimates and confidence intervals and output a table that looks like the following:



Variable	Coef.	95% C.I.
region		
NE	Ref.	N/A
N Cntrl	0.3	-4.6 to 5.3
South	-1.4	-6.1 to 3.2
West	-10.9	-16.3 to -5.5
medage	4.3	3.2 to 5.4
Intercept	-39.1	-73.9 to -4.4

To create our table of point estimates and confidence intervals, we add a 1×3 table with no borders to our new active document and fill the single row with Variable, Coef., and 95% C.I.. We also set the table width to be 4 inches, put the table in the center of the document, and add back the top and bottom borders.

```
. putpdf begin
. putpdf table tbl5 = (1,3), border(all,nil) width(4) halign(center)
. putpdf table tbl5(1,1)=("Variable"), border(top) border(bottom)
. putpdf table tbl5(1,2)=("Coef."), halign(center) border(top) border(bottom)
. putpdf table tbl5(1,3)=("95% C.I."), halign(right) border(top) border(bottom)
```

Afterward, we add one row to the end of the table and fill in the content of the first column in this row as region.

```
. putpdf table tbl5(1,.), addrows(1)
. putpdf table tbl5(2,1)=("region")
```

In the resulting table, the region variable has four levels, and each level takes up one row. The medage variable and the constant term take up another two rows. For each row, the first column contains the variable label, the second column contains the point estimate that is stored in the matrix b, and the third column contains a formatted string of the lower limit and upper limit of the confidence intervals. Those two levels are stored in matrix l1 and u1, respectively.

Based on the above information, we add those rows one by one at the end of the table and fill in the content and format for each cell in the corresponding row. Notice that NE is the base level of region; its point estimate and confidence interval are replaced by Ref. and N/A in the resulting table. In addition, we reset the top and bottom borders for medage and Intercept.

```

. local row 2
. local i 1
. foreach name in "NE" "N Cntrl" "South" "West" "medage" "Intercept" {
2.     putpdf table tbl5('row',.), addrows(1)
3.     local ++row
4.     if "'name'"=="NE" {
5.         local coef "Ref."
6.         local ci "N/A"
7.     }
8.     else {
9.         local coef : display %5.1f b['i',1]
10.        local low : display %5.1f ll['i',1]
11.        local upp : display %5.1f ul['i',1]
12.        local ci "'low' to 'upp'"
13.    }
14.    putpdf table tbl5('row', 1) = ("'name'"), halign(right)
15.    putpdf table tbl5('row', 2) = ("'coef'"), halign(center)
16.    putpdf table tbl5('row', 3) = ("'ci'"), halign(right)
17.    if "'name'"=="medage" | "'name'"=="Intercept" {
18.        putpdf table tbl5('row', 1), halign(left) border(top)
>        border(bottom)
19.        putpdf table tbl5('row', 2), border(top) border(bottom)
20.        putpdf table tbl5('row', 3), border(top) border(bottom)
21.    }
22.    local ++i
23. }
. putpdf save example, replace
successfully created "C:/mypath/example.pdf"

```



Aside from estimation results and summary statistics, you may want to export images to a PDF file. Images can be exported simply by appending them to an active paragraph or by inserting them in a table. The same formatting options are available regardless of how you export an image. Of course, when inserting an image in a table, the size is constrained to the dimensions of the given cell.

► Example 7: Nesting images in a table

We might want to add various images to the document and align them side by side, row by row, or both. To be more clear, we use an example to illustrate this purpose. In this example, we use another dataset—the Second National Health and Nutrition Examination Survey (NHANES II) ([McDowell et al. 1981](#)).

First, we fit a three-way full factorial model of systolic blood pressure on age group, sex, and body mass index (BMI). Then, we estimate the predictive margins for each combination of agegrp and sex at levels of BMI from 10 through 40 at intervals of 10 and graph the results.

```

. use https://www.stata-press.com/data/r19/nhanes2, clear
. regress bpsystol agegrp##sex##c.bmi
(output omitted)
. forvalues v=10(10)40 {
2.     margins agegrp, over(sex) at(bmi='v')
3.     marginsplot, title("")
4.     graph export bmi'v'.png
5. }
(output omitted)

```

Now we want to add those four plots into the document, requiring that the margins plots for `bmi=10` and `bmi=20` lay side by side on top of the other two side-by-side margins plots for `bmi=30` and `bmi=40`. It is also required that each plot have a subtitle indicating the level of BMI and that the final figure have a title. This complicated layout can be accomplished using `putpdf table`.

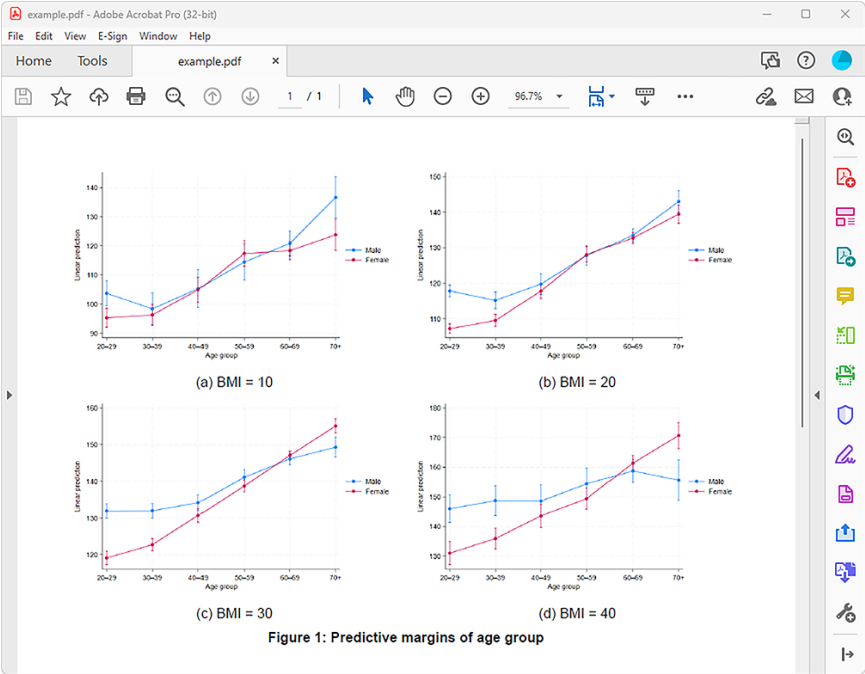
We create a new active document with a 4×2 table and remove all of its borders. We caption our table by using the `note()` option. In each cell on the odd rows, we add a plot. We fill each cell on the even rows with the title corresponding to the plot above it and center-align the text in the cell.

```
. putpdf begin
. putpdf table tbl6 = (4,2), border(all,nil)
> note(Figure 1: Predictive margins of age group)
. putpdf table tbl6(1,1)=image(bmi10.png)
. putpdf table tbl6(2,1)=("(a) BMI = 10"), halign(center)
. putpdf table tbl6(1,2)=image(bmi20.png)
. putpdf table tbl6(2,2)=("(b) BMI = 20"), halign(center)
. putpdf table tbl6(3,1)=image(bmi30.png)
. putpdf table tbl6(4,1)=("(c) BMI = 30"), halign(center)
. putpdf table tbl6(3,2)=image(bmi40.png)
. putpdf table tbl6(4,2)=("(d) BMI = 40"), halign(center)
```

We formatted our table and the cell contents as we added content to the document. However, we would also like to format the caption. The `note()` option adds an additional row at the end of the table that spans all the columns of the table. We can format the text of the note by specifying the last row (in this case, 5) and either `.` or `1` as the column index. Here, we center-align and bold the text of the caption. Because note text is always placed within a single merged cell, it does not matter how short or long the text is when you identify the cell location.

```
. putpdf table tbl6(5,.), halign(center) bold
. putpdf save example, replace
successfully created "C:/mypath/example.pdf"
```

This creates a table in `example.pdf` that looks like the following:



Stored results

`putpdf describe tablename` stores the following in `r()`:

Scalars

<code>r(nrows)</code>	number of rows in the table
<code>r(ncols)</code>	number of columns in the table

Reference

McDowell, A., A. Engel, J. T. Massey, and K. Maurer. 1981. "Plan and operation of the Second National Health and Nutrition Examination Survey, 1976–1980". In *Vital and Health Statistics*, ser. 1, no. 15. Hyattsville, MD: National Center for Health Statistics.

Also see

[RPT] [putpdf intro](#) — Introduction to generating PDF files

[RPT] [putpdf begin](#) — Create a PDF file

[RPT] [putpdf collect](#) — Add a table from a collection to a PDF file

[RPT] [putpdf pagebreak](#) — Add breaks to a PDF file

[RPT] [putpdf paragraph](#) — Add text or images to a PDF file

[RPT] [Appendix for putpdf](#) — Appendix for putpdf entries

Description Also see

Description

This is the collective appendix for the putpdf entries.

Colors

color, bgcolor, fgcolor, bcolor

aliceblue	darkslategray	lightsalmon	palevioletred
antiquewhite	darkturquoise	lightseagreen	papayawhip
aqua	darkviolet	lightskyblue	peachpuff
aquamarine	deeppink	lightslategray	peru
azure	deepskyblue	lightsteelblue	pink
beige	dimgray	lightyellow	plum
bisque	dodgerblue	lime	powderblue
black	firebrick	limegreen	purple
blanchedalmond	floralwhite	linen	red
blue	forestgreen	magenta	rosybrown
blueviolet	fuchsia	maroon	royalblue
brown	gainsboro	mediumaquamarine	saddlebrown
burlywood	ghostwhite	mediumblue	salmon
cadetblue	gold	mediumorchid	sandybrown
chartreuse	goldenrod	mediumpurple	seagreen
chocolate	gray	mediumseagreen	seashell
coral	green	mediumslateblue	sienna
cornflowerblue	greenyellow	mediumspringgreen	silver
cornsilk	honeydew	mediumturquoise	skyblue
crimson	hotpink	mediumvioletred	slateblue
cyan	indianred	midnightblue	slategray
darkblue	indigo	mintcream	snow
darkcyan	ivory	mistyrose	springgreen
darkgoldenrod	khaki	moccasin	steelblue
darkgray	lavender	navajowhite	tan
darkgreen	lavenderblush	navy	teal
darkkhaki	lawngreen	oldlace	thistle
darkmagenta	lemonchiffon	olive	tomato
darkolivegreen	lightblue	olivedrab	turquoise
darkorange	lightcoral	orange	violet
darkorchid	lightcyan	orangered	wheat
darkred	lightgoldenrodyellow	orchid	white
darksalmon	lightgray	palegoldenrod	whitesmoke
darkseagreen	lightgreen	palegreen	yellow
darkslateblue	lightpink	paleturquoise	yellowgreen

Unsupported estimation commands

Command name	Available stored results documented in
bayesmh	[BAYES] bayesmh
bayes	[BAYES] bayes
boxcox	[R] boxcox
exlogistic	[R] exlogistic
expoisson	[R] expoisson
menl	[ME] menl
rocfits	[R] rocfits
varbasic	[TS] varbasic
xthtaylor	[XT] xthtaylor
xtpcse	[XT] xtpcse

Also see

[RPT] **putpdf intro** — Introduction to generating PDF files

[RPT] **putpdf begin** — Create a PDF file

[RPT] **putpdf paragraph** — Add text or images to a PDF file

[RPT] **putpdf pagebreak** — Add breaks to a PDF file

[RPT] **putpdf table** — Add tables to a PDF file

Description

`set docx_hardbreak` specifies whether `putdocx textblock` inserts a space at the beginning of lines or respects spaces exactly as they are typed. `set docx_hardbreak off`, the default, respects spaces at the beginning and end of lines exactly as they are typed in the text block. `set docx_hardbreak on` inserts a space at the beginning of all nonempty lines, excluding the first line in the block of text and the first line of each paragraph. Thus, `set docx_hardbreak on` adds a space after hard line breaks.

`set docx_paramode` specifies whether `putdocx textblock` treats blank lines in a text block as an indication of a new paragraph. `set docx_paramode off`, the default exports the block of text as a single paragraph, treating blank lines as if they were not there. `set docx_paramode on` specifies that a blank line signals a new paragraph. The text following each blank line will begin a new paragraph.

Syntax

Set whether spaces are added after hard line breaks

```
set docx_hardbreak { on | off }
```

Set whether empty lines signal the beginning of a new paragraph

```
set docx_paramode { on | off }
```

Remarks and examples

The `set docx_hardbreak` and `set docx_paramode` commands control the spacing of paragraph content added with `putdocx textblock` commands.

By default, the content enclosed within a set of `putdocx textblock` commands is added to the `.docx` file with the beginning and end of line spaces respected exactly as they are typed and with any empty lines removed. To avoid having to include a space between the last word on each line and the first word on the following line, you can type

```
set docx_hardbreak on
```

This setting tells Stata to add an extra space at the beginning of all nonempty lines in the text block, except for the first line in the text block and the first line of each paragraph.

To insert paragraph breaks within a text block, you can type

```
set docx_paramode on
```

This setting tells Stata to begin a new paragraph with the content following an empty line. Empty lines refer to lines without any text, spaces, or tab characters.

When specified interactively, these settings will apply throughout the Stata session. When used within a do-file, Stata automatically restores the previous `set docx_hardbreak` and `set docx_paramode` settings when the do-file concludes.

► Example 1

Suppose we are writing a report on low birthweights using the data described in [Hosmer, Lemeshow, and Sturdivant \(2013, 24\)](#).

```
. use https://www.stata-press.com/data/r19/lbw
(Hosmer & Lemeshow data)
```

This will be an extensive document with multiple paragraphs, so it will be convenient to write multiple paragraphs within single blocks of text. Also, to avoid needing to add a space between the last word on one line and the first word on the following line, we can request that a space be added after hard line breaks, in which case Stata will automatically insert an extra space at the beginning of each line of text. We could specify the `paramode` and `hardbreak` options with each `putdocx textblock` command, but it will be simpler to change the settings before we begin creating our document. We specify these settings as follows:

```
set docx_paramode on
set docx_hardbreak on
```

Below, we count the number of mothers that smoked during pregnancy and store this number in the local macro `'smoker'`. We store the average birthweight for their babies in `'smoke_wt'`. Then, we summarize birthweight (`bwt`), which stores the total number of women in this dataset in `r(N)`.

```
summarize bwt if smoke==1
local smoker = 'r(N)'
local smoke_wt = 'r(mean)'
summarize bwt
putdocx begin
```

After creating a document in memory, we refer to these macros in the text block below:

```
putdocx textblock begin
We use data from Hosmer and Lemeshow to study the contributing factors to low
birthweight. This dataset contains information on each woman's weight, premature
labor history, and whether she smoked during pregnancy.

Smoking during pregnancy is associated with lower birthweights. Out of the
<<dd_docx_display: 'r(N)''>> women in this dataset, <<dd_docx_display: 'smoker''>>
smoked during pregnancy. The average birthweight is
<<dd_docx_display nformat(%5.2f): 'smoke_wt''>> grams for babies whose mothers smoked
putdocx textblock end
```

In this first block of text, we have two paragraphs separated by an empty line. With the settings specified, the empty line will indicate a paragraph break, and the text following the empty line will be added to the `.docx` file as a new paragraph. Also, a space will be added between the last word on each line and the first word on the line that follows.

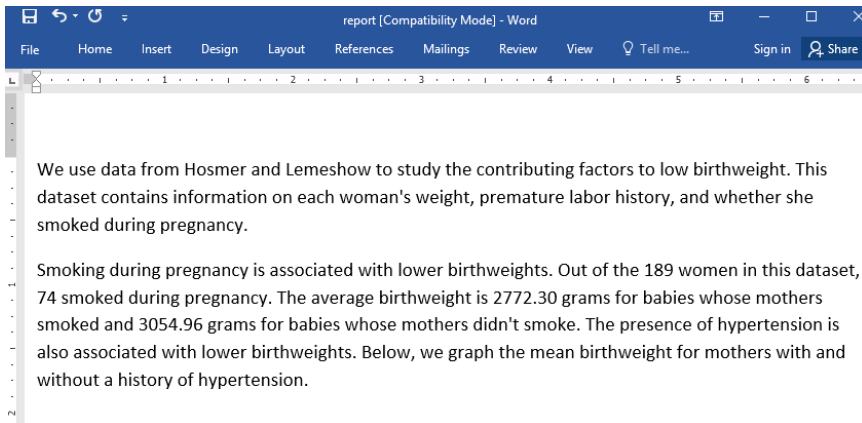
Below, we append some text regarding the mean birthweight for babies whose mothers did not smoke. We specify the `noparamode` option with `putdocx textblock append` to override the settings we previously specified.

```
summarize bwt if smoke==0
putdocx textblock append, noparamode
    and <<dd_docx_display nformat(%5.2f):'r(mean)''>> grams for babies whose mothers
didn't smoke.
```

The presence of hypertension is also associated with lower birthweights. Below, we graph the mean birthweight for mothers with and without a history of hypertension.

```
putdocx textblock end
putdocx save report, replace
```

Thus, `report.docx` looks like this:



Note that a paragraph break was not inserted in the block of text we appended.



Reference

Hosmer, D. W., Jr., S. A. Lemeshow, and R. X. Sturdivant. 2013. *Applied Logistic Regression*. 3rd ed. Hoboken, NJ: Wiley.

Also see

[RPT] [putdocx paragraph](#) — Add text or images to an Office Open XML (.docx) file

Glossary

dynamic document. A dynamic document contains both static narrative and dynamic tags. Any changes in the data or in Stata will change the output as the document is created. The main advantages of using dynamic documents are 1) results in the document come from executing commands instead of being copied from Stata and pasted into the document; 2) no need to maintain parallel do-files; and 3) any changes in data or in Stata are reflected in the final document when it is created.

dynamic tags. Dynamic tags are instructions that appear in the source files from which dynamic documents are created. These dynamic tags specify actions to be taken when source files are processed by Stata's dynamic documents commands, [dyndoc](#) and [dyntext](#). For instance, dynamic tags can indicate that a block of Stata code be run, that the result of a Stata expression be inserted in text, and that a Stata graph be exported to an image file and a link to the image file be included in the dynamic document; see [\[RPT\] Dynamic tags](#) for a complete list of available dynamic tags.

dynamic text file. A dynamic text file contains both plain text and dynamic tags. A dynamic text file can be processed by [dyntext](#) to create a text file that incorporates Stata results.

Markdown. Markdown is an easy-to-read, plain text, lightweight markup language. For a detailed discussion and the syntax of Markdown, see the [Markdown Wikipedia page](#).

Markdown is easily converted to an output format such as HTML. Stata uses Flexmark's Pegdown emulation as its default Markdown document processing engine. For information on Pegdown's flavor of Markdown, see the [Pegdown GitHub page](#).

Subject and author index

See the [combined subject index](#) and the [combined author index](#) in the *Stata Index*.