putpdf table — Add tables to a PDF file

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(numlist_i, .), cell_fmt_options
putpdf table tablename(., numlist_j), cell_fmt_options
putpdf table tablename(numlist_i, numlist_j), 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**.

table_options	Description
`memtable`	keep table in memory rather than add it to document
`width(#[`*unit*`|%]|`*matname*`)`	set table width
`halign(`*hvalue*`)`	set table horizontal alignment
`indent(#[`*unit*`])`	set table indentation
`spacing(`*position*`, #[`*unit*`])`	set spacing before or after table
`border(`*bspec*`)`	set pattern and color for border
`title(`*string*`[ , `*cell_fmt_options*`])`	add a title to the table
`note(`*string*`[ , `*cell_fmt_options*`])`	add a note to the table

cell_options	Description
`append`	append objects to current content of cell
`rowspan(#)`	merge cells vertically
`colspan(#)`	merge cells horizontally
`span(`$\#_1$`, `$\#_2$`)`	merge cells both horizontally and vertically
`linebreak[ (#) ]`	add line breaks into the cell
cell_fmt_options	options that control the look of cell contents

row_col_options	Description
`nosplit`	prevent row from breaking across pages
`addrows(#[ , before|after])`	add # rows in specified location
`addcols(#[ , before|after])`	add # columns in specified location
`drop`	drop specified row or column
cell_fmt_options	options that control the look of cell contents

cell_fmt_options	Description
`margin(`*type*`, #[`*unit*`])`	set margins
`halign(`*hvalue*`)`	set horizontal alignment
`valign(`*vvalue*`)`	set vertical alignment
`border(`*bspec*`)`	set pattern and color for border
`bgcolor(`*color*`)`	set background color
`nformat(`*%fmt*`)`	specify numeric format for cell text
`font(`*fspec*`)`	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`	strikeout text
`underline`	underline text
`allcaps`	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 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 *tablename*`1`, *tablename*`2`, ..., *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:

returnset	Description
escalars	all ereturned scalars
rscalars	all returned scalars
emacros	all ereturned macros
rmacros	all returned macros
ematrices	all ereturned matrices
rmatrices	all returned matrices
e*	all ereturned scalars, macros, and matrices
r*	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*`(`*numlist*$_i$`,` *numlist*$_j$`)` (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*`(`*numlist*$_i$`, .)` specify all cells on the *i*th row or on the rows identified by *numlist*$_i$.

tablename`(., `*j*`)` and *tablename*`(., `*numlist*$_j$`)` 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

`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.

`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

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

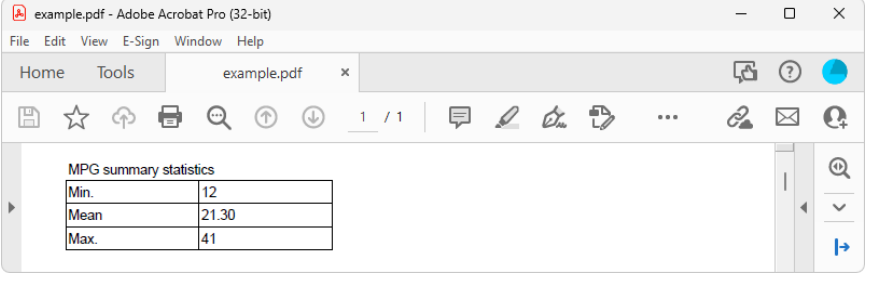
When we describe our table, it reports four rows instead of the three we specified. When we add a title, it is included as the first row of the table without any borders. With our table in place, we now run `summarize mpg` and view the list of stored results:

```
. 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.

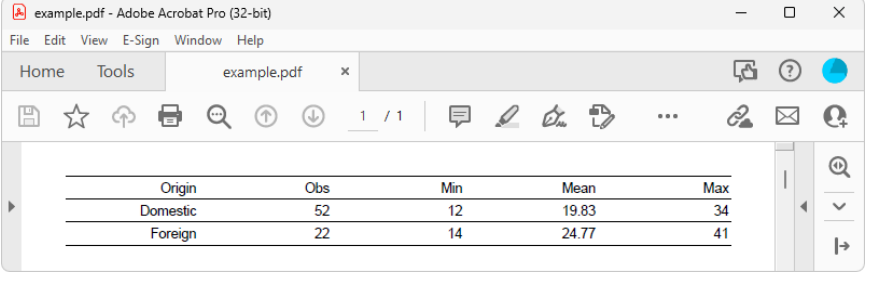
```
. rename foreign Origin
. putpdf table tbl1 = data("Origin Obs Min Mean Max"), varnames
> border(start, nil) border(insideV, nil) border(end, nil)
```

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.

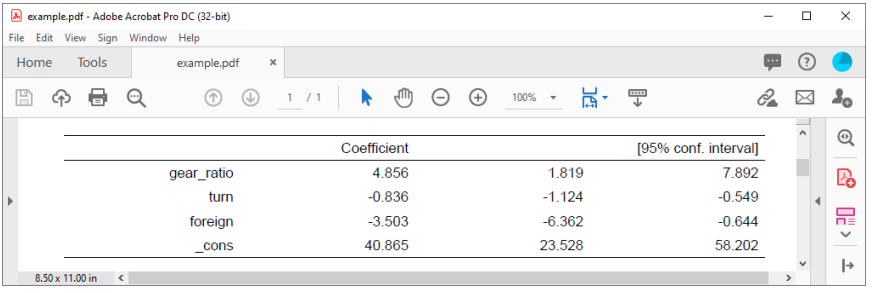
```
. putpdf begin
. putpdf table tbl2 = etable, width(100%)
. putpdf table tbl2(.,5), drop //drop p-value column
. putpdf table tbl2(.,4), drop //drop t column
. putpdf table tbl2(.,3), drop //drop SE column
```

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:



	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  -.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 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					
	Poor	Fair	Average	Good	Excellent	Total
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")
```

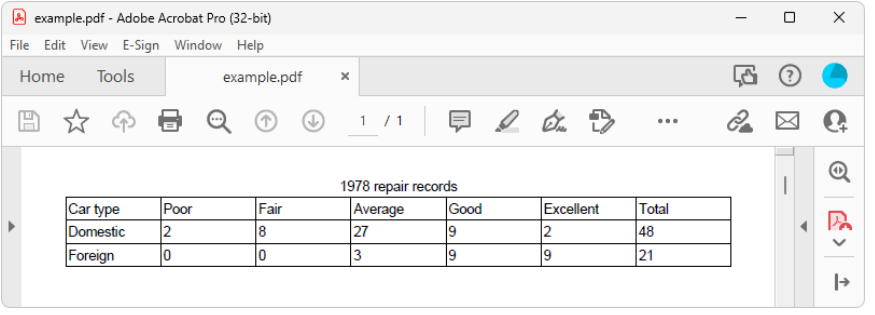
We have exported the tabulations, but we still need to label the columns and rows. We begin by looping over the two values of foreign, 0 and 1. We use the extended macro function `label` to store the labels for each category in the macro `‘rlbl’`. Within that loop, we also loop over the five columns 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 rlbl: label (foreign) ‘i’
  3.  local row = ‘i’ + 2
  4.  putpdf table tbl4(‘row’,1) = (‘"‘rlbl’"’)
  5.
.     forvalues j=1/5 {
  6.        local clbl: label (rep78) ‘j’
  7.           local cstart = ‘j’ + 1
  8.              putpdf 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.           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:



1978 repair records

Car type	Poor	Fair	Average	Good	Excellent	Total
Domestic	2	8	27	9	2	48
Foreign	0	0	3	9	9	21

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 `ll` and `ul`, 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.

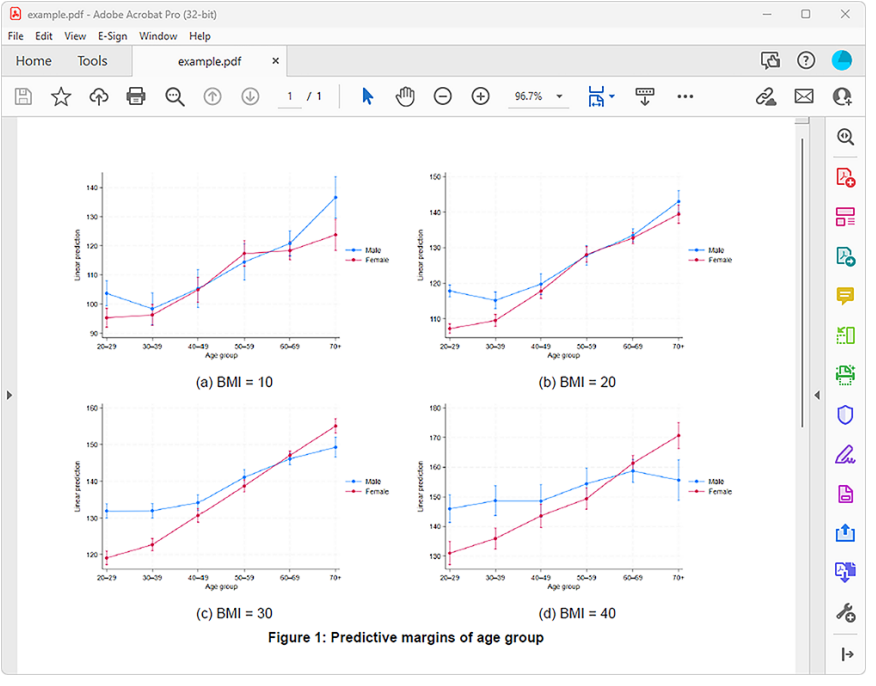
We create a new active document with a 4×2 table and remove all of its borders. We caption our table by using the note() option. In each cell on the odd rows, we add a plot. We fill each cell on the even rows with the title corresponding to the plot above it and center-align the text in the cell.

```
. 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

`putdpdf describe` *tablename* stores the following in `r()`:

Scalars

`r(nrows)`	number of rows in the table
`r(ncols)`	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