STATA CUSTOMIZABLE TABLES AND COLLECTED RESULTS REFERENCE MANUAL
RELEASE 19

A Stata Press Publication
StataCorp LLC
College Station, Texas

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

ISBN-10: 1-59718-447-0
ISBN-13: 978-1-59718-447-2

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 Customizable Tables and Collected Results Reference Manual*. College Station, TX: Stata Press.

Contents

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

Description

This is the [TABLES] manual. What is surprising is that, used alone, the commands in this manual cannot create a table. The tables created here are all based on results collected from other commands, commands not documented in this manual, commands like `regress`, `margins`, `bayes`, `ttest`, `mi`, `mean`, `table`, and so on.

This manual documents the commands that collect results from other commands; lay out those results into one-way, two-way, or multiway tables; customize the headers of those tables; change the appearance of the results; and export the tables to Microsoft Word, Microsoft Excel, PDF, HTML, LaTeX, SMCL, or Markdown.

We call the collected results from one or more commands collections. The purpose of this manual is to explain how to create collections, manage collections, create tables from collections, and export those tables.

For more details about how to create tables step by step, see Mitchell (2025).

Remarks and examples

Remarks are presented under the following headings:

What is in this manual?

Presenting results is the final step for most research, and a major part of presenting results is creating effective tables.

Here are some tables created with the collection system:

```
. use https://www.stata-press.com/data/r19/nhanes2l
. table ...
...
...
...
. collect layout ...
. collect preview
```

	Male		Female	
Diabetes status				
Not diabetic	4698.0	(95.6)	5152.0	(94.8)
Diabetic	217.0	(4.4)	282.0	(5.2)
Age, mean (sd)	47.4	(17.2)	47.7	(17.3)
BMI, mean (sd)	25.5	(4.0)	25.6	(5.6)
Health status				
Excellent	1252.0	(25.5)	1155.0	(21.3)
Very good	1213.0	(24.7)	1378.0	(25.4)
Good	1340.0	(27.3)	1598.0	(29.5)
Fair	722.0	(14.7)	948.0	(17.5)
Poor	382.0	(7.8)	347.0	(6.4)
Systolic BP, mean (sd)	132.9	(21.0)	129.1	(25.1)

```
. use https://www.stata-press.com/data/r19/nhanes2l
. collect: ...
...
...
...
. collect layout ...
. collect preview
```

	Region				
	NE	MW	S	W	All
Age (years)					
Odds ratio	1.07	1.06	1.06	1.06	1.06
SE	(0.01)	(0.01)	(0.01)	(0.01)	(0.00)
Weight (kg)					
Odds ratio	1.03	1.03	1.02	1.02	1.03
SE	(0.01)	(0.01)	(0.01)	(0.01)	(0.00)
Female					
Odds ratio	1.93	1.41	1.48	1.25	1.48
SE	(0.44)	(0.27)	(0.26)	(0.25)	(0.15)
Intercept					
Odds ratio	0.00	0.00	0.00	0.00	0.00
SE	(0.00)	(0.00)	(0.00)	(0.00)	(0.00)

This manual is a little bit about the `collect:` prefix. It is a lot about the

```
...
...
...
```

you saw above in the command listings.

What are collections?

Collections are the collected results from one or more commands. They contain every result stored by the commands. They also contain labels for everything in the collection. Some labels are system default labels such as "Coefficient" for regression coefficients or χ^2 for chi-squared statistics. Some labels come from you and your dataset. If the collected commands reference variables, such as regression coefficients, the variables will be labeled with the variable labels from the dataset. If the collected commands use categorical variables that are value labeled, those labels become part of the collection.

Collections also contain styles. Styles determine how everything looks on the tables you create. Styles determine how row and column headers are composed. Styles determine what numeric format is used. Styles determine whether text is bolded, italicized, colored, etc. When you create a collection, it will have the default system styles unless you specified that it start from a set of styles you previously saved.

You have great control over which values or labels are affected by styles. You can choose an overall numeric format for all values or a custom numeric format for coefficients, their standard errors, and their confidence intervals. All of this while using a different format for their z statistics and a yet different format for their p-values. And the control can be even finer still. You can choose to have all coefficient statistics for the variable `weight` highlighted by giving their cells a light-blue background.

You can modify anything in a collection. You can modify any label. You can modify any style.

You cannot modify values. Those were produced by your commands, and they are sacrosanct. You can modify anything that identifies, labels, formats, or presents those values.

You reference everything stored in a collection using tags. Those tags are created automatically for you when you collect results. You can also specify additional tags while collecting, and you can even remap tags in a collection. Tags are organized into groupings called dimensions; this organization makes it easier to specify what you want on the rows and columns of your tables.

For an illustrative look "under the hood" of the collection system, see Mitchell (2025, chap. 5).

Do you need collections?

There are three primary reasons you may need collections.

1. You want to create a table showing the results from more than one command.
2. You want to customize how a table looks—layout, headers, numeric formats, bolding, italics, colors, etc.
3. You want to present your results in Microsoft Word, Microsoft Excel, PDF, HTML, LaTeX, SMCL, or Markdown.

Other commands in Stata are built for creating specific kinds of tables from data. Conversely, collections do not create results from data. They give you a framework to format the results you have collected from other commands.

Collecting results is easy; simply prefix almost any command with `collect:`, or type `collect get` after the command has run. You have created a collection. Every time you type `collect:` or `collect get` again, you are adding to a collection.

The dtable and etable commands

The collection system was designed to give you the flexibility to create a table specific to your style and needs. However, there are certain tables that are used so commonly that they necessitated their own special commands. These commands are `dtable` and `etable`, and they are designed for creating and exporting tables all in a single step, without any knowledge of the commands in this manual.

`dtable` is designed to create a table of descriptive statistics, such as summary statistics for continuous and categorical variables, summary statistics for groups of your data, and tests of equality. `etable` is designed to create a table of estimation results, from one or more models. You can use it to create a table with the active estimation results, results from a preceding `margins` command, or results you stored with `estimates store`.

Both `dtable` and `etable` allow you to create a table complete with a title and notes, and export it to Microsoft Word, Microsoft Excel, PDF, HTML, LaTeX, SMCL, or Markdown. Additionally, they automatically store the results in a collection, allowing you to customize the table further with the `collect` suite of commands. To learn more, see [R] **dtable** and [R] **etable**.

If your goal is to create a table with a combination of estimation results, summary statistics, and other results, you will benefit from learning about the `table` command below.

The table command

Stata has a couple of different commands that produce tabular results. If your intent is to summarize data to understand them, you may need only these commands and not need collections. However, if you want to customize the layout and appearance of the results from these commands, you will need to collect their results.

The `table` command is unique in that it creates a stunningly large range of tables and stores the results in a collection. If you came here to do any of the following things, you should begin with the `table` command:

- One-way tabulations of frequencies, percentages, and proportions
- Two-way tabulations of frequencies, percentages, and proportions
- Multiway tabulations of frequencies, percentages, and proportions
- One-way, two-way, and multiway tables of summary statistics
- Tables of hypothesis tests
- Tables of regression results (this includes the possibility of multiple regression commands)
- Combinations of the above

If you want to customize the results of `table` or export them, simply start with a `table` command, and then use any of the collection commands in this manual to customize and export the results. You may find that you need to combine the results of several `table` commands to create a collection with all the results you need on a table. You can do that by using `collect combine` or the `append` option of `table`. You can even combine the results from the `table` command with other commands such as regression commands, `lincom`, `nlcom`, `ttest`, `margins`, and more.

Acknowledgments

We are grateful to the authors of community-contributed commands for creating and exporting tables. We thank John Luke Gallup of Portland State University for `outreg`; Ian Watson, freelance researcher and of the University of New South Wales, for `tabout`; Ben Jann of the University of Bern for `estout`; Roy Wada of the Boston Public Health Commission for `outreg2`; Attaullah Shah of the Institute of Management Sciences for `asdoc`; and many more for their previous and ongoing contributions to Stata's reporting capabilities.

Reference

Mitchell, M. N. 2025. *Create and Export Tables Using Stata*. College Station, TX: Stata Press.

Description

A brief overview of the manual and suggested reading order.

Remarks and examples

If you are new to collections, read almost all the intros in order. In particular, read these intros in order:

[TABLES] **Intro**	Introduction
[TABLES] **Intro 1**	How to read this manual (this entry)
[TABLES] **Intro 2**	A tour of concepts and commands
[TABLES] **Intro 3**	Workflow outline

[TABLES] **Intro 3** also serves as a reference. The workflow is both a workflow and an overview of the commands within that workflow.

We did not list [TABLES] **Intro 4** and [TABLES] **Intro 5** for immediate reading. If you would like to see all the commands organized by their function, then read

[TABLES] **Intro 4**	Overview of commands

Finally, to see a list of all the tabulation commands that we do not address in this manual, see

[TABLES] **Intro 5**	Other tabulation commands

A few of the commands listed here return some of their results, but not their tabulations, in `r()`, and those results can be collected.

After reading the introduction entries on the reading list, you will be ready to start any tables project. You will need the remaining entries as reference.

Also see

[TABLES] **Intro 2** — A tour of concepts and commands

[TABLES] **Intro 3** — Workflow outline

Description Remarks and examples Also see

Description

There is one key concept on which the collection system is built—tags.

In this entry, we introduce tags and how they are created and used by the `collect` commands. Along the way, we will introduce several of the most important `collect` commands. Our focus here is on concepts and general features. We will not attempt to cover everything. See [TABLES] **Intro 3** for the quickest overview of the features.

We make no attempt in this entry to create pretty or interesting tables. Our sole purpose is to introduce concepts and commands.

Remarks and examples

Remarks are presented under the following headings:

Tags, dimensions, and levels

Your goal is to construct tables from the results of one or more commands. You need something to organize results from commands in such a way that you can conveniently place the results onto the rows and columns of tables. You would also like to control how everything looks, from the row and column headers to numeric formats, or even to the background color of an emphasized result. You do all that using the collection system, and the collection system needs to do lots of bookkeeping. The bookkeeping system for `collect` is tags.

We start by collecting results. Collecting results is as simple as placing the prefix `collect` in front of any command that returns results. Let's also place a `by` prefix in front of our command so we have results by each level of the `by` variables.

Introducing collect:

```
. use https://www.stata-press.com/data/r19/nhanes2l
(Second National Health and Nutrition Examination Survey)

. collect clear

. sort sex region

. collect: by sex region: summarize weight

--------------------------------------------------------------------------------
-> sex = Male, region = NE

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
      weight |      1,018    78.15295    12.89267      47.17     129.84

--------------------------------------------------------------------------------
-> sex = Male, region = MW

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
      weight |      1,310    78.24791    13.50132       41.5     139.03

--------------------------------------------------------------------------------
-> sex = Male, region = S

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
      weight |      1,332     77.5923    14.27054      30.84     158.53

--------------------------------------------------------------------------------
-> sex = Male, region = W

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
      weight |      1,255    77.98812     13.6871      44.11     175.88

--------------------------------------------------------------------------------
-> sex = Female, region = NE

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
      weight |      1,078    65.50096     14.0839      39.12     148.21

--------------------------------------------------------------------------------
-> sex = Female, region = MW

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
      weight |      1,464    66.50488     14.7564      34.93     159.44

--------------------------------------------------------------------------------
-> sex = Female, region = S

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
      weight |      1,521    67.16907    15.19103      35.27     138.91

--------------------------------------------------------------------------------
-> sex = Female, region = W

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
      weight |      1,373    66.11902    14.66786      36.06     134.61
```

So we have computed means, standard deviations, and the minimum and maximum of `weight` for each combination of the levels of variables `sex` and `region`. By placing the `collect:` prefix in front of the `by:` command, we have collected those results into the default collection. We `collect clear`ed first to be sure we were starting clean and not adding to an existing collection.

For readers more familiar with pounds, these weights are in kilograms; you can double these numbers in your head. Or multiply by 2.2 to be more accurate.

What do we mean by "collected"? The results have been stored, but more importantly they have been tagged. Let's trim down all of that `by:` output and focus on the means. Here is a "picture" of what `collect:` has done.

```
================================       ===========================================
by sex region: summarize weight  -->                  Collection
================================       ===========================================
```

Or, more specifically,

```
================================       ===========================================
 Variable |   ...      Mean            value            tags
----------+--------------------        ------ ------------------------------------

   sex = Male, region = NE
   weight |   ...  78.15295 ...  -->   78.15 sex[Male]   region[NE] result[mean]

   sex = Male, region = MW
   weight |   ...  78.24791 ...  -->   78.25 sex[Male]   region[MW] result[mean]

   sex = Male, region = S
   weight |   ...   77.5923 ...  -->   77.59 sex[Male]   region[S]  result[mean]

   sex = Male, region = W
   weight |   ...  77.98812 ...  -->   77.99 sex[Male]   region[W]  result[mean]

   sex = Female, region = NE
   weight |   ...  65.50096 ...  -->   65.50 sex[Female] region[NE] result[mean]

   sex = Female, region = MW
   weight |   ...  66.50488 ...  -->   66.50 sex[Female] region[MW] result[mean]

   sex = Female, region = S
   weight |   ...  67.16907 ...  -->   67.17 sex[Female] region[S]  result[mean]

   sex = Female, region = W
   weight |   ...  66.11902 ...  -->   66.11 sex[Female] region[W]  result[mean]
================================       ===========================================
```

Consider the first mean, 78.15. In the collection, it is tagged with `sex[Male]`, `region[NE]`, `result[mean]`. The second mean is tagged with `sex[Male]`, `region[MW]`, `result[mean]`. So one of its tags is the same as the first value—both are tagged `sex[Male]`. The `region` tags differ across the two means—`region[MW]` and `region[NE]`. All the values are tagged with `result[mean]`.

Scanning the "picture", it is clear that each value is tagged with the levels of the `sex` and `region` variables from its `by` group. That seems sensible.

Each tag has two parts—*part1*`[`*part2*`]`. Having two parts lets us group related things using *part1*. Having two parts also lets us refer to all the tags with the same *part1* by just saying the name of *part1* and not having to enumerate all the names in *part2*.

In the collection system, we do not call them "part1" and "part2". We could, but eventually this entry would start to sound like a Dr. Seuss children's book. We call "part1" dimension, and we call "part2" level, or level within dimension, *dimension*`[`*level*`]`.

Every tag always has this two-part structure.

In our collection, we have considered three dimensions—`sex`, `region`, and `result`. Dimension `sex` has two levels—`Male` and `Female`. Dimension `region` has four levels—`NE`, `MW`, `S`, and `W`.

We can specify all levels in the `sex` dimension by typing either `sex[Male] sex[Female]` or just `sex`.

Introducing collect layout

Let's take advantage of referring to groups of tags by just their dimension name and create our first table. The command for laying out tables is `collect layout`, and it wants us to specify what goes on the rows and columns of the table. We computed means across two categorical variables, and `collect` tagged those means with the categories of those variables. Those tags seem like the natural things to put on the rows and columns of our table.

The basic syntax of `collect layout` is

`collect layout (`*row tags*`) (`*column tags*`) (`*table tags*`)`

We will specify all the `sex` tags for the rows and all the `region` tags for the columns. Recalling that the dimension names typed alone represent all the tags in the dimension, we type

```
. collect layout (sex) (region) (result[mean])
Collection: default
      Rows: sex
   Columns: region
    Tables: result[mean]
   Table 1: 2 x 4
```

	NE	MW	S	W
Male	78.15	78.25	77.59	77.99
Female	65.50	66.50	67.17	66.12

The row headers in the table result from enumerating all the tags in dimension `sex`—`Male` and `Female`. The column headers result from enumerating all the tags in the dimension `region`—`NE`, `MW`, `S`, and `W`. Each cell in the table is identified by the intersection of the levels of `sex` and `region` from the cell's row and column headers. So the first cell is identified by `sex[Male]` and `region[NE]`, and it is filled in with the value in the collection that has those two tags (78.15). Continuing down the first column, we see the cell at the bottom left of the table gets its tags from its row and column and is thus `sex[Female]` and `region[NE]`, which is 65.50 from the collection. And so on. That is how `collect layout` fills in a simple table like ours.

The only thing a bit surprising is that we specified something for the *table tags*, `result[mean]`, when we wanted only one table. We have not discussed it yet, but `summarize` stored multiple results, and the `collect` prefix collected all of them. In addition to the means, our collection contains the standard deviation, the minimum, the maximum, and several other results. So we needed to tell `collect layout` which statistic we wanted, and we did that by specifying a table tag. We wanted only one statistic, means, and only one table, so we specified only one tag—`result[mean]`.

We have been telling a little fib about the names of some dimension levels. The `by` variables `sex` and `region` are numeric variables in the dataset, and their values are labeled with the labels we see on the `by` results and in the table we produced—`Male`, `Female`, `NE`, `MW`, `S`, and `W`. To ease in mapping the results of our `by: summarize` command to the tags in the collection, we pretended that the levels of `sex` and `region` were the level labels. In truth, the collection mirrors the dataset. The levels of `sex` are actually numeric—`1` for `Male` and `2` for `Female`. The same is true for the levels of region—`1` for `NE`, `2` for `MW`, `3` for `S`, and `4` for `W`. The collection stores the labels for the levels separately.

We were not fibbing about `mean` in dimension `result`. `mean` really is the name of the level for the means. Dimension levels can be either numeric or string. If the string contains spaces, you must enclose it in quotes wherever it is used.

So to be more truthful, the collection looks more like

```
==========================================
              Collection
==========================================
value            tags
-----   ----------------------------------

 78.15  sex[1] region[1] result[mean]

 78.25  sex[1] region[2] result[mean]

 77.59  sex[1] region[3] result[mean]

 77.99  sex[1] region[4] result[mean]

 65.50  sex[2] region[1] result[mean]

 66.50  sex[2] region[2] result[mean]

 67.17  sex[2] region[3] result[mean]

 66.12  sex[2] region[4] result[mean]
 -----------------------------------------

 dimension  level   label
 ---------  -----  ------
 sex          1     Male
              2     Female

 region       1     NE
              2     MW
              3     S
              4     W
 result     mean    Mean
 =========================================
```

From here on, we will use the actual numeric levels created by `collect` for dimensions `sex` and `region`.

Introducing collect recode

As a sidebar, with a small collection like ours, we could have easily turned our fib into the truth. The command `collect recode` recodes dimension levels from one value to another. Were we to type

```
. collect recode sex 1=Male 2=Female
. collect recode region 1=NE 2=MW 3=S 4=W
```

then everything we said above would be true. And we could use terms like `sex[Female]` rather than `sex[2]` in everything we type below.

Using collect layout

You might be thinking that you can do everything we have done so far with the `table` command, and you are right. In fact, you could have created a collection that is very similar to the one we are working with by typing

```
. table (sex) (region), statistic(mean weight)
```

Let's start doing things that you cannot do with `table` directly.

By the way, the collection that `table` creates is so similar to the one we created with `collect: by:` that you could do everything we do below after either the `table` command above or the `collect: by:` command we started with. The main difference you would see is that `table` computed subtotals for `sex` and `region` and created levels for those totals in the `sex` and `region` dimensions. You can prevent that by adding the option `nototals`.

First, let's transpose our table by swapping where `sex` and `region` appear in the command.

```
. collect layout (region) (sex) (result[mean])

Collection: default
      Rows: region
   Columns: sex
    Tables: result[mean]
   Table 1: 4 x 2

---------------------
   |   Male  Female
---+-----------------
NE |  78.15   65.50
MW |  78.25   66.50
S  |  77.59   67.17
W  |  77.99   66.12
---------------------
```

Wait! You say, "I could have done that with `table` by typing".

```
. table (region) (sex), statistic(mean weight)
```

That is not the same thing. `table` went back through the dataset, recomputed statistics, and then presented them in tabular form. If your dataset had 1 billion observations, that could take some time. We just told `collect layout` to show us the existing collection in a different way.

Let's go on.

Selecting specific levels of a dimension

We have been using dimensions `sex` and `region` to represent all the tags associated with their levels. That implies that we did not need to use all the levels of `sex` and `region` in our layout command. And, indeed, that is true. We could type just a few tags specifically, or even one.

```
. collect layout (region[1] region[3] region[4]) (sex[2]) (result[mean])

Collection: default
      Rows: region[1] region[3] region[4]
   Columns: sex[2]
    Tables: result[mean]
   Table 1: 3 x 1

---------------
   |   Female
---+-----------
NE |    65.50
S  |    67.17
W  |    66.12
---------------
```

We explicitly typed out the list of region tags. There is a shorthand for specifying lists of levels within a dimension—type the list within the brackets. The following would have produced an identical table:

```
. collect layout (region[1 3 4]) (sex[2]) (result[mean])
```

Taken to extremes, collect layout is the way to pull a single value out of a collection.

```
. collect layout (region[3]) (sex[2]) (result[mean])

Collection: default
      Rows: region[3]
   Columns: sex[2]
    Tables: result[mean]
   Table 1: 1 x 1

---------
  | Female
--+-------
S |  67.17
---------
```

What is in my collection?

We have been ignoring that result dimension. Let's rectify that.

Introducing collect levelsof

First, let's list the levels of result.

```
. collect levelsof result

Collection: default
 Dimension: result
    Levels: N Var max mean min sd sum sum_w
```

If you use summarize much, that list of levels may look familiar. Let's use that list to be a little more explicit about what values collect actually collects. It collects everything that is returned by your command in e() or r(). The final summarize from our by command is the last r-class command we have run. Here are the results returned by that summarize.

```
. return list

scalars:
                r(sum) =  90781.40996932983
                r(max) =  134.6100006103516
                r(min) =  36.06000137329102
                 r(sd) =  14.66785984772278
                r(Var) =  215.1461125124382
               r(mean) =  66.11901672930068
              r(sum_w) =  1373
                  r(N) =  1373
```

The names of the r() results returned by summarize are a one-to-one match with the level names in dimension result. They are ordered differently because collect keeps the levels sorted alphabetically (with capitals first). Regardless, the names of the levels are exactly the names of the r() results, with "r()" stripped away. The same would be true if we collected results from a command that returns in e(). Every result is collected, and it is tagged with its r() or e() name. Well, almost every result; we will amend that in collect get, but you will not care.

As we saw earlier, every collected value has multiple tags, but one of them will always be its `result[`*name*`]`, where *name* is taken from its `e()` or `r()` name.

The simple list of levels from `collect levelsof` does not tell us much. We can learn a bit more about the levels by listing their labels.

Introducing collect label list

```
. collect label list result, all

  Collection: default
   Dimension: result
       Label: Result
Level labels:
           N  Number of observations
         Var  Variance
         max  Maximum
        mean  Mean
         min  Minimum
          sd  Std. dev.
         sum  Sum of variable
       sum_w  Sum of the weights
```

Now we are getting somewhere. Those results are everything that was reported on the `summarize` output plus a "`Sum of variable`", "`Sum of the weights`", and a "`Variance`".

Where do result labels come from?

Where did those labels come from? They are system default labels for collections. There is a default label for nearly every result returned in `r()` or `e()` by official commands.

Introducing collect label levels

It is easy for you to change a label. Perhaps you think "`Number of observations`" is too verbose, particularly if you want to make it a column in a table. Let's make it way shorter; lots of folks just go with "`N`".

```
. collect label levels result N "N", modify
```

Maybe we should also shorten the other two long labels.

```
. collect label levels result sum "Sum" sum_w "Sum wts.", modify
```

Introducing collect label save

Later, after you have made lots of label changes, you can save your preferred labels in a file. Type

```
. collect label save mylabels
```

where `mylabels` is whatever filename you prefer. Over time, you may override most of the default system labels.

Introducing collect label use

You can later apply those labels to a collection by typing

```
. collect label use mylabels
```

You do not have to worry if your collection does not contain some of the things you are labeling. The labels exist separately, and there is no harm in labeling things not in your collection. In fact, if those things are later created in your collection because you collect more results, they will get your labels automatically. So you can type `collect label use mylabels` when you first create a collection or right before you create a table; it makes no difference.

Now that we know what other results are tagged by dimension `result`, let's put some of those in a table. One possibility that comes to mind is to remove the shackles of `result[mean]` from our earlier layout command and ask for all levels of `result` as tables.

```
. collect layout (region) (sex) (result)
Collection: default
      Rows: region
   Columns: sex
    Tables: result
   Table 1: 4 x 2
   Table 2: 4 x 2
   Table 3: 4 x 2
   Table 4: 4 x 2
   Table 5: 4 x 2
   Table 6: 4 x 2
   Table 7: 4 x 2
   Table 8: 4 x 2
N
```

	Male	Female
NE	1018.00	1078.00
MW	1310.00	1464.00
S	1332.00	1521.00
W	1255.00	1373.00

Variance

	Male	Female
NE	166.22	198.36
MW	182.29	217.75
S	203.65	230.77
W	187.34	215.15

(output omitted)

Sum wts.

	Male	Female
NE	1018.00	1078.00
MW	1310.00	1464.00
S	1332.00	1521.00
W	1255.00	1373.00

Interactions in collect layout

Well, that was easy to type but not very interesting. For a table like this, it is time to learn about interactions.

First, let's consider our collection for a minute. We chose this particular problem earlier because it was two dimensional, just like many tables. We chose `region` for the rows and `sex` for the columns. The interaction of those two dimensions produces the cells in the table. By "interaction", we mean all combinations of the levels of `region` with the levels of `sex`. In one cell, you must be both male and in the Northeast. In another cell, you must be both female and in the South.

But wait. Our collection does not really have just two dimensions. That was an artifact of our considering only the mean. We have a whole other dimension—`result`. Our results really form a cube—`region X sex X result`. There is a value in every cell of that cube. Now you see the reason we call *part1* of our tags a dimension.

`collect layout` automatically interacts the row and column specifications. For our current example, each row represents a level of dimension `region`, and each column represents a level of `sex`. Each cell results from the interaction of the levels of its row and column. When we added the `result` dimension to create separate tables, each `sex`, `region`, and `result` triad represented one of the cells in one of the tables. Each cell was the result of a three-way interaction.

There is a term for interactions that you place on the rows of tables—"super rows". Likewise, tables can have super columns. If a table has either super rows or super columns, it is representing an underlying three-dimensional set of results. If it has both super rows and super columns, it is representing a four-dimensional set of results. You might have super-super rows or super-super columns. `collect` allows over 20 supers in each of the row, column, and table specifications; so you can represent up to silly-dimensional results.

Adding a super row or a super column is as easy as explicitly interacting two dimensions in the `collect layout` specification. You interact two dimensions by placing a # between them. Let's put our original row and column dimensions both onto the rows.

```
. collect layout (sex#region) (result[mean])

Collection: default
      Rows: sex#region
   Columns: result[mean]
   Table 1: 10 x 1

-------------
       | Mean
-------+-----
Male   |
  NE   | 78.15
  MW   | 78.25
  S    | 77.59
  W    | 77.99
Female |
  NE   | 65.50
  MW   | 66.50
  S    | 67.17
  W    | 66.12
-------------
```

Now the levels of dimension `sex` form super rows and the levels of `region` form rows within `sex`. These are the same results from our very first table, just organized differently.

We moved `result[mean]` to the column specification because there was no longer a reason to specify a tables dimension.

We could have specified a tables dimension and typed

```
. collect layout (sex#region) () (result[mean])
```

Note that an empty `()` is perfectly acceptable. It indicates that there are no tags for the columns.

We could even have pulled the interaction of dimension `result` into the rows specification and not specified any columns or tables.

```
. collect layout (sex#region#result[mean])
```

All of these commands produce a single column of results. Type them and see. The labels change a bit because `collect layout` tries to keep you informed of what you are seeing.

Now we are ready to put our three-dimensional data onto a table. Let's try `result` on the columns of the table.

```
. collect layout (sex#region) (result)
Collection: default
      Rows: sex#region
   Columns: result
   Table 1: 10 x 8
```

	N	Variance	Maximum	Mean	Minimum	Std. dev.	Sum	Sum wts.
Male								
NE	1018.00	166.22	129.84	78.15	47.17	12.89	79559.70	1018.00
MW	1310.00	182.29	139.03	78.25	41.50	13.50	1.0e+05	1310.00
S	1332.00	203.65	158.53	77.59	30.84	14.27	1.0e+05	1332.00
W	1255.00	187.34	175.88	77.99	44.11	13.69	97875.09	1255.00
Female								
NE	1078.00	198.36	148.21	65.50	39.12	14.08	70610.03	1078.00
MW	1464.00	217.75	159.44	66.50	34.93	14.76	97363.14	1464.00
S	1521.00	230.77	138.91	67.17	35.27	15.19	1.0e+05	1521.00
W	1373.00	215.15	134.61	66.12	36.06	14.67	90781.41	1373.00

We hope that is what you were expecting.

Introducing collect style cell

Some of the numbers are oddly formatted, for example, two decimal places on the observation count! This is a good time to admit that we cheated a bit at the outset. We changed the default formatting to get pretty numbers we could talk about. If you have been following along, you were already onto us because your tables showed more decimal places than ours.

Here is what we typed earlier but did not tell you about:

```
. collect style cell result, nformat(%8.2f)
```

Styles control literally everything about how a table looks. Without getting too much into styles right now, what our style command "said" was, "Set the numeric format for all results to be %8.2f." Let's set it back to its system default and redraw our table.

```
. collect style cell result, nformat(%9.0g)
. collect preview
```

	N	Variance	Maximum	Mean	Minimum	Std. dev.	Sum	Sum wts.
Male								
NE	1018	166.221	129.84	78.15295	47.17	12.89267	79559.7	1018
MW	1310	182.2857	139.03	78.24791	41.5	13.50132	102504.8	1310
S	1332	203.6484	158.53	77.5923	30.84	14.27054	103352.9	1332
W	1255	187.3368	175.88	77.98812	44.11	13.6871	97875.09	1255
Female								
NE	1078	198.3562	148.21	65.50096	39.12	14.0839	70610.03	1078
MW	1464	217.7513	159.44	66.50488	34.93	14.7564	97363.14	1464
S	1521	230.7675	138.91	67.16907	35.27	15.19103	102164.2	1521
W	1373	215.1461	134.61	66.11902	36.06	14.66786	90781.41	1373

Introducing collect preview

`collect preview`! That is a new command. We were not changing the layout, so there was no need to specify a new layout. We just asked `collect` to `preview` our existing layout using the style settings currently in effect.

Even so, "preview" seems an odd word. What we see in the Results window is often not our end goal. Often, we are creating a table to be exported to Microsoft Word, HTML, LaTeX, or some other format. Moreover, some of the styles we use cannot be shown in the Results window. So this is just a preview of what you might ultimately obtain when you export your results.

Note that `collect preview` does not display the report about the structure of the table that `collect layout` displays. `collect preview` provides cleaner output—just the table.

With the "new" numeric format, our table shows the numbers we should have been seeing all along.

Reordering columns

Continuing with `collect layout`, you can select the levels of dimension `result` you want, and in any order you want, perhaps,

```
. collect layout (sex#region) (result[mean sd min max N])
Collection: default
      Rows: sex#region
   Columns: result[mean sd min max N]
   Table 1: 10 x 5
```

	Mean	Std. dev.	Minimum	Maximum	N
Male					
NE	78.15295	12.89267	47.17	129.84	1018
MW	78.24791	13.50132	41.5	139.03	1310
S	77.5923	14.27054	30.84	158.53	1332
W	77.98812	13.6871	44.11	175.88	1255
Female					
NE	65.50096	14.0839	39.12	148.21	1078
MW	66.50488	14.7564	34.93	159.44	1464
S	67.16907	15.19103	35.27	138.91	1521
W	66.11902	14.66786	36.06	134.61	1373

Change the order of the levels specified to `collect layout`, and you change the order of the columns on the table.

```
. collect layout (sex#region) (result[N min mean max sd N])
```

You can even repeat levels.

```
. collect layout (sex#region) (result[max max max max max max])
```

(Tabulus maximus?)

Type either command and see.

We could even present just the counts as a frequency cross-tabulation. Feel free to type

```
. collect layout (region) (sex) (result[N])
```

You can also organize the rows and columns differently. You might type any of these layouts or try some of your choosing.

```
. collect layout (sex#result[mean N]) (region)
. collect layout (region#result[mean min max]) (sex)
. collect layout (region#result[mean min max]) (sex)
```

More layout

Our `result` options increase dramatically if we collect `summarize, detail`.

```
. collect clear
. collect: by sex region: summarize weight, detail
```

Let's see what our `result` choices are now.

```
. collect label list result, all

  Collection: default
   Dimension: result
       Label: Result
Level labels:
           N  Number of observations
         Var  Variance
    kurtosis  Kurtosis
         max  Maximum
        mean  Mean
         min  Minimum
          p1  1st percentile
         p10  10th percentile
         p25  25th percentile
          p5  5th percentile
         p50  50th percentile
         p75  75th percentile
         p90  90th percentile
         p95  95th percentile
         p99  99th percentile
          sd  Std. dev.
    skewness  Skewness
         sum  Sum of variable
       sum_w  Sum of the weights
```

We could create a table of whatever percentile distributions interest us, perhaps the quartiles,

```
. collect layout (sex#region) (result[min p25 p50 p75 max])
```

or a finer grain,

```
. collect layout (sex#region) (result[p5 p10 p25 p50 p75 p90 p95])
```

The authors typed that and found that the labels on the percentiles are far too long. So let's shorten them.

```
. collect label levels result p5 "5th" p10 "10th" p25 "25th"
> p50 "50th" p75 "75th" p90 "90th" p95 "95th", modify
. collect preview
```

	5th	10th	25th	50th	75th	90th	95th
Male							
NE	59.42	62.82	69.63	76.89	85.62	95.82	101.61
MW	58.97	62.655	69.17	77.055	85.16	95.2	102.97
S	57.49	60.56	67.19	76.43	85.84	95.03	103.19
W	57.95	62.03	68.49	76.77	85.96	95.03	101.49
Female							
NE	47.51	50.24	55.45	62.88	72.24	84.48	91.74
MW	48.31	50.69	56.59	63.62	73.425	85.39	94.46
S	47.74	50.8	56.36	64.41	75.3	86.98	95.82
W	47.85	50.69	56.25	63.39	72.92	85.96	95.6

We would like to have that %8.2f format back about now.

If you are a fan of third and fourth moments, you could assess and compare all the distributions using skewness and kurtosis.

```
. collect layout (sex#region) (result[mean sd skewness kurtosis])
Collection: default
      Rows: sex#region
   Columns: result[mean sd skewness kurtosis]
   Table 1: 10 x 4
```

	Mean	Std. dev.	Skewness	Kurtosis
Male				
NE	78.15295	12.89267	.5601461	3.705207
MW	78.24791	13.50132	.7798423	4.354643
S	77.5923	14.27054	.6834379	4.384609
W	77.98812	13.6871	.8854262	5.942613
Female				
NE	65.50096	14.0839	1.154802	5.090129
MW	66.50488	14.7564	1.327805	6.098792
S	67.16907	15.19103	1.100521	4.796148
W	66.11902	14.66786	1.231803	5.036233

Introducing collect style autolevels

There is an alternative way to specify the levels on dimension `result` that we used in the last two tables. Instead of specifying them directly in the `collect layout` command, we can preset levels to be used when a dimension name is specified without levels. If you type

```
. collect style autolevels result mean sd skewness kurtosis
```

then whenever `result` appears alone in a `collect layout` command, only levels `mean`, `sd`, `skewness`, and `kurtosis` will be enumerated. We call these levels "automatic levels". It is just as though you typed `result[mean sd skewness kurtosis]`.

So typing

```
. collect style autolevels result mean sd skewness kurtosis
. collect layout (sex#region) (result)
```

produces exactly the same result as

```
. collect layout (sex#region) (result[mean sd skewness kurtosis])
```

Every time you type `collect style autolevels` on the same dimension, it adds whatever levels you type to any existing autolevels for the dimension. So typing

```
. collect style autolevels result p5 p10 p25
. collect style autolevels result p50 p75 p90 p95
```

is equivalent to typing

```
. collect style autolevels result p5 p10 p25 p50 p75 p90 p95
```

Typing

```
. collect style autolevels result, clear
. collect style autolevels result p5 p10 p25 p50 p75 p90 p95
. collect layout (sex#region) (result)
```

produces exactly the same table we created earlier when we typed

```
. collect layout (sex#region) (result[p5 p10 p25 p50 p75 p90 p95])
```

`collect style autolevels` can be particularly convenient when you are exploring several table layouts and you want to use the same `result` levels on all the tables. Or, for that matter, the same levels of any dimension used in the table.

What is in my collection, regression edition

We have already seen one unusual dimension—`result`. The dimensions representing categorical variables, `sex` and `region`, are easy to understand. Anyone who has created a cross-tabulation has used categorical variables as the rows and columns of a table. Dimension `result` was a little bit different. It is just a place where we are keeping related identifiers (levels)—in this case, all the names of results returned in `r()` and `e()`.

We warn you, `collect` uses other unusual dimensions. And it uses a few unusual levels.

Consider the output from a regression.

```
. regress bpsystol age weight i.sex

      Source |       SS           df       MS      Number of obs   =    10,351
-------------+----------------------------------   F(3, 10347)     =   1501.75
       Model |   1709209.9         3  569736.633   Prob > F        =    0.0000
    Residual |  3925460.13    10,347  379.381476   R-squared       =    0.3033
-------------+----------------------------------   Adj R-squared   =    0.3031
       Total |  5634670.03    10,350  544.412563   Root MSE        =    19.478

------------------------------------------------------------------------------
    bpsystol | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
         age |   .6374325   .0111334    57.25   0.000     .6156088    .6592562
      weight |   .4170339    .013474    30.95   0.000     .3906221    .4434456
             |
         sex |
     Female  |   .8244702   .4140342     1.99   0.046     .0128832    1.636057
       _cons |   70.13615   1.187299    59.07   0.000     67.80881    72.46348
------------------------------------------------------------------------------
```

The result levels _r_b, _r_se, . . .

The results are already laid out as a table with the coefficient names on the rows and the coefficient statistics on the columns. Neither the rows nor the columns fit into the dimension and level names we have been using.

Let's consider the columns first—the coefficient statistics. We certainly have an appropriate dimension where we can place these: the `result` dimension. What is tricky is how to name their levels. The coefficients themselves are saved as a row vector named `e(b)`, so we could name their level `b` in `result`, as we have all the other stored results. Spoiler alert, we do not.

The problem is we do not store vectors for the standard error, the t statistic, the p-value, or the confidence interval. These are stored in hidden places or can be derived from other results. You do not care about that; you want to use what you see in the `regress` results in your own tables. So we gave these results special level names—`_r_b` for the regression coefficients, `_r_se` for the standard errors, and so on. Here is the full list of special level names for regression and regressionlike results:

Identifier	Result
`_r_b`	coefficients or transformed coefficients reported by *command*
`_r_se`	standard errors of `_r_b`
`_r_z`	test statistics for `_r_b`
`_r_z_abs`	absolute values of `_r_z`
`_r_df`	degrees of freedom for `_r_b`
`_r_p`	p-values for `_r_b`
`_r_lb`	lower bounds of confidence intervals for `_r_b`
`_r_ub`	upper bounds of confidence intervals for `_r_b`
`_r_ci`	confidence intervals for `_r_b`
`_r_cri`	credible interval (CrI) of Bayesian estimates
`_r_crlb`	lower bound of CrI of Bayesian estimates
`_r_crub`	upper bound of CrI of Bayesian estimates

We admit the `_r_` is a bit much to type and requires explanation. There is a reason for the leading underscore. `collect` will collect all the results from `e()` and `r()` for any official command or from any command written by you or by other users. Those results could have any valid name. By convention, we have told users that anything with a leading underscore is reserved for official names. There is also the precedence of `_b[`*coefname*`]` and _se[*coefname*] being supported in expressions to retrieve coefficients and their standard errors.

As an aside, all the `_r_` names you see above are now supported in expressions. After the regression command above, you could type

```
. display _r_b[age] / _r_se[age]
```

to compute the t statistic by hand and display it.

There is also a reason we chose r. Consider the logistic regression

```
. logistic highbp age weight i.sex
Logistic regression                                     Number of obs =  10,351
                                                        LR chi2(3)    = 2326.44
                                                        Prob > chi2   =  0.0000
Log likelihood = -5887.5446                             Pseudo R2     =  0.1650

------------------------------------------------------------------------------
      highbp | Odds ratio   Std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
         age |   1.052054   .0014852    35.95   0.000     1.049147    1.054969
      weight |   1.044683    .001759    25.96   0.000     1.041242    1.048137
             |
         sex |
     Female  |   1.036659   .0498306     0.75   0.454     .9434528    1.139074
       _cons |    .002525   .0004077   -37.05   0.000     .0018401     .003465
------------------------------------------------------------------------------
Note: _cons estimates baseline odds.
```

The default "coefficients" displayed after logistic are the odds ratios, not the raw coefficients. You can see the raw coefficients instead by adding the option coef. The "r" in _r_b stands for "reported". After our logistic regression, the odds ratios, not the raw coefficients, are collected. In this case, result[_r_b] tags the odds ratios. If we add coef to our command, or even if we replay the results with the option coef,

```
. logistic, coef
```

the raw coefficients are collected. _r_b then stands for the raw coefficient estimates. You can collect whichever transformation you prefer. When transformations are available, whatever you are reporting is what is collected. Type two collect commands if you want to collect both transformed and raw coefficients.

There are quite a few commands that report transformations of their coefficients—incidence rate ratios for poisson, hazard ratios for stcox, standardized coefficients for sem, and several others. Many of these estimators also have panel-data and multilevel commands.

The _r_ results are collected after all regression and regression-like commands. The regression-like commands include mean, proportion, ratio, bayesmh, margins, contrast, and others.

The colname dimension

There is still the issue of what dimension name we should use for the rows of a regression table. They look like variables, so why not variable? Because those rows can contain lots of things that are not variables: for example, the ancillary parameters for variance on many regression commands, parameters on latent variables in sem and gsem, contrasts or expressions in margins, and so on.

collect uses the dimension colname to hold these variable/parameter/estimate tags. There truly is no good meaningful name for all the things this dimension can hold.

There is also a technical reason for using colname. The _r_ results are all related to e(b), and e(b) is a row vector. Let's list e(b) for our logistic regression.

```
. matrix list e(b)
e(b)[1,5]
       highbp:     highbp:     highbp:     highbp:     highbp:
                                   1b.          2.
          age      weight         sex         sex       _cons
y1  .05074447   .04371396           0   .03600346   -5.981495
```

Those labels immediately above the matrix values are the column names for the matrix. All matrices in Stata have row and column names. That way, you can refer to the rows and columns by name as well as by index number. The matrix's column names collectively are called its `colname`. We can use a macro function to display just the column names.

```
. display "`: colname e(b)'"
age weight 1b.sex 2.sex _cons
```

Considering just `e(b)` (`_r_b`), `collect` is really collecting a matrix. To identify a cell in a matrix `collect` not only needs a tag for the whole matrix, `result[_r_b]`, but also needs tags for the specific row and specific column that identify a particular cell. The column tags are placed in dimension `colname` because that is what Stata calls the column names of a matrix. For our logistic model, the `colname` tags associated with all the `_r_` results are `colname[age]`, `colname[weight]`, `colname[1.sex]`, `colname[2.sex]`, and `colname[_cons]`.

If you guessed from the matrix we listed that there would be a `rowname` tag for the `_r_b` "matrix" that we collected, you would be right. That tag is `rowname[y]`. You won't use the `rowname` dimension nearly so often as you will use the `colname` dimension.

Labels on levels of dimension colname

There is something else special about `colname`. We discussed earlier that the levels of the `result` dimension are labeled using a set of system default labels. `collect` can also automatically label most levels of `colname`. That is because most levels of `colname` are variable names. If a variable is labeled, `collect` picks up that label and uses it to label the level. What is more, if a level represents a factor variable, such as `2.sex`, then `collect` labels that level of the factor variable with the appropriate value label from the dataset. It sounds complicated, but it is really just doing what you want. When we type

```
. quietly collect: mean weight, over(sex)

. collect style autolevels result _r_b _r_se _r_ci

. collect layout (colname) (result)

Collection: default
      Rows: colname
   Columns: result
   Table 1: 2 x 3

--------------------------------------------------------------
                      | Coefficient Std. error       95% CI
----------------------+---------------------------------------
Weight (kg) @ Male    |    77.98423   .1945289 77.60292  78.36555
Weight (kg) @ Female  |    66.39418   .1998523 66.00243  66.78593
--------------------------------------------------------------
```

we see "`Male`" and "`Female`" as part of our row headers, not "1" and "2".

Note too that we just used some of the `_r_` levels of result and that we used dimension `colname` too. No need for fanfare. They are just other levels and dimensions that we can use to lay out our tables.

`colname` is not the only dimension that picks up labels from variables. Dimensions `rowname`, `coleq`, `roweq`, `var`, and `across` also fetch variable labels for the levels and factor-variable levels whenever they can.

It turns out the `_r_` levels and the `colname` dimension are not truly unusual. They work just the way any other levels or dimensions work. Their names are just arbitrary.

If you are not liking the row headers in the table above, you can change them. See `collect style row`.

collect layout with regression results

We claimed this subsection was about regression collections, so we should at least create a basic table of regression results from our first regression. First, we type

```
. collect clear
. collect: regress bpsystol age weight i.sex
```

Then, we type

```
. collect style autolevels result _r_b _r_se _r_z _r_p
. collect layout (colname) (result)
Collection: default
      Rows: colname
   Columns: result
   Table 1: 5 x 4
```

	Coefficient	Std. error	t	p-value
Age (years)	.6374325	.0111334	57.25	0.000
Weight (kg)	.4170339	.013474	30.95	0.000
Male	0	0		
Female	.8244702	.4140342	1.99	0.046
Intercept	70.13615	1.187299	59.07	0.000

We used `autolevels` to specify the automatic levels for `result`. That looks a lot like the regression output, except we did not ask for the confidence intervals, there is less column spacing, and this table uses labels rather than variable names on the row headers.

Introducing collect style showbase

There is a lot we could do to make this table prettier, but let's at least get rid of the row for `Male`. `Male` is the base level for the factor variable `i.sex` and we do not need to see its zero coefficient. To turn off displaying base levels for factor variables, we type

```
. collect style showbase off
```

Recall that we do not have to respecify our layout just to see the effect of style changes. We just type

```
. collect preview
```

	Coefficient	Std. error	t	p-value
Age (years)	.6374325	.0111334	57.25	0.000
Weight (kg)	.4170339	.013474	30.95	0.000
Female	.8244702	.4140342	1.99	0.046
Intercept	70.13615	1.187299	59.07	0.000

The base level is gone.

That is all we are going to style on this table. We will have much more to say about styles in section *Let's talk styles*.

At this point, it should come as no surprise that we can transpose the table by swapping the position of `colname` and `result` in our layout.

```
. collect layout (result) (colname)
Collection: default
      Rows: result
   Columns: colname
   Table 1: 4 x 4
```

	Age (years)	Weight (kg)	Sex Female	Intercept
Coefficient	.6374325	.4170339	.8244702	70.13615
Std. error	.0111334	.013474	.4140342	1.187299
t	57.25	30.95	1.99	59.07
p-value	0.000	0.000	0.046	0.000

Let's clear the automatic levels so they does not surprise us later.

```
. collect style autolevels result, clear
```

Okay, it did bite the authors when they were writing this entry, and we do not want you to be surprised in the same way. It is pretty easy to convince yourself that collections are broken when you have an `autolevels` set that is at odds with levels you are trying to report.

Tables of model statistics

Before we leave this simple regression, let's look at one more thing. You may think that the regression coefficients are the only "tabular" results we have collected. But there is another set of results lurking in our collection, the model-level statistics. They are all about this one model, so collectively they are a set of one-dimensional results. Even so, a one-dimensional table is still a table.

We can also tell that the model statistics have been collected by listing the labels of dimension `result`.

```
. collect label list result

  Collection: default
   Dimension: result
       Label: Result
Level labels:
           F  F statistic
           N  Number of observations
        _r_b  Coefficient
       _r_ci  __LEVEL__% CI
       _r_df  df
       _r_lb  __LEVEL__% lower bound
        _r_p  p-value
       _r_se  Std. error
       _r_ub  __LEVEL__% upper bound
        _r_z  t
    _r_z_abs  |t|
        beta  Standardized coefficient
         cmd  Command
     cmdline  Command line as typed
      depvar  Dependent variable
        df_m  Model DF
        df_r  Residual DF
   estat_cmd  Program used to implement estat
          ll  Log likelihood
        ll_0  Log likelihood, constant-only model
   marginsok  Predictions allowed by margins
       model  Model
         mss  Model sum of squares
     predict  Program used to implement predict
  properties  Command properties
          r2  R-squared
        r2_a  Adjusted R-squared
        rank  Rank of VCE
        rmse  RMSE
         rss  Residual sum of squares
       sum_w  Sum of the weights
       title  Title of output
         vce  SE method
```

It takes a bit of scanning, but about midway down we see the `Model DF`, the `Residual DF`, and the `Log likelihood`. A bit farther down, we see the `R-squared`, the `Adjusted R-squared`, and the `RMSE`.

Do not be distracted by the `__LEVEL__%`; that is just the way labels obtain the confidence level that can be specified using the `level()` option of regression commands.

Previously, we pulled out the coefficient statistics by interacting dimensions `result` and `colname`. How do we ask for just model-level results? They are a one-way table (listing) of results, so we do not need to specify anything for our columns. We just ask for dimension `result` on the rows.

```
. collect layout (result)

Collection: default
      Rows: result
   Table 1: 23 x 1
```

F statistic	1501.751
Number of observations	10351
Command	regress
Command line as typed	regress bpsystol age weight i.sex
Dependent variable	bpsystol
Model DF	3
Residual DF	10347
Program used to implement estat	regress_estat
Log likelihood	-45420.36
Log likelihood, constant-only model	-47291.07
Predictions allowed by margins	XB default
Model	ols
Model sum of squares	1709210
Program used to implement predict	regres_p
Command properties	b V
R-squared	.3033381
Adjusted R-squared	.3031361
Rank of VCE	4
RMSE	19.47772
Residual sum of squares	3925460
Sum of the weights	10351
Title of output	Linear regression
SE method	ols

Well, we certainly have our model statistics, but we have a lot of other "junk" too—the `Dependent variable`, a flag for `Predictions allowed by margins`, the `Rank of VCE`, and even the `Program used to implement predict` and the `Command line as typed`. We are going to have to be specific with `collect layout` about the levels of `result` we want.

```
. collect layout (result[N r2 rmse df_m df_r F])

Collection: default
      Rows: result[N r2 rmse df_m df_r F]
   Table 1: 6 x 1
```

Number of observations	10351
R-squared	.3033381
RMSE	19.47772
Model DF	3
Residual DF	10347
F statistic	1501.751

In explaining how we ask for the model statistics compared with how we ask for the coefficient statistics, we said, "They are a one-way table (listing) of results, so we do not need to specify anything for our columns." That is true, but it is also a pretty fast explanation. If it seems logical to you, you are good to go. If you would like to understand more fully why it is true, see section *How collect layout processes tag specifications* in [TABLES] **Collection principles**.

What is in my collection, multiple-equation models (dimension coleq)

Another "unusual" dimension that is useful for multivariate models is `coleq`. Let's collect the results from a simple multivariate regression.

```
. collect clear
. collect:  mvreg bpsystol bpdiast = age weight
Equation             Obs   Parms        RMSE    "R-sq"          F        P>F
------------------------------------------------------------------------------
bpsystol          10,351       3    19.48051    0.3031       2250     0.0000
bpdiast           10,351       3    11.51474    0.2067   1348.469     0.0000

------------------------------------------------------------------------------
             | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
bpsystol     |
         age |   .6379892   .0111315    57.31   0.000     .6161692    .6598091
      weight |   .4069041   .0124786    32.61   0.000     .3824435    .4313646
       _cons |   71.27096   1.041742    68.42   0.000     69.22894    73.31297
-------------+----------------------------------------------------------------
bpdiast      |
         age |    .187733   .0065797    28.53   0.000     .1748355    .2006306
      weight |   .3116502    .007376    42.25   0.000     .2971918    .3261086
       _cons |   50.37585    .615764    81.81   0.000     49.16884    51.58287
------------------------------------------------------------------------------
```

What is new about this regression is that it has multiple equations—one for `bpsystol` and one for `bpdiast`. It is sensible to tag each equation in the model and to put those tags into a dimension where they can be referenced together. That is just what `collect` does.

What does it name that dimension? Let's look at the `e(b)` matrix again.

```
. matrix list e(b)
e(b)[1,6]
     bpsystol:   bpsystol:   bpsystol:    bpdiast:    bpdiast:    bpdiast:
          age      weight       _cons         age      weight       _cons
y1  .63798917   .40690407   71.270956   .18773302   .31165024   50.375852
```

We see that there are `colnames` on this matrix, as there were on the simple regression. But we also see `bpsystol:` and `bpdiast:` above the `colnames`. Those are the dependent variables of our equation, and they also label the columns of the matrix. Collectively, we call `bpsystol` and `bpdiast` the matrix's `coleqs`, and there are matrix commands for setting and fetching the `coleq`. So `coleq` is the name `collect` gives to the dimension that holds the tags for the equations. In our model, the levels of those tags are the dependent variable names—`bpsystol` and `bpdiast`. Let's confirm

```
. collect label list coleq, all
  Collection: default
   Dimension: coleq
       Label: Depvars, parameters, and column equations
Level labels:
     bpdiast  Diastolic blood pressure
    bpsystol  Systolic blood pressure
```

Indeed `coleq` is a dimension. It has its own nice, long label—`Depvars, parameters, and column equations`. Its levels are indeed the dependent variable names from our multivariate regression—`bpdiast` and `bpsystol`. And those dimensions have their own nice, long labels—`Diastolic blood pressure` and `Systolic blood pressure`.

`collect label list` can tell us a lot about what is in a dimension, how we might use it in a layout, and whether we are likely to want to change its labels for our table.

We clearly cannot use our univariate regression layout specification.

```
. collect layout (colname) (result)
```

Every cell in that table would have two values, one for the `bpdiast` dependent variable and one for the `bpsystol` dependent variable. That specification does not uniquely identify the cells in the table. We need to add dimension `coleq`. Let's try it in the tables specification first.

```
. collect style autolevels result _r_b _r_ci _r_se _r_z _r_p
. collect layout (colname) (result) (coleq)
Collection: default
      Rows: colname
   Columns: result
    Tables: coleq
   Table 1: 3 x 5
   Table 2: 3 x 5
Systolic blood pressure
```

	Coefficient	95% CI		Std. error	t	p-value
Age (years)	.6379892	.6161692	.6598091	.0111315	57.31	0.000
Weight (kg)	.4069041	.3824435	.4313646	.0124786	32.61	0.000
Intercept	71.27096	69.22894	73.31297	1.041742	68.42	0.000

```
Diastolic blood pressure
```

	Coefficient	95% CI		Std. error	t	p-value
Age (years)	.187733	.1748355	.2006306	.0065797	28.53	0.000
Weight (kg)	.3116502	.2971918	.3261086	.007376	42.25	0.000
Intercept	50.37585	49.16884	51.58287	.615764	81.81	0.000

We have presented our regression results in two tables.

That is not the best arrangement if we want to compare across the two regressions. Let's shuffle the equations onto the columns and put both the `colnames` and the `result` dimensions on the rows.

```
. collect layout (colname#result) (coleq)

Collection: default
      Rows: colname#result
   Columns: coleq
   Table 1: 18 x 2

--------------------------------------------------------------------
               |  Systolic blood pressure Diastolic blood pressure
---------------+----------------------------------------------------
Age (years)    |
  Coefficient  |                 .6379892                  .187733
  95% CI       |       .6161692  .6598091        .1748355  .2006306
  Std. error   |                 .0111315                 .0065797
  t            |                    57.31                    28.53
  p-value      |                    0.000                    0.000
Weight (kg)    |
  Coefficient  |                 .4069041                 .3116502
  95% CI       |       .3824435  .4313646        .2971918  .3261086
  Std. error   |                 .0124786                  .007376
  t            |                    32.61                    42.25
  p-value      |                    0.000                    0.000
Intercept      |
  Coefficient  |                 71.27096                 50.37585
  95% CI       |       69.22894  73.31297        49.16884  51.58287
  Std. error   |                 1.041742                  .615764
  t            |                    68.42                    81.81
  p-value      |                    0.000                    0.000
--------------------------------------------------------------------
```

Now it is easy to compare the regression coefficients and their statistics across dependent variables. Again, there is a lot we could do to make this table prettier. The justification makes the CIs jut out. As we predicted, the labels on `bpsystol` and `bpdiast` are too long for column headers. There are too many digits in the results. And more. We will address those types of concerns in *Let's talk styles*.

What is in my collection, collecting results from multiple commands (dimension cmdset)

We have been collecting results from a single command. It is just as easy to collect and tabulate results from several commands.

Let's collect results from two regressions.

```
. collect clear
. collect: regress bpsystol age weight
. collect: regress bpsystol age weight i.hlthstat
```

In the second regression, we added a factor variable that records self-reported health status.

With two regressions in our collection, we have two coefficients for `age` and `weight`. We have two of every statistic associated with those coefficients. That is painfully obvious, but important when specifying a layout. Because we have two of nearly everything, we need another dimension to tell the coefficients in the regression apart.

If only we had a dimension that identified the specific commands from which we collected results. We do, dimension cmdset. Let's look at its levels.

```
. collect label list cmdset, all

  Collection: default
   Dimension: cmdset
       Label: Command results index
Level labels:
           1
           2
```

Well, that is minimalist. The levels are 1 and 2 and they are unlabeled. Regardless, cmdset is a counter (or index) for each command from which we collected results. That is enough. Let's put that on the columns and put both the colname and result dimensions on the rows. To keep things short, let's just show the coefficients and their standard errors.

```
. collect style autolevels result _r_b _r_se

. collect layout (colname#result) (cmdset)

Collection: default
      Rows: colname#result
   Columns: cmdset
   Table 1: 24 x 2

------------------------------------
              |         1          2
--------------+---------------------
Age (years)   |
  Coefficient |  .6379892   .6071483
  Std. error  |  .0111315   .0119737
Weight (kg)   |
  Coefficient |  .4069041   .4039598
  Std. error  |  .0124786    .012471
Excellent     |
  Coefficient |                    0
  Std. error  |                    0
Very good     |
  Coefficient |              .715111
  Std. error  |             .5519263
Good          |
  Coefficient |             2.233169
  Std. error  |             .5453581
Fair          |
  Coefficient |             4.133798
  Std. error  |             .6492333
Poor          |
  Coefficient |             3.549244
  Std. error  |             .8558511
Intercept     |
  Coefficient |  71.27096   71.22963
  Std. error  |  1.041742   1.073791
------------------------------------
```

And we need not stop there. We can add the results of as many commands as we like to a collection. Let's add a third regression with one more covariate.

```
. collect: regress bpsystol age weight i.hlthstat i.sex
```

To see those results on our table, we do not have to respecify our layout. We still want the commands on the columns. We have just added one more command. All we need to do is repreview the table.

```
. collect preview
```

	1	2	3
Age (years)			
Coefficient	.6379892	.6071483	.6070032
Std. error	.0111315	.0119737	.011973
Weight (kg)			
Coefficient	.4069041	.4039598	.4122565
Std. error	.0124786	.012471	.0134793
Excellent			
Coefficient		0	0
Std. error		0	0
Very good			
Coefficient		.715111	.6759903
Std. error		.5519263	.5524101
Good			
Coefficient		2.233169	2.184542
Std. error		.5453581	.5461395
Fair			
Coefficient		4.133798	4.062105
Std. error		.6492333	.6506867
Poor			
Coefficient		3.549244	3.537842
Std. error		.8558511	.8558125
Male			
Coefficient			0
Std. error			0
Female			
Coefficient			.6725152
Std. error			.4148375
Intercept			
Coefficient	71.27096	71.22963	70.32292
Std. error	1.041742	1.073791	1.210646

Just what we expected.

We could make this table prettier; see section *Let’s talk styles*.

Let's at least get rid of the base levels of the factor variables and make the column headers a bit more informative.

```
. collect style showbase off
. collect label levels cmdset 1 "Base" 2 "Partial" 3 "Full"
. collect preview
```

	Base	Partial	Full
Age (years)			
Coefficient	.6379892	.6071483	.6070032
Std. error	.0111315	.0119737	.011973
Weight (kg)			
Coefficient	.4069041	.4039598	.4122565
Std. error	.0124786	.012471	.0134793
Very good			
Coefficient		.715111	.6759903
Std. error		.5519263	.5524101
Good			
Coefficient		2.233169	2.184542
Std. error		.5453581	.5461395
Fair			
Coefficient		4.133798	4.062105
Std. error		.6492333	.6506867
Poor			
Coefficient		3.549244	3.537842
Std. error		.8558511	.8558125
Female			
Coefficient			.6725152
Std. error			.4148375
Intercept			
Coefficient	71.27096	71.22963	70.32292
Std. error	1.041742	1.073791	1.210646

You cannot only collect from multiple commands but also collect from multiple sets of related commands. In the current example, we could have collected results from `test` commands for the additional covariates in the `Partial` and `Full` models. Or we could have collected the results of `lrtest` for the same purpose. Or we could have collected the results of `margins` commands that might have estimated the effect of dropping weight by 10%. Any or all of these results could have been collected and added below the coefficients in the table above. For an example, see [TABLES] **Example 6**.

Seeing what is my collection

We have been pulling dimension names out of thin air and using them. Let's do more. You can ask your collection about its dimensions at any time.

Introducing collect dims

```
. collect dims

Collection dimensions
Collection: default
─────────────────────────────────────────
                     Dimension   No. levels
─────────────────────────────────────────
Layout, style, header, label
                        cmdset   3
                         coleq   1
                       colname   10
             colname_remainder   1
                      hlthstat   5
                 program_class   1
                        result   33
                   result_type   3
                       rowname   1
                           sex   2
Style only
                  border_block   4
                     cell_type   4
─────────────────────────────────────────
```

We read from the output that the current collection is the `default` collection. And we see a list of dimensions in groups.

Header `Layout, style, header, label` is telling you that you can do anything in the collection system with the dimensions in that group. You can lay out tables using `collect layout`. You can set cell styles on specific dimensions and levels using `collect style cell`. (Cell styles are all the styles for how things look—bolding, numeric formats, color, etc.) You can set whether the headers show labels, names or nothing for dimensions, or levels of dimensions, using `collect style header`. You can set the content of the labels used in the row and column headers using `collect label`.

The second grouping reads `Style only`. The only thing you can do with these dimensions and their levels is set cell styles.

A third grouping, not shown here but appearing between the above two, reads `Header, label`. You can do only two things with the dimensions in this group. You can set whether labels or names are shown in the headers, and you can change the content of the labels used in the headers. This group is populated by factor variables found in dimension `coleq`, `roweq`, or `rowname` but not `colname` or `var`.

It is not a syntax error to use any of these dimensions on one of the commands that are not in its usage group. Style and label commands are always allowed so long as their syntax is legal. The dimensions and levels that they reference do not need to exist in the current collection.

Let's return to the output of `collect dims`. In the first grouping of dimensions, we immediately recognize `cmdset`, `colname`, `coleq`, and `result`. They need no further explanation. That leaves three dimensions in the first group that we do not recognize—`colname_remainder`, `program_class`, and `result_type`. Let's list their levels and labels to search for clues.

First, colname_remainder,

```
. collect label list colname_remainder, all
  Collection: default
   Dimension: colname_remainder
       Label: Covariate names with factors removed
Level labels:
       _cons
```

colname_remainder is not interesting in this example. This dimension is created when collect augments the tags on a result with the factor variables from dimensions colname and var already in the tag. colname_remainder is the remaining (nonfactor) elements of interactions or _cons when the colname level is a single factor variable. This dimension might be necessary to help uniquely match items when you specify factor variables directly in collect layout instead of using them as levels within dimension colname or var.

Second, program_class,

```
. collect label list program_class, all
  Collection: default
   Dimension: program_class
       Label: Result program class
Level labels:
      eclass
```

Well, that could not be more boring. The single, unlabeled level is eclass. We collected results from two commands, two regress commands, and regress returns only results in e(). Results returned in e() are called e-class results, ergo, eclass. Had we also collected results from summarize, or even margins, then we would see a second level here—rclass.

We cannot think of a reason to use dimension program_class in the collect system. You could set the background to red for results returned by e-class commands and set the background to blue for results returned by r-class commands. We do not know why you would, but you could. Perhaps you are writing Stata documentation and want to emphasize where the results came from.

Third, result_type,

```
. collect label list result_type, all
  Collection: default
   Dimension: result_type
       Label: Result type
Level labels:
       macro  Macro
      matrix  Matrix
      scalar  Scalar
```

The levels are macro, matrix, and scalar. Those are the types of results that can be returned in e() or r(). Again, not something you would use often in specifying a layout or styling cells. But you could. If you added the interaction #result_type[scalar] to any term in the row, column, or table specification in collect layout, you would limit the table to include only scalar results.

Factor variables in regressions and other commands

In the first group, we see the two dimensions, hlthstat and sex. Those are the two factor variables from our regressions. collect creates dimensions for factor variables from regressions and from other commands that accept factor variables in the *varlist*.

These dimensions are similar to the dimensions that are named after the by variables in our very first example in this entry. All of these dimensions can be used to specify rows and columns in collect layout. Even dimensions hlthstat and sex can be used; however, they must be specified as i.hlthstat and i.sex when used in the colname dimension.

One way to tag regression results is colname[i.hlthstat]. So we do get a table by typing

```
. collect layout (colname[i.hlthstat]#result) (cmdset)

Collection: default
      Rows: colname[i.hlthstat]#result
   Columns: cmdset
   Table 1: 12 x 2

-----------------------------------
              |   Partial      Full
--------------+--------------------
Very good     |
  Coefficient |   .715111  .6759903
  Std. error  |  .5519263  .5524101
Good          |
  Coefficient |  2.233169  2.184542
  Std. error  |  .5453581  .5461395
Fair          |
  Coefficient |  4.133798  4.062105
  Std. error  |  .6492333  .6506867
Poor          |
  Coefficient |  3.549244  3.537842
  Std. error  |  .8558511  .8558125
-----------------------------------
```

We have selected just the i.hlthstat levels of dimension colname. Note that the "Base" column is no longer in the table. Factor variable i.hlthstat was not in the base regression, so there is no "Base" column to report when the table is limited to colname[i.hlthstat].

We can even limit the table to just some of the levels of the factor variable i.hlthstat. To do that, we use standard factor-variable notation.

```
. collect layout (colname[2.hlthstat 4.hlthstat]#result) (cmdset)

Collection: default
      Rows: colname[2.hlthstat 4.hlthstat]#result
   Columns: cmdset
   Table 1: 6 x 2

-----------------------------------
              |   Partial      Full
--------------+--------------------
Very good     |
  Coefficient |   .715111  .6759903
  Std. error  |  .5519263  .5524101
Fair          |
  Coefficient |  4.133798  4.062105
  Std. error  |  .6492333  .6506867
-----------------------------------
```

You can use full factor-variable notation, so typing

```
. collect layout (colname[i(2 4).hlthstat]#result) (cmdset)
```

would produce the same table.

Another thing we can do with dimensions `hlthstat` and `sex` is change their labels and the labels on their levels. Let's relabel the 4th level of `hlthstat`, and then repreview our most recent table.

```
. collect label levels hlthstat 4 "Between Very good and Poor", modify
. collect preview
```

```
─────────────────────────────────────────────
                           │  Partial     Full
───────────────────────────┼─────────────────
Very good                  │
  Coefficient              │  .715111 .6759903
  Std. error               │ .5519263 .5524101
Between Very good and Poor │
  Coefficient              │ 4.133798 4.062105
  Std. error               │ .6492333 .6506867
─────────────────────────────────────────────
```

That leaves the two dimensions in the `Style only` group of `collect dims`—`border_block` and `cell_type`. These dimensions are for advanced use, but let's list the levels and labels for `cell_type` anyway.

```
. collect label list cell_type, all
   Collection: default
    Dimension: cell_type
        Label: Table cell type
 Level labels:
column-header
       corner
         item
   row-header
```

The levels `row-header`, `column-header`, `item`, and `corner` are referring to the cells in the four parts of a table—the cells in the row headers, the cells in the column headers, the `item` cells in the body of the table, and the no mans land of the upper left corner. When you type

```
. collect style cell cell_type[row-header], shading(background(blue))
```

you are changing the background color of all the cells in the row-header region to blue.

See [TABLES] **Example 4** for an example using dimension `cell_type`.

Surprisingly, the levels of dimension `border_block` are exactly the same as the levels of `cell_type`. Whereas dimension `cell_type` refers to the cells in the table regions, dimension `border_block` refers to the entire block of the region.

Special dimensions created by table

We have covered the most important special dimensions that can be created when you collect results. There may be others if your collection was created by `table`. The nomenclature is familiar now, so let's cover these dimensions quickly. Not because they are unimportant but because you are now ready to drink from the fire hose. Our examples will be terse and intended solely to demonstrate features, not to be interesting or meaningful.

The `table` command is built on top of the collection system. The `table` command builds a collection to hold all the results you request, customizes some styles, creates a layout, and then previews the table.

`table` names the collection it creates `Table`. If you run another `table` command, the collection `Table` is replaced with the collection created by the new `table` command. Collection `Table`, when it exists, always contains the collection for the most recent `table` command.

Dimension variables

We mentioned much earlier that there is not much difference in the collection created by a command like `collect: by region: summarize` ... and a command like `table region` .... Both create a dimension named `region`, and its levels are the distinct values that the variable `region` takes on in the dataset. We discussed this type of dimension at length in *Tags, dimensions, and levels* through *Interactions in collect layout* and will say no more here.

Variables from statistic() option—dimension var

When you specify statistics using the `statistic()` option of `table`, `table` creates a dimension named `var` whose levels are the names of the variables for which statistics were computed. Take the simple table,

```
. table region, statistic(mean age lead) statistic(sd age lead)

-------------------------------------------------------------------------
        |               Mean                    Standard deviation
        |  Age (years)   Lead (mcg/dL)    Age (years)   Lead (mcg/dL)
--------+----------------------------------------------------------------
Region  |
  NE    |     47.81584        14.83784       17.01692        5.782612
  MW    |     46.52776        14.78544       17.37627        6.698146
  S     |     48.19068        13.29985       16.86443        6.200866
  W     |     47.83828        14.52686       17.53498        5.704972
  Total |     47.57965        14.32033       17.21483        6.166468
-------------------------------------------------------------------------
```

We can learn more about this table by typing `collect layout`:

```
. collect layout

Collection: Table
      Rows: region
   Columns: result#var
   Table 1: 6 x 4

-------------------------------------------------------------------------
        |               Mean                    Standard deviation
        |  Age (years)   Lead (mcg/dL)    Age (years)   Lead (mcg/dL)
--------+----------------------------------------------------------------
Region  |
  NE    |     47.81584        14.83784       17.01692        5.782612
  MW    |     46.52776        14.78544       17.37627        6.698146
  S     |     48.19068        13.29985       16.86443        6.200866
  W     |     47.83828        14.52686       17.53498        5.704972
  Total |     47.57965        14.32033       17.21483        6.166468
-------------------------------------------------------------------------
```

When specified without arguments, `collect layout` redisplays the most recent table it created, and yes, `table` used `collect layout` to create its table. Let's focus on the header that we have heretofore ignored. It tells us what the row specification was—`region`. And it tells us what the column specification was—`result#var`. Knowing those specifications can be truly convenient. If we want to rearrange the table rows and columns, we know which dimensions to use.

Dimension var is the new player in that specification. Let's look at var a little more closely.

```
. collect label list var

  Collection: Table
   Dimension: var
       Label: Statistic option variable
Level labels:
         age  Age (years)
        lead  Lead (mcg/dL)
```

We see levels age and lead. Those are the names of the variables we specified in the statistic() option. Dimension var looks a lot like the dimension colname, which we saw when collecting regression results. Great we know how to use dimensions like that. Let's shuffle our table so that the means and standard deviations are near each other.

```
. collect layout (var#result) (region)

Collection: Table
      Rows: var#result
   Columns: region
   Table 1: 6 x 5
```

	Region				
	NE	MW	S	W	Total
Age (years)					
Mean	47.81584	46.52776	48.19068	47.83828	47.57965
Standard deviation	17.01692	17.37627	16.86443	17.53498	17.21483
Lead (mcg/dL)					
Mean	14.83784	14.78544	13.29985	14.52686	14.32033
Standard deviation	5.782612	6.698146	6.200866	5.704972	6.166468

Dimension colname and matching to regressions

We said that dimension var looked a lot like dimension colname. In fact, they serve exactly the same purpose. So much so that table also creates dimension colname, which is identical to dimension var. This can be useful if you are trying to put results from table on the same rows or columns as results from regressions or regressionlike commands. Recall that collect puts covariate names into dimension colname.

Here is a silly example using colname to align the results from table and regress.

First, we type the table command.

```
. table, statistic(mean age lead) statistic(sd age lead)
```

Mean	
Age (years)	47.57965
Lead (mcg/dL)	14.32033
Standard deviation	
Age (years)	17.21483
Lead (mcg/dL)	6.166468

Then, we add our regression results to the `table` results.

```
. collect, name(Table): regress bpsystol age lead

      Source |       SS           df       MS      Number of obs   =     4,948
-------------+----------------------------------   F(2, 4945)      =    775.82
       Model |  640033.944         2  320016.972   Prob > F        =    0.0000
    Residual |  2039746.25     4,945  412.486602   R-squared       =    0.2388
-------------+----------------------------------   Adj R-squared   =    0.2385
       Total |  2679780.19     4,947  541.698038   Root MSE        =     20.31

------------------------------------------------------------------------------
    bpsystol | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
         age |   .6517974   .0168645    38.65   0.000     .6187355    .6848593
        lead |   .2680019   .0468828     5.72   0.000     .1760907     .359913
       _cons |    96.0544   1.057516    90.83   0.000      93.9812     98.1276
------------------------------------------------------------------------------
```

Note that we used the `collect` option `name()`, which we used to place our results into collection `Table`—the collection produced by the `table` command.

Behind the scenes, `table` sets the automatic levels of results to be only the results you have specified on the `table` command or what `table` thinks are sensible results to show if you have included a `command()` option. We need to add the regression results we wanted displayed to the automatic levels. Let's add coefficients and their standard errors.

```
. collect style autolevels result _r_b _r_se
```

All that is left is to specify how we want our table to look.

```
. collect layout (colname) (result)

Collection: Table
      Rows: colname
   Columns: result
   Table 1: 3 x 4

--------------------------------------------------------------------------
              |      Mean   Standard deviation   Coefficient   Std. error
--------------+-----------------------------------------------------------
Age (years)   |  47.57965             17.21483      .6517974     .0168645
Lead (mcg/dL) |  14.32033             6.166468      .2680019     .0468828
Intercept     |                                     96.0544     1.057516
--------------------------------------------------------------------------
```

We have both our `table` and `regress` results in one table.

We could organize the table as one column.

```
. collect layout (colname#result)

Collection: Table
      Rows: colname#result
   Table 1: 13 x 1

---------------------------------
Age (years)           |
  Mean                |  47.57965
  Standard deviation  |  17.21483
  Coefficient         |  .6517974
  Std. error          |  .0168645
Lead (mcg/dL)         |
  Mean                |  14.32033
  Standard deviation  |  6.166468
  Coefficient         |  .2680019
  Std. error          |  .0468828
Intercept             |
  Coefficient         |   96.0544
  Std. error          |  1.057516
---------------------------------
```

Why would we want the results in one column? Perhaps we would like to compare the results across groups.

If we just add the `region` variable as the row specification to our `table` command, we will compute the means by the levels of `region`.

```
. table region, statistic(mean age lead) statistic(sd age lead) nototal
```

If we insert `by region:` into the command that collects regression results, the regression results will also be computed by the levels of `region`.

```
. collect, name(Table): by region, sort: regress bpsystol age lead
```

We still need to add to the automatic levels.

```
. collect style autolevels result _r_b _r_se
```

All that is left is to add dimension `region` as our column specification.

```
. collect layout (colname#result) (region)

Collection: Table
      Rows: colname#result
   Columns: region
   Table 1: 13 x 4
```

	Region NE	MW	S	W
Age (years)				
Mean	47.81584	46.52776	48.19068	47.83828
Standard deviation	17.01692	17.37627	16.86443	17.53498
Coefficient	.6819023	.6143461	.6761958	.6459431
Std. error	.0390149	.0305406	.034739	.0319899
Lead (mcg/dL)				
Mean	14.83784	14.78544	13.29985	14.52686
Standard deviation	5.782612	6.698146	6.200866	5.704972
Coefficient	.3411097	.26796	.3455647	.1104092
Std. error	.1148679	.0796156	.0934401	.0969654
Intercept				
Coefficient	93.83657	97.91132	93.83902	98.2411
Std. error	2.50846	1.837282	2.150045	2.09812

Index of command() options—dimension command

The `table` command itself can collect results from multiple commands. Here is an example of two nested regressions.

```
. table, command(regress bpsystol age lead)
>        command(regress bpsystol age lead weight)
```

regress bpsystol age lead	
Age (years)	.6517974
Lead (mcg/dL)	.2680019
Intercept	96.0544
regress bpsystol age lead weight	
Age (years)	.6373174
Lead (mcg/dL)	.1183383
Weight (kg)	.3998766
Intercept	70.08091

Clearly, `table` is keeping track of the commands we typed; the full commands are shown right there on the table. The commands are the super rows, and the regression coefficients from the `result` dimension are the rows. `table` creates the dimension `command` and uses it to hold a level for each `command()` option.

```
. collect label list command

  Collection: Table
   Dimension: command
       Label: Command option index
Level labels:
           1  regress bpsystol age lead
           2  regress bpsystol age lead weight
```

We can put the commands on the columns for a more conventional regression comparison table.

```
. collect layout (colname#result) (command)

Collection: Table
      Rows: colname#result
   Columns: command
   Table 1: 4 x 2
```

	regress bpsystol age lead	regress bpsystol age lead weight
Age (years)	.6517974	.6373174
Lead (mcg/dL)	.2680019	.1183383
Weight (kg)		.3998766
Intercept	96.0544	70.08091

We should clearly shorten the labels on the levels of `command` using the `collect label levels` command. We might also want to add the standard errors of the coefficients or other coefficient statistics using `collect style autolevels result`. We leave that as an exercise.

Index of command() and statistic() options—dimension statcmd

What if our `table` command has both `command()` and `statistic()` options?

```
. table region, statistic(mean age lead) statistic(sd age lead) ///
    command(regress bpsystol age lead) nototal
```

We are not going to show the output from that command because it would wrap on this page. Let's instead see how the table was laid out.

```
. collect layout

Collection: Table
      Rows: region
   Columns: statcmd#result#colname
   Table 1: 5 x 7
```

(output omitted)

We again omit the table from the output because it would wrap. Let's focus on the header. The only dimension we do not recognize is `statcmd` in the `Columns:` listing. Let's look at `statcmd`.

```
. collect label list statcmd

  Collection: Table
   Dimension: statcmd
       Label: Statistic/command option index
Level labels:
           1  Mean
           2  Standard deviation
           3  regress bpsystol age lead
```

So each level of `statcmd` represents one of our `statistic()` or `command()` option. Let's transpose our row and column specifications so we can finally see a table.

```
. collect layout (statcmd#result#colname) (region)
Collection: Table
      Rows: statcmd#result#colname
   Columns: region
   Table 1: 13 x 4
```

	Region NE	MW	S	W
Mean				
Mean				
Age (years)	47.81584	46.52776	48.19068	47.83828
Lead (mcg/dL)	14.83784	14.78544	13.29985	14.52686
Standard deviation				
Standard deviation				
Age (years)	17.01692	17.37627	16.86443	17.53498
Lead (mcg/dL)	5.782612	6.698146	6.200866	5.704972
regress bpsystol age lead				
Coefficient				
Age (years)	.6819023	.6143461	.6761958	.6459431
Lead (mcg/dL)	.3411097	.26796	.3455647	.1104092
Intercept	93.83657	97.91132	93.83902	98.2411

Other dimensions

One other dimension that `table` sometimes creates automatically is `across()`. That dimension holds all the combinations of any `across()` options that are specified to determine over which groups percentages and proportions are computed. You will not use this dimension often.

`table` also creates any dimensions that `collect` would create for any commands that appear in `command()` options. Which is to say, any of the dimensions we have discussed in this entry and more. We already saw such dimensions when we included `command(regress ...)` in some of our examples above.

Let's talk styles

Overview

Styles affect how almost everything on your table looks, is organized, or composed. Even so, we are not going to categorize all the styles or even discuss what you can do with styles. That is done in the individual style entries. This entry is about concepts and how you use those concepts. For a categorization of styles with links to their entries, go to [TABLES] **Intro 4** and see these sections:

Change styles—formats, bolding, colors, and more

Control display of zero coefficients in regression results

Modify labels in row and column headers

There is a bit of labeling in that last section, but it also links to styles. In row and column headers, both content and format matter.

Basic targeting

What is common to all styles is changing what you want changed and not changing what you do not want changed. You may want to make all coefficients italicized but not any of the other results. You may want to emphasize all the statistics on the coefficient age by making them bold but not change the rest of the covariates. Hitting your target is what matters. So we will call this targeting.

We are going to use numeric format to demonstrate. Changes to numeric format can be seen in all export formats and in the Results window. Changes to numeric formats can even be seen in the Linux console version of Stata.

Let's use a table created from one of our simple regressions from earlier. We will not show the regression results,

```
. collect clear
. collect: regress bpsystol age weight lead
```

but we will show the table we lay out.

```
. collect layout (colname) (result[_r_b _r_ci _r_se _r_z _r_p])
Collection: default
      Rows: colname
   Columns: result[_r_b _r_ci _r_se _r_z _r_p]
   Table 1: 4 x 5
```

	Coefficient	95% CI		Std. error	t	p-value
Age (years)	.6373174	.6057546	.6688803	.0160998	39.59	0.000
Weight (kg)	.3998766	.3644918	.4352614	.0180494	22.15	0.000
Lead (mcg/dL)	.1183383	.0296721	.2070044	.0452276	2.62	0.009
Intercept	70.08091	67.04886	73.11296	1.546613	45.31	0.000

Command `collect style cell` has option `nformat()`, which lets us set the numeric format. Let's change all numeric formats on the entire table to %7.4f.

```
. collect style cell, nformat(%7.4f)
```

We did not specify anything after `cell`, so we are changing the format for everything. Let's see the effect of that change.

```
. collect preview
```

	Coefficient	95% CI		Std. error	t	p-value
Age (years)	0.6373	0.6058	0.6689	0.0161	39.5853	0.0000
Weight (kg)	0.3999	0.3645	0.4353	0.0180	22.1546	0.0000
Lead (mcg/dL)	0.1183	0.0297	0.2070	0.0452	2.6165	0.0089
Intercept	70.0809	67.0489	73.1130	1.5466	45.3125	0.0000

Everything has four decimals. What if we want to change the format of only the coefficients? Recall that the coefficients are level _r_b in dimension result. We simply specify the tag result[_r_b] as the only value for which we want to change the format.

```
. collect style cell result[_r_b], nformat(%7.2f)
. collect preview
```

```
-----------------------------------------------------------------------
              |  Coefficient        95% CI       Std. error        t p-value
--------------+--------------------------------------------------------
Age (years)   |         0.64   0.6058   0.6689      0.0161 39.5853  0.0000
Weight (kg)   |         0.40   0.3645   0.4353      0.0180 22.1546  0.0000
Lead (mcg/dL) |         0.12   0.0297   0.2070      0.0452  2.6165  0.0089
Intercept     |        70.08  67.0489  73.1130      1.5466 45.3125  0.0000
-----------------------------------------------------------------------
```

Only the coefficients have two decimal places.

The format for the coefficients, their confidence intervals, and their standard errors is usually the same. Here is how we specify all of those results to have two decimal places.

```
. collect style cell result[_r_b _r_ci _r_se], nformat(%7.2f)
. collect preview
```

```
-------------------------------------------------------------------
              |  Coefficient      95% CI     Std. error        t p-value
--------------+----------------------------------------------------
Age (years)   |         0.64   0.61     0.67       0.02 39.5853  0.0000
Weight (kg)   |         0.40   0.36     0.44       0.02 22.1546  0.0000
Lead (mcg/dL) |         0.12   0.03     0.21       0.05  2.6165  0.0089
Intercept     |        70.08  67.05    73.11       1.55 45.3125  0.0000
-------------------------------------------------------------------
```

We typed result[_r_b _r_ci _r_se] to target all three of the results, just as we would type result[_r_b _r_ci _r_se] on collect layout to select the three results for rows or columns. Styles are yet another reason why tags, dimensions, and levels are so important in the collection system.

We could go on formatting results, but you get the idea.

We can target any dimension that tags any value or label on our table. If we wanted to draw our reader's attention to the results for covariate lead, we might change the color of its row to red, or we might bold the text. Instead, we will change the numeric format as a proxy for one of those more reasonable changes.

```
. collect style cell colname[lead], nformat(%7.5f)
. collect preview
```

```
-----------------------------------------------------------------------
              |  Coefficient        95% CI       Std. error        t p-value
--------------+--------------------------------------------------------
Age (years)   |         0.64     0.61     0.67         0.02 39.5853  0.0000
Weight (kg)   |         0.40     0.36     0.44         0.02 22.1546  0.0000
Lead (mcg/dL) |      0.11834  0.02967  0.20700      0.04523 2.61651 0.00891
Intercept     |        70.08    67.05    73.11         1.55 45.3125  0.0000
-----------------------------------------------------------------------
```

And now the results for lead are "emphasized".

Let's fit this same regression on males, females, and all data. The table command makes that easy. We will not show the results of table.

```
. table sex, command(regress bpsystol age weight lead)
```

Instead, we will show some tidier results.

```
. collect layout (colname#result[_r_b _r_se]) (sex)
Collection: Table
      Rows: colname#result[_r_b _r_se]
   Columns: sex
   Table 1: 8 x 3
```

	Sex Male	Sex Female	Sex Total
Age (years)	.4756206	.783255	.6373174
	.0221995	.023314	.0160998
Weight (kg)	.3499395	.440647	.3998766
	.0281172	.0262451	.0180494
Lead (mcg/dL)	.1154999	.1008595	.1183383
	.0580126	.0850915	.0452276
Intercept	81.09842	61.13921	70.08091
	2.700181	2.133394	1.546613

It is hard to tell the standard errors from the coefficients on that table. We could use a header style to add row labels for the coefficient and standard error, but let's instead put parentheses around the standard errors. That can be done using the `sformat()` option of `collect style cell`.

```
. collect style cell result[_r_se], sformat((%s))
. collect preview
```

	Sex Male	Sex Female	Sex Total
Age (years)	.4756206	.783255	.6373174
	(.0221995)	(.023314)	(.0160998)
Weight (kg)	.3499395	.440647	.3998766
	(.0281172)	(.0262451)	(.0180494)
Lead (mcg/dL)	.1154999	.1008595	.1183383
	(.0580126)	(.0850915)	(.0452276)
Intercept	81.09842	61.13921	70.08091
	(2.700181)	(2.133394)	(1.546613)

Yes, somewhat surprisingly, you can apply both a numeric and a string format to a value. Once the value is numerically formatted, it is then passed through a string format. For numeric values, that string format is primarily used just as we used it here—to adorn the result.

Advanced targeting

What if we want to emphasize just one result in this whole table? What if the age coefficient for females was of particular import to our research? We saw just above that we could specify multiple tags by including multiple levels in a dimension using styles. We can also use tag interactions when applying styles. It takes three tags to identify the result we described—`result[_r_b]`, `colname[age]`, and `sex[2]`. The way we specify that all of those tags are required is to interact them—`result[_r_b]#colname[age]#sex[2]`. The translation of that interaction term into English is literally result must be coefficient and covariate must be age and sex must be female. We put that term as the argument to `collect style cell` and type the command.

```
. collect style cell result[_r_b]#colname[age]#sex[2], nformat(%7.2f)
```

Previewing our table gives

```
. collect preview
```

	Sex Male	Female	Total
Age (years)	.4756206	0.78	.6373174
	(.0221995)	(.023314)	(.0160998)
Weight (kg)	.3499395	.440647	.3998766
	(.0281172)	(.0262451)	(.0180494)
Lead (mcg/dL)	.1154999	.1008595	.1183383
	(.0580126)	(.0850915)	(.0452276)
Intercept	81.09842	61.13921	70.08091
	(2.700181)	(2.133394)	(1.546613)

Our desired coefficient has been "highlighted".

More likely, we want to "highlight" both the coefficient and its standard error. That just requires that we specify the tags for both coefficient and standard error, rather than just for the coefficient.

```
. collect style cell result[_r_b _r_se]#colname[age]#sex[2], nformat(%7.2f)
. collect preview
```

	Sex Male	Female	Total
Age (years)	.4756206	0.78	.6373174
	(.0221995)	(0.02)	(.0160998)
Weight (kg)	.3499395	.440647	.3998766
	(.0281172)	(.0262451)	(.0180494)
Lead (mcg/dL)	.1154999	.1008595	.1183383
	(.0580126)	(.0850915)	(.0452276)
Intercept	81.09842	61.13921	70.08091
	(2.700181)	(2.133394)	(1.546613)

Okay, we will do one thing just for looks. Let's get rid of that obnoxious vertical rule. You never see those in publications.

```
. collect style cell border_block, border(right, pattern(nil))
. collect preview
```

	Sex Male	Female	Total
Age (years)	.4756206	0.78	.6373174
	(.0221995)	(0.02)	(.0160998)
Weight (kg)	.3499395	.440647	.3998766
	(.0281172)	(.0262451)	(.0180494)
Lead (mcg/dL)	.1154999	.1008595	.1183383
	(.0580126)	(.0850915)	(.0452276)
Intercept	81.09842	61.13921	70.08091
	(2.700181)	(2.133394)	(1.546613)

We specified the `border_block` dimension, but we did not need to target a specific level. We turned off right borders on every block in the table, which includes those that were creating that vertical rule. `pattern(nil)` is a programmery way of saying no line.

Saving and using

Do not forget you can save and use styles; see [TABLES] **collect style save**.

If you get a table styled just the way you want, you can save its style and apply that style to other similar tables. There is also nothing wrong with keeping all your style commands in their own do-file and running that do-file before you preview a similar table.

Either way works fine. The advantage of keeping your style commands in a do-file is that you can review and change them in the do-file. Keeping a do-file is more challenging if you are using the Table Builder to style your table.

Exporting

We are not going to say much about exporting, which seems odd given that exporting will be the end goal for many tables. There just is not much to say. You type `collect export`, followed by a filename with the format you want as the file suffix. That's about it. This is an entry about concepts, and exporting does not have many concepts to explain.

What we will tell you is that not all styles export to all export formats. If you are exporting to Microsoft Word or to HTML, you are in luck. Almost all styles export to those formats. If you are exporting to plain text (`.txt`), you are out of luck. Aside from numeric formats and some text positioning, almost no styles export to plain text.

To learn more about exporting tables from a collection, see `collect export`.

Saving collections

You can save and restore collections. There is not anything conceptually interesting to add to that.

We do recommend that if you are typing `collect` commands interactively that you do save your work by saving your collection.

Managing collections

You can list the collections in memory, set the current collection, copy collections, combine collections, rename collections, and drop collections. All of those operations can be useful. None of those operations is fraught with conceptual challenges.

Just to be clear: combining collections is no different from adding to an existing collection using repeated `collect` prefixes or `collect get` commands without `collect clearing`.

Also see

[TABLES] **Intro 3** — Workflow outline

[TABLES] **Intro 4** — Overview of commands

Description   Remarks and examples   References   Also see

Description

We outline the basic steps and reference the key commands in creating tables using collections.

Remarks and examples

We do not discuss the commands below, but you can click on the links to learn more about any command.

Outline of basic steps and key commands

1. Collect results from Stata commands.

   . **collect:** *command* ...

   . **collect get** ...

   . **table** ...

   . **dtable** ...

   . **etable** ...

   Every time you type `collect:` or `collect get`, you are adding the results to a collection. So collections may contain the results from multiple commands.

2. Perhaps combine collections.

   . **collect combine** ...

3. See what is in the collection.

   . **collect levelsof** ...

   . **collect dims** ...

   . **collect label list** ...

4. Lay out the rows and columns of your table.

   . **collect layout** ...

   Or use the Tables Builder.

   Then, reconsider your layout. And repeat until you have what you want.

5. Remove any unwanted results from the collection.

   . **collect unget** ...

6. Decide you do not like the default labels or titles in the headers, and change them.

```
. collect label ...
```

```
. collect style header ...
```

```
. collect style row ...
```

```
. collect style col ...
```

7. Customize your table—formats, bolding, italics, colors, and more.

```
. collect style cell ...
```

```
. collect stars ...
```

This often requires several steps.

This might involve applying a style shipped with Stata or one that you have previously saved.

Some customizations are specific to your intended export format.

8. Export your table.

```
. collect export ...
```

9. Perhaps save your layout, headers, and customizations as a style file.

```
. collect style save ...
```

```
. collect label save ...
```

Save your labels too.

You can now skip or abbreviate steps 5 and 6 on future tables that are similar to this table.

Even if those styles and labels do not get you all the way with a new table, they may save you a number of steps.

10. Perhaps save the collection.

```
. collect save ...
```

You can now come back to the collection and continue making changes to this table or create a different table from the same results.

Before you can effectively perform steps 3 through 6 you will need a working knowledge of tags, dimensions, and the levels that identify tags in a dimension. See [TABLES] **Intro 2**.

It is also helpful to have a basic understanding of how `collect layout` lays out a table. You need this to effectively handle the inevitable surprises that occur when performing step 4–lay out the rows and columns of your table.

To see examples demonstrating this workflow, see [TABLES] **Example 1**–[TABLES] **Example 7**. Also, see Juul and Frydenberg (2021, chap. 15) and Mitchell (2025).

References

Juul, S., and M. Frydenberg. 2021. *An Introduction to Stata for Health Researchers*. 5th ed. College Station, TX: Stata Press.

Mitchell, M. N. 2025. *Create and Export Tables Using Stata*. College Station, TX: Stata Press.

Also see

[TABLES] **Intro 4** — Overview of commands

Description Remarks and examples Also see

Description

We give an overview of all commands in the `collect` suite, organized by their intended use.

Remarks and examples

Remarks are presented under the following headings:

Introduction

In [TABLES] **Intro 3**, we introduced the basic workflow for creating a table using `collect`. Here we provide a more detailed overview of all the commands in the `collect` suite and information on how each one may be useful in the process of creating a table.

Prepare to collect results

Before collecting results for a new table, you will want to start with an empty collection. There are two ways to do this. You can create a new empty collection or clear all collections from memory.

`collect create`	Create a new collection
`collect clear`	Clear all collections in memory

If you have not collected any results since you opened Stata, you can skip this step—the collected results will be placed in the empty `default` collection.

Collect results

The next step in creating a table is to collect results from one or more Stata commands.

`collect` prefix	Collect results from the prefixed command
`collect get`	Collect results from a previously run command

Alternatively, you can use the `table`, `dtable`, and `etable` commands to create an initial table and place the results in a collection in one step.

Combine collections

You can work with multiple collections at once by iteratively using `collect create` followed by `collect get` or the `collect` prefix. If you want to create a single table with results from multiple collections, you can first combine the collections.

`collect combine`	Collect results from existing collections

Explore the collection

Values in the collection are organized according to their associated tags (comprising dimensions and levels within the dimensions). Before creating and modifying a table, you will need to know about the tags, dimensions, and levels of those dimensions in your collection. These will be used in subsequent `collect` commands.

`collect dims`	List dimensions in a collection
`collect levelsof`	List levels of a dimension
`collect label list`	List the levels of a dimension along with their labels

Modify the collection

After collecting results, you may want to modify the tags that are associated with the values in your collection. This allows you to customize the way values will later align in row and column headers when you lay out the table. Additionally, you can create composite results if you would like to display multiple results in a single cell.

`collect addtags`	Add tags to items in a collection
`collect unget`	Remove results from a collection
`collect recode`	Recode dimension levels in a collection
`collect remap`	Remap tags in a collection (modifying tags within or across dimensions)
`collect composite`	Manage composite results in a collection

Lay out rows and columns of the table

With results stored in a collection, you can construct a table by identifying what belongs on the rows and columns (and possibly even separate tables).

`collect layout`	Arrange values in the collection into a table
`collect style autolevels`	Specify statistics to be automatically added to the table

Preview the table

At the time you lay out your table, you will see a preview of the table. As you make changes to the table using the commands described in the following sections, you will likely want to see a preview of the table after each change.

`collect preview`	Preview the table in a collection

Modify labels in row and column headers

Once an initial table is created, you may want to modify what appears in the row, column, and table headers by default. You can select whether labels, titles, or nothing appears for each dimension and for each level of a dimension. You can also modify the default labels.

`collect label dim`	Add or modify the label for a dimension
`collect label levels`	Add or modify labels for levels within a dimension
`collect label use`	Apply labels from an external label file
`collect label drop`	Drop dimension and level labels
`collect style header`	Specify whether titles, labels, or nothing is shown for a dimension or for levels of a dimension
`collect style row`	Change arrangement of row headers, how factor variables are displayed, how duplicates are reported, and how long labels wrap
`collect style column`	Change arrangement of column headers, how factor variables are displayed, how duplicates are reported, and the width and spacing of columns
`collect style table`	Change display of factor variables in table headers

Control display of zero coefficients in regression results

When regression results are included in a table, coefficients with values of 0 are reported for covariates that are dropped because of collinearity, base levels of factor variables, and empty cells in factor-variable interactions. You can choose to show or hide these 0-valued coefficients.

`collect style showomit`	Show or hide omitted covariates
`collect style showbase`	Show or hide base levels of factor variables
`collect style showempty`	Show or hide empty cells of factor-variable interactions

Change styles—formats, bolding, colors, and more

To complete a table, you may want to modify the look of cells in the body of the table or in the row and column headers.

`collect style cell`	Modify formats, bolding, italics, colors, and more
`collect stars`	Add stars representing statistical significance
`collect style html`	Change appearance of cell borders and header cells for tables exported to HTML
`collect style putdocx`	Change width, indentation, and spacing of tables to be included in a report created by `putdocx`
`collect style putpdf`	Change width, indentation, and spacing of tables to be included in a report created by `putpdf`
`collect style use`	Apply styles from an external style file
`collect style clear`	Clear all collection styles

Add a title and notes

You can also add a title and notes to a table and customize their appearance with bolding, italics, font color, and more.

`collect title`	Add a custom table title in a collection
`collect style title`	Collection styles for table titles
`collect notes`	Add table notes in a collection
`collect style notes`	Collection styles for table notes

Query collection style properties

After applying styles from an external file and making several edits, you may want to check the current settings. You can query style properties for row headers, base levels, the position of the intercept, and more.

`collect query`	Query collection style properties

Export the table

After customizing the table, you can export it to Microsoft Word, HTML, PDF, Microsoft Excel, LaTeX, Markdown, SMCL, or plain text. You can also incorporate the table into a report created with `putdocx`, `putpdf`, or `putexcel`.

`collect export`	Export table from a collection
`putdocx collect`	Add a table to a report created by `putdocx`
`putpdf collect`	Add a table to a report created by `putpdf`
`putexcel` *ul_cell* `= collect`	Add a table to a report created by `putexcel` with the top left cell of the table in *ul_cell*

Save styles and labels

If you have built a table with styles or labels you would like to apply to other tables, you can save these to a file.

`collect label save`	Save labels to a file
`collect style save`	Save styles to a file

Save the collection

If you would like to use the collection you created in the future to build a new table or further modify the existing table, you can save the collection and use it later.

`collect save`	Save a collection to disk
`collect use`	Use a collection from disk

Manage collections

You can work with one or more collections in memory. With multiple collections, you can set the active collection. You can also list, copy, rename, and drop collections.

`collect dir`	Display names of all collections in memory
`collect set`	Set the current (active) collection
`collect copy`	Copy a collection
`collect rename`	Rename a collection
`collect drop`	Drop collections from memory

Also see

[TABLES] **Intro 3** — Workflow outline

[TABLES] **Intro 5** — Other tabulation commands

Description

Tables can be produced by a few other commands.

Remarks and examples

The `table` command is not considered an "other" table command. It is not listed below. Although `table` is documented in [R], it is part of the collection system. It is actually implemented on top of the collection system documented in this manual. `table` leaves behind a collection that can be manipulated using all the tools in this manual.

Other tabulation commands are the following:

Command	Description
`tabulate` (`oneway`)	One-way tabulations with percentages and cumulative percentages
`tabulate` (`twoway`)	Two-way tabulations. Optionally computes statistics for independence of the rows and columns
`tabulate, summarize()`	One-way or two-way tabulations of summary statistics
`tabstat`	Tables of summary statistics
`svy: tabulate` (`oneway`)	Survey version of `tabulate` (`oneway`)
`svy: tabulate` (`twoway`)	Survey version of `tabulate` (`twoway`)

Aside from the independence statistics computed by two-way `tabulate` and two-way `svy: tabulate` and the cumulative percentages computed by (one-way) `tabulate`, all of these commands have been superseded by [R] **table**. The independence statistics are returned in `r()` and can be collected.

The `svy` versions of `tabulate` also have options that return MEFF, DEFF, and other survey statistics. Those results are returned in `e()` and can be collected.

Aside from the independence statistics and survey statistics, none of these commands returns results, and thus, their tabulations cannot be used in collections. That said, it is often useful to collect independence statistics and include them in tables created from collections.

To be fully truthful, `tabstat` with the `save` option will store results into `r()`. You would never collect these results because `table` can compute all of the statistics that `tabstat` can, and more. More importantly, `table` automatically puts all the statistics it computes into a collection that is easy to work with.

These commands are all are still documented primarily because they provide an easy and familiar way to quickly analyze your data. That is, the data and the independence statistics.

Also see

[TABLES] **Intro 4** — Overview of commands

Description

After you have collected results using `collect get`, `collect:`, `etable`, `dtable`, or `table`, use the Tables Builder to create tables of the results. In the Tables Builder, you can

- choose which results go in a table,
- lay out the rows and columns,
- change how row and column headers are arranged,
- change the look of anything, and
- export a table to Microsoft Word, Microsoft Excel, PDF, HTML, LaTeX, SMCL, Markdown, and plain text.

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Build and style table

Remarks and examples

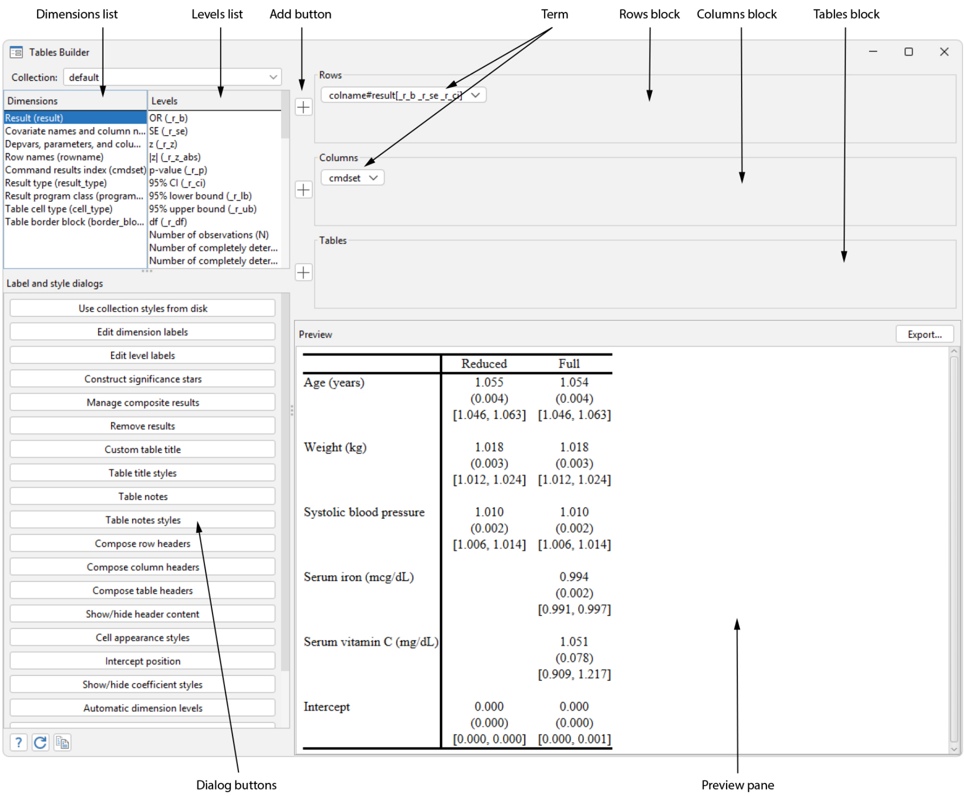
Remarks are presented under the following headings:

Overview

The Tables Builder works with an existing collection. You create a collection by collecting results from commands using `collect get`, `collect`, `etable`, `dtable`, and `table`. The Tables Builder lets you choose which results go in a table, lay out the rows and columns of a table, change how row and column headers are arranged, change the look of anything in the table, and export a table to Microsoft Word, Microsoft Excel, PDF, HTML, LaTeX, SMCL, Markdown, and plain text.

You launch the Tables Builder by selecting the menu item **Statistics** > **Summaries, tables, and tests** > **Tables and collections** > **Build and style table**

You can see the major components of the Tables Builder in this diagram.



To use the Builder, you will need a basic understanding of what a collection is. In particular, if seeing the words *Dimensions* and *Levels* above leaves you confused, read *Tags, dimensions, and levels* in [TABLES] **Intro 2**.

If you are creating a table only to understand your data, you probably just need to lay out your table. If you are creating a table for publication or to otherwise share with others, that is likely to be an iterative process. It may take several steps to get the layout you want, then some steps to get the headers exactly as you want them, and yet more steps to adjust numeric formats and text styles.

Laying out a table

The first step in creating a table is deciding what goes on the table's rows and columns. We cannot help you with that. The second step is getting those things onto the rows and columns, and the Builder can help with that. The dimensions listed in the *Dimensions* list represent categories of values pulled from results you collected. Each line in the list is a dimension description followed by the dimension name in parentheses.

Choose which category you want to place on the rows of your table by clicking on one of the dimensions in the list. If you want all the values associated (tagged) with that dimension, then just click on the + to the left of the *Rows* box. If you do that, all the levels of that category become potential rows in the table. And you will see a term added to the *Rows* box. That term will be identified with the name of the dimension.

If you want only a subset of the dimension's levels, then select them from the *Levels* list before hitting the + control. As with the dimensions, each line in the list is a level description followed by the level name in parentheses. You can click one level, then shift-click another to select a range. You can control-click levels to select levels that are not neighbors in the list. If you select levels, their names will become part of the description in the added term.

Do not worry too much about selecting the levels; you can modify the selected levels later.

We say "potential rows" because you must identify specific stored values using your selected dimensions, or the Builder cannot create a table. For a detailed discussion of what is required to properly specify a layout, see *How collect layout processes tag specifications* in [TABLES] **Collection principles**.

If your table can be represented by a single dimension, you will see results in the Preview pane, and you are done with the basic layout. This might be the case if you are using the Builder to pretty up the results from a one-way tabulation command such as `table rep78`.

More likely, you are creating a two-way, or even multiway, table. To create a two-way table, you will need to put a dimension into the *Columns* box. Again, click on the dimension, optionally select some levels, and then click on the + beside the *Columns* box. A term for that dimension will appear in the *Columns* box.

If two dimensions are enough to lay out your table, you will see a table with the results you want in the Preview pane. If two dimensions are not enough for your table, you will see one of two things in the Preview pane.

1. You might see a message with some suggestions for other dimensions that may help lay out your table. If so, try adding one of those dimensions to one of the terms in your *Rows* or *Columns* box. (See *Laying out a multiway table*.)
2. You might see a table with results you are not interested in, and either the rows or columns have labels that may confuse you but certainly do not interest you. What has happened? This is really the same problem. You need a multiway table to present your results, and you have only laid out a two-way table. Let's tell you how to proceed and then tell you why it happened.

First, think about the table you are trying to create. One, or more, of the dimensions you have not yet put into the *Rows* or *Columns* box is required to identify the super-rows or super-columns of a multiway table.

If you cannot think of any dimensions that you should add, clear your current layout by clicking on the  button in the lower left of the Builder. Repeat the process of selecting and adding dimensions to your layout, but this time carefully select only the levels you are interested in. Now you will see a message in the Preview pane, and you can go back and proceed from item 1 above.

So what happened to create that useless table? The commands that collect results collect everything by default, including things you may not want on your table. You may have noticed this when selecting dimensions for your table. You may have seen levels like `c1` and `c2` that do not interest you. Regardless, they are in the collection, as are the values they tag. When the Builder laid out your table by searching over all the levels in the dimensions you selected for rows and columns, it found results that exactly matched some of the row/column tag combinations. That happened because you asked for a two-way table when you wanted to ask for a three-way or higher multiway table. Once you add the dimensions for a multiway table, the results found for the mistaken two-way table will no longer be found—they only match the unwanted two-way table. For more details than you probably want about what is required to properly specify a layout, see *How collect layout processes tag specifications* in [TABLES] **Collection principles**.

Laying out a multiway table

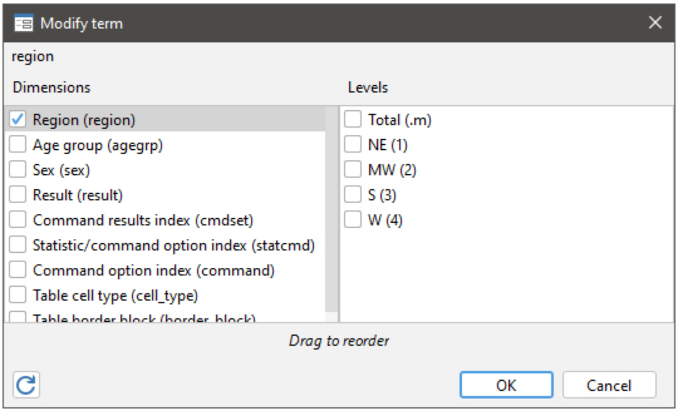
A multiway table has super-rows, super-columns, or both to present results that require more than two dimensions. Consider a cross-tabulation of frequencies by the categorical variables `sex`, `region`, and `agegrp`. Those counts form a cube with dimensions for each of the categorical variables. We cannot put three-way results directly on a two-way table. What we can do is create a three-way table using super-columns or super-rows for one of the dimensions. Here is what that looks like with super-rows for `sex`:

```
. use https://www.stata-press.com/data/r19/nhanes2l
(Second National Health and Nutrition Examination Survey)

. table (sex agegrp) (region)

----------------------------------------------------
              |                Region
              |     NE       MW        S        W    Total
--------------+-------------------------------------
Sex           |
  Male        |
    Age group |
      20-29   |    204      340      290      282    1,116
      30-39   |    169      199      200      202      770
      40-49   |    122      182      175      131      610
      50-59   |    146      141      170      145      602
      60-69   |    290      333      381      365    1,369
      70+     |     87      115      116      130      448
      Total   |  1,018    1,310    1,332    1,255    4,915
  Female      |
    Age group |
      20-29   |    240      344      299      321    1,204
      30-39   |    165      234      243      210      852
      40-49   |    124      194      190      154      662
      50-59   |    146      166      215      162      689
      60-69   |    297      388      430      376    1,491
      70+     |    106      138      144      150      538
      Total   |  1,078    1,464    1,521    1,373    5,436
  Total       |
    Age group |
      20-29   |    444      684      589      603    2,320
      30-39   |    334      433      443      412    1,622
      40-49   |    246      376      365      285    1,272
      50-59   |    292      307      385      307    1,291
      60-69   |    587      721      811      741    2,860
      70+     |    193      253      260      280      986
      Total   |  2,096    2,774    2,853    2,628   10,351
----------------------------------------------------
```

There are several ways to lay out a three-way table in the Builder.

Method 1. The easiest way is to start by laying out a two-way table using just two of your three required dimensions. You do that exactly as we did in *Laying out a table*. At that point, you probably have a message in the Preview pane that you need to add more dimensions. Or you might have a useless table that is not what you want. Either is to be expected. Ignore that and move on.

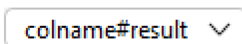
Time to add your third dimension. Click on the in one of the two terms you just created. If you want a table with super-rows, click on the on the term in the *Rows* box. If you want a table with super-columns, click on the on the term in the *Columns* box. From the resulting drop-down list, select **Edit**. You will be presented with a *Modify term* dialog that looks something like



This dialog box looks and acts a lot like the *Dimensions* and *Levels* list from the main Builder window. Note that there will be a check mark in the box beside the dimension you previously selected for the term. That means the checked dimension is part of this term. If you had selected specific levels when creating the term, those will have check marks beside their lines in the *Levels* list of the dialog.

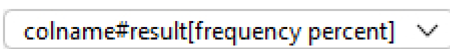
Now add your third dimension—the one you have not yet used—to this term. To do that, just click on the box beside the third dimension in the dialog. If you only want some of the levels on your table, you can click on the boxes beside the levels you want on the table. Now click on **OK**. You will see your term updated in the *Rows* or *Columns* box. More importantly, if the three dimensions you have selected do indeed identify the table, you will see a table in the Preview pane.

The term you modified will now look something like



where `colname` and `result` are the names of the dimensions you selected for the term. Your dimension names will probably not be `colname` and `result` unless you happen to be creating a comparative regression table. They will be the names of the dimensions that define your table.

The term might also look like



if you specifically selected the levels `frequency` and `percent` from the levels of dimension `result`.

The # is actually part of the language for specifying table layouts when you use the command `collect layout` instead of the Tables Builder. # instructs `layout` or the Builder to interact the levels of the dimensions. If you think about how dimensions in the *Rows* and *Columns* boxes are used, they too are interacted. In the case of the rows and columns of a two-way table, we want all possible pairings of the levels from the row and column dimensions. Those pairings form the cells in the table. When we created the interaction in our term, we were requesting all possible triplets of the levels of the three dimensions. The third dimension is organized as super-rows or super-columns.

Method 2. We can create a term with an interaction directly from the *Dimensions* list. We start as we always do. Pick a dimension, and possibly select levels; then, use one of the + controls to put it into the *Rows* or *Columns* box. Then, click on that newly added term. It will be highlighted. Then, select the third dimension from the *Dimension* list, and click on the + control for the *Rows* or *Columns* box where the term is highlighted. The new dimension will be added to the highlighted term, and it will become an interaction term. Finally, add one or more other dimensions to the empty *Rows* or *Columns* box.

Other methods. We did not have to perform all the steps in method 1 or method 2 in exactly the order as written. And we can mix and match the methods from method 1 and method 2. Put one term into the *Rows* box and then another into the *Columns* box; then use method 2 to create interaction terms. We can delete terms and start over.

Moreover, we can create four-way tables, five-way tables, all the way up to silly-way tables. We can use method 1 and method 2 to create as complex a multiway table as we like. The important thing is to get our the interaction terms that specify our rows, super-rows, super-super-rows, . . . ; and our columns, super-columns, . . . built so that we get the table we want.

Modifying the layout

There are four main things we might want to change in a layout:

1. Make the rows columns and the columns rows.
2. Make super-rows rows and rows super-rows.
3. Make super-columns columns and columns super-columns.
4. Add or remove levels from a dimension.

1. Make the rows columns and the columns rows. Swapping rows and columns is both a common desire and easy. You might accidentally put a cities ⌄ dimension on the columns and realize that 87 cities makes for a truly wide table. Drag the `cities` dimension to the *Rows* box, where you always wanted it, and drag whatever dimension is in the *Rows* box to the *Columns* box. Done.

This works with terms that have interactions too. If you drag an interaction term from the *Rows* box into the *Columns* box, you are simply converting the whole super-row/row structure into a super-column/column structure. And vice versa when dragging from the *Columns* box to the *Rows* box.

2. Make super-rows rows and rows super-rows. This is a little more subtle, and you may have already encountered it if you tried your own example when we discussed multiway tables above. We showed a table in *Laying out a multiway table*. It had levels of the dimension `sex` as super-rows and levels of the dimension `agegrp` as rows. That makes the comparison of age groups within sex easy. What if you really wanted to compare females and males within age groups?

We need to swap the dimensions that are on the rows and super-rows. First, click on the ⌄ on the sex#agegrp ⌄ term in the *Rows* box, and select **Edit** to launch the *Modify term* dialog. You would see that the `sex` and `agegrp` dimensions are checked and that `sex` appears above `agegrp`. Simply click-and-hold on the `agegrp` dimension, and drag it above the `sex` dimension. Or click-and-hold on the `sex` dimension, and drag it below **agegrp**. Click on **OK**. The dimensions on the rows and super-rows have been swapped. The rows for females and males are now adjacent.

That is the general idea, and it also works for multiway terms where more than two dimensions are interacted.

3. Make super-columns columns and columns super-columns. This is really just step 2. Reread step 2, and substitute “column” everywhere you see “row”.

4. Add or remove levels from a dimension. You have rows you do not want, or columns that you want are missing. You may have done this to yourself when creating terms by selecting too few or too many levels for a dimension. Or you may have a collection from `table` that makes pretty draconian assumptions about which `result` dimensions you want from any `command()` options.

If you have been reading along, you may have guessed the answer to this one. Click on the ⌄ of the term for which you want to add or remove levels, and select **Edit** to launch the *Modify term* dialog. In the dialog, click on the dimension for which you want to change the list of levels. Then check or uncheck levels until all the levels you want are checked, and only the levels you want are checked.

Laying out stacked dimensions

There are cases where you do not want to interact a dimension with another dimension on the rows or columns as we did in *Laying out a multiway table*. Instead, you want to stack some results below some other results. (Or perhaps stack them to the right if the original results are on the columns.)

Consider a table comparing two regressions. Each regression appears in a column, so you already have the term `cmdset ⌄` in the *Columns* box. The coefficients and perhaps their standard errors and confidence intervals appear on the rows. The values for these statistics require that you also know the covariates. That means the *Rows* box will have an interaction term that looks like `colname#result[_r_b _r_se _r_ci] ⌄`.

Suppose you also want to place some of the model-level statistics below the coefficients and their statistics. Say you want the R-square, the model F statistic, and the p-value of the model F statistic. To add these statistics to the bottom of each regression, click on **Result (result)** in the *Dimensions* list, and then control-click on **R-square (r2)**, **Model F test (F)**, and **Model test p-value (p)** in the *Levels* list. Then, after making sure that nothing is selected in the *Rows* box, click on the + beside the *Rows* box. You will see a new term added to the *Rows* box, and the model-level statistics will be added to the bottom of the table.

Placing multiple results in a cell

Rather than stacking results or placing them on adjacent columns, you may prefer to place multiple results in a single cell. For example, suppose you have a table comparing regression results. You currently have `cmdset ⌄` in the *Columns* box and `colname#result[_r_b _r_ci] ⌄` in the *Rows* box, but you would rather place the confidence intervals in the same cell as the coefficients. You can create this layout by creating a composite result from the coefficients and confidence intervals. Click on **Manage composite results**, and provide a name for this new composite result; for example, you might name this result `coefci`. Then, for the *Composite result elements*, you would select `Coefficient (_r_b)` and `95% CI (_r_ci)`. You can then lay out your table by interacting `colname` with the level `coefci` of dimension `result`.

Multiple tables

We have thus far completely ignored the *Tables* box. It has just one job. Instead of creating super-rows or super-columns, any dimension placed in the *Tables* box creates multiple tables—one for each level of the dimension. If you place or create an interaction term in the *Tables* box, a new table will be produced for all combinations of the levels in the term.

What is more, you already know how to use it. Placing dimensions into the *Tables* box or creating interaction terms in the *Tables* box works exactly as it works in the *Rows* and *Columns* boxes. You can also drag terms into the *Tables* box from the *Rows* or *Columns* boxes, and vice versa.

Changing row and column headers

Row headers are the descriptions of the rows that you see along the left of a table. Column headers are the descriptions of the columns that you see along the top of a table. These headers are the reader's guide to understanding your table. If the headers are good, your table will be easy to read and understand. If the headers are bad, readers will be left scratching their heads.

Perhaps you are lucky, and your variable labels along with the default result labels and the default header composition will be just what you want. Perhaps not. If not, there is plenty of control for the headers in the Builder. You just have to know where to look.

Let's start with the easiest part, the text actually written in the headers.

Text/labels

You can change the label for any dimension on your table. This is the label for the dimension itself, not for the levels of the dimension. It is the levels that actually form the rows and columns of your table. Sometimes, the dimension label is shown in the header; sometimes, it is not. See *Layout* for a discussion.

To change one or more of your dimension labels, click on the button **Edit dimension labels** to launch the dimension labels dialog. There you can select dimensions, see their current label, and change that label.

You can also change the labels for levels in a dimension. These labels almost always appear on the table because they identify the rows and columns. An exception is comparative regression tables that sometimes identify standard errors by placing them in parentheses and confidence intervals by placing them in brackets. Such tables often dispense with labeling the statistics, the levels, altogether. Regardless, if you want to change labels for levels, click on the **Edit level labels** button to launch a dialog for editing labels. In that dialog, you can choose the dimension with the levels you want to change, choose the level you want to change, see the current label for the level, and change that label.

Layout

There are two components to header layout—what things are shown and hidden, and how the labels and names are composed to form the header.

Let's start with what is shown and hidden. In *Text/labels*, we alluded to cases where dimension labels themselves might or might not be shown. Consider a dimension created from a birth sex variable. The two levels in that dimension would likely be labeled `Female` and `Male`. Those labels are self-explanatory and do not need any additional labeling to be clear. We do not need, and probably do not want, the label for the dimension itself. We would be perfectly happy with row headers that look like this:

```
Female  ...
Male  ...
```

Now consider a variable from a Likert scale. Say we are talking about websites, and the label on the dimension is "Easy to navigate". The levels are labeled "Strongly agree", "Agree", "Neutral", "Disagree", and "Strongly disagree". Those level labels do not mean anything by themselves. We need the dimension label—"Easy to navigate". We now need row headers that look like this:

```
Easy to navigate
  Strongly agree  ...
  Agree  ...
  Neutral  ...
  Disagree  ...
  Strongly disagree  ...
```

Click on the **Show/hide header content** to launch a dialog that lets you control which labels or names are shown. You can choose to show labels, show names, or show nothing (hide). You can make this decision for dimensions, for all levels in a dimension, or for individual levels in a dimension. You can even set the default behavior across all dimensions and levels.

Once your header content is set, you can change how row and column headers are composed or constructed. You do that by clicking on the **Compose row headers** or **Compose column headers** button.

That word "composed" encompasses a lot of choices. Let's consider a few.

Do you want cross-tabulation row headers that look like

```
Female  East
Female  West
Female  North
Female  South
Male    East
...
```

or like

```
Female
  East
  West
  North
  South
Male
  East
...
```

Then choose either **Split elements across columns** or **Stack elements in a single column** on the *Compose row headers* dialog.

Do you want your factor-variable interactions to look like

```
Sex # Region
  Female # East
  Female # West
  Female # North
  Female # South
  Male # East
  ...
```

or like

```
Sex
Region
  Female
    East
    West
    North
    South
  Male
    East
...
```

Then choose between **Compose factor-variable elements in a single cell** and **Split factor-variable elements into separate cells** on the *Compose row headers* dialog.

Similar choices can be made about column headers on the *Compose column headers* dialog.

There are many other changes to row and column headers you can make from these dialog boxes. We suggest you launch the dialogs and explore.

Appearance

You can change many things about the appearance of row and column headers, including borders and horizontal or vertical rules, bolding, italics, font color, cell color, margins, and justification.

To change any of these properties for the row headers, click on the **Cell appearance styles** button. In the resulting dialog, click on one of the check boxes for the dimension and level rows, and then select **Table cell type (cell_type)** as the dimension. If you want to change the appearance of row headers, select `row-header` for the level. Alternately, if you want to change the appearance of column headers, select `column-header` for the level. Then click on any of the dialogs tabs—**Borders**, **Diagonals**, **Fonts**, **Shading**, **Margins**, **Alignments**, or **Formats**—and make any appearance changes you want on that tab.

You can even target the appearance change to specific rows or columns. You do that by choosing a second dimension on the dialog. Choose one of the dimensions that is on the rows or columns of your table. Then, select the level of the dimension for the specific row or column header you want to change. Finally, make the appearance changes you want on one of the tabs.

Show/hide factor-variable base levels and empty cells

Many applications of factor variables require that one of the levels in the variable be declared a base level. In those applications, nothing is estimated for that level, making its coefficient effectively `0`. By default, tables produced by the Builder include base levels for factor variables in the table with `0`s for coefficients or means and blanks for standard errors and other statistics about the coefficient or mean.

If you would rather have base categories dropped from a table, click on the **Show/hide coefficient styles** button. On the resulting dialog, select **Show base levels for factor variables**. You then have several choices for which base levels are shown. You can show base levels for factor variables but not for interactions, show base levels for both factor variables and for interaction, or remove all base levels.

Factor variables can be interacted, and sometimes there are no observations in one of those interactions. Consider the interaction of race (which includes Aleut Eskimo) with city regions in Los Angeles (LA). If there are no Aleutians residing in East LA, then we say that cell of the interaction is empty.

As with base categories, you can control whether empty cells are shown or hidden on the dialog launched by the **Show/hide coefficient styles** button.

We put this discussion in *Changing row and column headers* because showing or hiding factor-variable base levels adds or removes entire rows or columns from the table, including their headers.

Show/hide omitted coefficients

Regressions and other estimators require that covariates not be collinear. If they are, this is flagged in the output with a `0` coefficient and an `(omitted)` note. By default, tables produced by the Builder include collinear covariates with `0`s for coefficients or means, and blanks for standard errors and other statistics about the coefficient or mean.

You can specify that collinear covariates instead be dropped from the results by clicking on the **Show/hide coefficient styles** button. On the resulting dialog, select **Show omitted coefficients**, and then click on the **Off** radio button.

Changing cell/results appearance

You can change just about anything about how the values in your table look—numeric formats, borders, horizontal and vertical rules, bolding, italics, font, text color, cell color, margins, justification, and more. This is all done in the dialog launched by the **Cell appearance styles** button.

If you want to change the default look of everything on the table, including the headers, click on the radio button labeled **Edit base style**. Then click on one of the tabs—**Borders**, **Diagonals**, **Fonts**, **Shading**, **Margins**, **Alignments**, or **Formats**. Whatever changes you make on those tabs will apply to all text or cells throughout the entire table, both cells in the body of the table and in the headers. Because you are modifying the base appearance, if you have previously made changes to more specific tags (dimensions and levels), those changes will still be applied.

To make changes that override the default appearance for everything, first click on the radio button labeled **Edit styles for specified tags**.

If you want to change the appearance of only the results and cells in the body of the table, and not the headers, then select the dimension **Table cell type (cell_type)** and select the level **item**. Now any changes you make on the other tabs applies to all results and cells in the main body of the table, and not to any of the headers.

You can be specific about which results are affected by the appearance changes you make on the other tabs. For example, if your table has regression results, you could select the **Result (result)** dimension, then select **Coefficient (_r_b)**. Any changes you make after that affect only the coefficients. You might then click on the **Formats** tab and change the format type to **Fixed numeric** with 2 digits to the right of the decimal. You could make the same changes to the **Std. error (_r_se)** and **95% CI (_r_ci)** by repeating the process on those levels.

You can get even more specific. So far, we have picked only one dimension. The **Main** tab of the dialog allows you to pick up to 10 dimensions. Continuing with the regression example from the prior paragraph, after selecting **Coefficient (_r_b)** from the **Result (result)** dimension, we might now pick a second dimension, say, **Covariate names and column names (colname)**. Imagine we are trying to highlight the results from one of the covariates in our regression. We would then select that covariate from the *Level* drop-down list. So we have two things: the coefficient and the covariate we want to emphasize. With those two tags chosen, any changes we make on the other tabs affect only the coefficient and our chosen covariate. We can highlight those results any way we wish—bold, italics, text color, cell color, etc. You could make the same changes to the **Std. error (_r_se)** and **95% CI (_r_ci)** of the covariate by repeating the process on those levels.

It is important to realize that not all appearance edits can be rendered on all export formats. For example, plain text (.txt) is called "plain" for a reason. Aside from numeric formats and some horizontal and vertical rules, nothing from the appearance edits can be rendered in plain text. Most appearance edits can be rendered in HTML, PDF, Microsoft Word, and Microsoft Excel. Many, but fewer, can be rendered in LaTeX. No export format can render all the changes you can make on this dialog.

Because the Preview pane renders one of the exports, you also will not be able to see all appearance edits in the Preview pane. On Windows and Macintosh, the preview is HTML, so you will see most appearance edits. On Linux, the preview is plain text, so you will see almost none of your edits.

Numeric formats

Make changes to numeric formats on the **Formats** tab. There you can also specify the delimiters for confidence and credible intervals.

Borders and horizontal or vertical rules

Add or remove borders around cells on the **Borders** tab. You can create horizontal rules by specifying borders only on the top or bottom of cells. You can create vertical rules by specifying borders only on the left or right of cells.

Bold, italics, text color, and such

Make table text bold or italic, or change its color on the **Fonts** tab. You can also underline text on the **Fonts** tab, or render it with a strikeout. You can even put text in all capitals or initial capitals. There are also some special settings for LaTeX and SMCL export. You can specify the font family, but that must be done in a way that is supported by your intended export format.

Cell color

Change background and foreground cell color on the **Shading** tab. You can also set fill patterns on that tab.

Margins

Set margins for the cells on the **Margins** tab. You can add "in", "cm", or "pt" to any value you type here to specify that the margin is to be in inches, centimeters, or printers points.

Justification/alignment

Align text to the left or right and to the top or bottom on the **Alignments** tab.

Adding significance stars

You can add significance stars to coefficients in the dialog launched by clicking on the **Construct significance stars**. In fact, you can add any text you like to any result you like using rules on the value of any result you like. You can even limit the application of the "stars" to selected tags. A tag is defined by a dimension and one of its levels.

Adding a custom table title

You can add a custom table title in the dialog launched by clicking on the **Custom table title**.

Changing table title appearance

You can change the appearance of table titles in the dialog launched by the **Table title styles** button. Click on the **Fonts** tab to change the font, size, text color, bolding, italics, and more. Click on the **Shading** tab to change the background color, foreground color, and fill pattern.

Adding table notes

You can add table notes in the dialog launched by clicking on the **Table notes**.

Changing table note appearance

You can change the appearance of table notes in the dialog launched by the **Table notes styles** button. Click on the **Fonts** tab to change the font, size, text color, bolding, italics, and more. Click on the **Shading** tab to change the background color, foreground color, and fill pattern.

Exporting a table

Export your table by clicking on the **Export ...** button. You can choose from several export formats—Microsoft Word, Microsoft Excel, PDF, HTML, LaTeX, SMCL, Markdown, and plain text.

Advanced tools

Position of intercept

When your table contains regression results, the intercept is placed by default after all the other coefficients. If you prefer, you can have it placed before the other coefficients. Use the dialog launched from the **Intercept position** button.

Automatic dimension levels

If you requested specific results when creating your collection, the meaning of a dimension used alone in the Builder has been changed. For example, to collect results, you could type

```
. collect _r_b _r_ci: regress ...
```

If you then add the **Result (result)** dimension to one of the *Rows* or *Columns* boxes without choosing any levels from the `Level` list, that dimension will no longer represent the `result` dimension and all of its levels. It will instead represent only the levels you specified on dimension `result` when you collected the results.

Another way automatic levels are defined is when your collection was created by the `table` command and you specified one or more `command()` options. `table` makes some choices on its own about what you would like to see in your table.

Regardless of whether you requested specific results at collection time or `table` made some choices for you, you can change these "automatic" dimension levels by clicking on the **Automatic dimension levels** button. On that dialog, you can clear the automatic levels for any dimension. You can redefine the automatic levels for any dimension. Or you can create a new list of automatic levels for any dimension.

All of this can be tremendously helpful when typing commands to lay out a table. The Builder makes it so easy to select levels while you are choosing dimensions for the rows or columns that automatic levels are rarely helpful.

The most likely surprise you will encounter in the Builder occurs after a `table` command that included regressions specified in the `command()` option. You might then place the **Results (results)** dimension into the *Rows* or *Columns* boxes without selecting specific levels. If you do, only the coefficients will be shown. That is because `table` set the automatic levels of dimension `result` to be just the coefficients.

You could click on the **Automatic dimension levels** button and redefine the automatic levels for `result`. But why? You can just click on the ⌄ on the result ⌄ term, select **Edit**, and click on whatever levels you want in the *Modify term* dialog.

Rename dimension levels

You can rename, or formally recode, any level from any dimension in the dialog launched by clicking on the **Recode dimension levels** button. That would rarely be done from the Builder.

Remap tags

Another, even more advanced, operation you can perform from the Builder is to remap dimensions and their tags. Click on the **Remap tags** button to launch a dialog. On that dialog, you can move a level from one dimension to another dimension, create new dimensions and populate them from existing levels in existing dimensions, and rename a dimension. You can even limit this remapping to values where combinations of other tags are set.

collect get — Collect results from a Stata command

Description

The `collect get` command and `collect` prefix put results into a collection. Collected results are scalars, macros, and matrices from `e()` and `r()` as well as scalar and matrix expressions. Tables can be constructed from the results in the collection.

The `collect get` command identifies results from a previous Stata command that are to be put into a collection. The `collect` prefix puts results returned by the prefixed command into a collection.

Both the `collect get` command and the `collect` prefix allow you to specify a list of results to add to the automatic `result` levels (automatic levels) for subsequent table layouts. Specifying automatic levels at the time you collect results is an alternative to selecting the results to include at the time you lay out your table; see [TABLES] **collect layout**.

Quick start

Consume results from the regression model, and place them in the current collection

```
collect: regress y x
```

Same as above, and also add coefficients `_r_b` and standard errors `_r_se` to the list of automatic results

```
collect _r_b _r_se: regress y x
```

Fit a linear regression for each level of `catvar`, collect `e()` results from each regression, and add statistics `e(r2)` and `e(r2_a)` to the automatically included results

```
by catvar: collect e(r2) e(r2_a): regress y x
```

Consume results from `r()` results, and add statistics `r(stat1)` and `r(stat2)` to the list of automatic levels

```
collect get r(stat1) r(stat2)
```

Same as above, but place the results in collection `c2`

```
collect get r(stat1) r(stat2), name(c2)
```

Same as above, and attach the tags `dim1[lev1]` and `dim2["lev 2"]` to `r(stat1)` and `r(stat2)`

```
collect get r(stat1) r(stat2), name(c2) tags(dim1[lev1] dim2["lev 2"])
```

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Collect results

Syntax

Basic prefix syntax to consume results from Stata commands

`collect` [`get`] `:` *command*

Full prefix syntax to consume results from Stata commands

[*prefix* `...:`] `collect` [`get`] [*resultlist*] [*if*] [*in*] [, `tags(`*tags*`)` `commands(`*commands*`)`] `:` *command*

Consume results after running a Stata command

`collect get` *resultlist* [, `name(`*cname*`)` `tags(`*tags*`)` `commands(`*commands*`)`]

where *prefix* may be `by`, `capture`, `frame`, `noisily`, `quietly`, or `version`.

where *resultlist* is

result [*result* [...]]

result, when specified with the `collect` prefix, identifies individual results to be added to the list of automatic results.

result, when specified with `collect get`, indicates the type of results to be stored and identifies the results to add to the list of automatic results.

result may be one of the following:

result	Description
`r()`	all `r()` results
`e()`	all `e()` results
explist	returned results and named expressions

Specifying any `r()` result will also cause `collect` to consume all `r()` results. Specifying any `e()` result will also cause `collect` to consume all `e()` results.

explist specifies which results to collect. *explist* may include *result identifiers* and *named expressions*.

result identifiers are results stored in `r()` and `e()` by the *command*. For instance, *result identifiers* could be `r(mean)`, `r(C)`, or `e(chi2)`. After estimation commands, *result identifiers* also include the following:

Identifier	Result
`_r_b`	coefficients or transformed coefficients reported by *command*
`_r_se`	standard errors of `_r_b`
`_r_z`	test statistics for `_r_b`
`_r_z_abs`	absolute values of `_r_z`
`_r_df`	degrees of freedom for `_r_b`
`_r_p`	p-values for `_r_b`
`_r_lb`	lower bounds of confidence intervals for `_r_b`
`_r_ub`	upper bounds of confidence intervals for `_r_b`
`_r_ci`	confidence intervals for `_r_b`
`_r_cri`	credible interval (CrI) of Bayesian estimates
`_r_crlb`	lower bound of CrI of Bayesian estimates
`_r_crub`	upper bound of CrI of Bayesian estimates

named expressions are specified as *name* = *exp*, where *name* may be any valid Stata name and *exp* is an expression, typically an expression that involves one or more *result identifiers*. Expressions with spaces must be bound in parentheses. Examples of named expressions are `diff = (r(mean_2) - r(mean_1))` and `sd = sqrt(r(variance))`.

tags is a dimension and its corresponding level, or it is a space-separated list of dimensions and their corresponding levels:

dimname`[`*levvalue*`]` [*dimname*`[`*levvalue*`]` ...]

If *levvalue* contains spaces, it must be enclosed in double quotes.

Options

Main

`name(`*cname*`)` specifies the collection into which results will be saved, instead of the current collection.

Options

`tags(`*tags*`)` specifies additional tags to attach to the results being consumed. Each tag takes on the form *dimname* `[`*levvalue*`]`; to specify multiple tags separate them with a space. The following dimension names are not allowed in `tags()`: `border_block`, `cell_type`, `program_class`, and `result_type`.

`commands(`*commands*`)` specifies command names that posted the results being consumed but does not change what results are collected. This option exists to aid the search for command-specific result labels.

Unlike the `collect:` prefix, `collect get` cannot determine the command that posted the results being collected, so this option allows you to help it search for command-specific result labels. This option is also allowed by the `collect:` prefix; this is useful, for example, when the prefixed *command* is a custom program that employs official Stata commands to post its results.

Remarks and examples

Remarks are presented under the following headings:

Introduction

The first step in creating a table using `collect` is to collect the results that you wish to display in the table from Stata commands. The `collect get` command and the `collect` prefix consume results from a Stata command and place them in a collection.

To collect results, we can type either

```
.  command
. collect get  results
```

or

```
. collect: command
```

These two methods are almost equivalent. They differ in how they determine which results are to be collected. The `collect` prefix determines which results are stored by the *command* and puts all of their values into the collection. On the other hand, when `collect get` is used after *command*, we must specify results that are to be collected. We can do this in a generic way. For instance, to fit a regression model with *command* and collect results, we can type

```
.  command
. collect get e()
```

and all results stored in `e()` will be collected. However, we can be more specific about the results to be collected. If we wish to include coefficients, referred to as `_r_b`, and standard errors, referred to as `_r_se`, in the tables we are about to create, we can indicate this by typing

```
.  command
. collect get _r_b _r_se
```

All the results in `e()` will still be added to the collection. The difference is that we have now flagged `_r_b` and `_r_se` as results to be automatically included in the tables we create. For instance, if we now specify a table layout by typing

```
. collect layout (colname) (result)
```

the coefficients and standard errors will appear in the table. See [TABLES] **collect layout** for information on specifying the table layout. For our purposes here, the important issue is that we have specified that the statistics, denoted by `result`, are going to be placed on the columns. Because we specified `_r_b` and `_r_se` in our `collect get` command, these are the two statistics that will be reported. We refer to these selected levels as “automatic levels”. The automatic levels simplify the table layout specification, but we are not limited by the results we flag as automatic levels. If we, for instance, decide that we want to include confidence intervals (`_r_ci`) instead of standard errors, we can state this directly in our `collect layout` command.

```
. collect layout (colname) (result[_r_b _r_ci])
```

Above, we selected automatic levels using `collect get` after fitting a model. We can similarly select automatic levels with the `collect` prefix. For instance, we can type

```
. collect _r_b _r_se: command
```

The effects of specifying automatic levels with the prefix are the same as specifying automatic levels with the `collect get` command. Automatic levels can be changed at any time via the command `collect style autolevels`; see [TABLES] **collect style autolevels**.

Support for other prefix commands

Some prefix commands are supported as a prefix to `collect`, others are supported as a prefix to the *command* being executed, and a few are not supported in either form. The following sections provide more details on which prefix commands are supported with the `collect` prefix.

Fully supported

The following prefix commands can be specified as a prefix to the `collect` prefix or as a prefix within *command*.

Command name	Entry
`by`	[D] **by**
`noisily`	[P] **quietly**
`quietly`	[P] **quietly**
`version`	[P] **version**

For example, you can type

```
by var1: collect: regress y x1 x2
```

or

```
collect: by var1, sort: regress y x1 x2
```

Likewise, `noisily`, `quietly`, and `version` may be specified before or after the `collect` prefix.

For the `by` prefix, the levels of the `by` variables are also collected as part of each results set. `by` is not allowed to be specified simultaneously as a prefix to the `collect` prefix and within *command*.

Partially supported

The following commands are allowed to be specified as a prefix to the `collect` prefix but are not allowed as a prefix in *command*.

Command name	Entry
`capture`	[P] **capture**
`frame`	[D] **frame prefix**
`xi`	[R] **xi**

For example, you can type

```
frame fr1: collect: regress y x1 x2
```

but not

```
collect: frame fr1: regress y x1 x2
```

The following commands are not allowed to be specified as a prefix to the `collect` prefix but are allowed as a prefix in *command*.

Command name	Entry
`bayes`	[BAYES] **bayes**
`bootstrap`	[R] **bootstrap**
`fmm`	[FMM] **fmm**
`fp`	[R] **fp**
`jackknife`	[R] **jackknife**
`mfp`	[R] **mfp**
`mi estimate`	[MI] **mi estimate**
`permute`	[R] **permute**
`svy`	[SVY] **svy**

For example, you can type

```
collect: bayes: regress y x1 x2
```

but not

```
bayes: collect: regress y x1 x2
```

Not supported

The following commands are not allowed to be specified as a prefix to the `collect` prefix, nor are they allowed as a prefix in *command*.

Command name	Entry
`nestreg`	[R] **nestreg**
`rolling`	[TS] **rolling**
`simulate`	[R] **simulate**
`statsby`	[D] **statsby**
`stepwise`	[R] **stepwise**

Collecting results from margins, contrast, and pwcompare

`margins`, `contrast`, and `pwcompare` return results in `r()` by default. They can also store results in `e()` when the `post` option is specified.

When the `post` option is specified, `collect get` and the `collect` prefix work just as they would with any estimation command.

However, when margins, contrast, and pwcompare are specified without post, their interaction with collect get and the collect prefix is unique. Specifically, if you use collect get after one of these commands, you will need to specify r() to indicate that stored results should be collected from r(), just as you would with any command that stores results in r(). However, you can also specify the following identifiers if you wish to collect some of the statistics reported in the table and add them to the list of automatic results.

Identifier	Result
_r_b	coefficients or transformed coefficients reported by *command*
_r_se	standard errors of _r_b
_r_z	test statistics for _r_b
_r_z_abs	absolute values of _r_z
_r_p	p-values for _r_b
_r_lb	lower bounds of confidence intervals for _r_b
_r_ub	upper bounds of confidence intervals for _r_b
_r_ci	confidence intervals for _r_b
_r_df	degrees of freedom for _r_b

If you use the collect prefix with margins, contrast, and pwcompare, it can determine whether post was specified and, thus, whether results should be collected from r() or e(). In either case, the identifiers above can be specified with the collect prefix to add selected statistics reported by these commands to the list of automatic results.

Results not collected by default

Above, we told you that if you type

```
. collect: command
```

all results that *command* stores will be added to the collection. This is almost true. However, the following results are not stored by default.

```
e(b)
r(b)
e(V)
r(V)
e(b_fitorder)
e(b_idx)
e(b_keep)
e(b_sd)
e(Cns)
e(gauss_hasbeta)
e(gauss_V)
e(ilog)
e(gradient)
e(Jacobian)
e(table)
r(table)
e(V_modelbased)
e(V_sd)
e(V_vs)
```

In addition, any stored result with a name that begins with `datasignature`, `filename`, or `simtime` will not be collected by default.

Finally, any hidden results, the results you see only if you type

```
. ereturn list, all
```

or

```
. return list, all
```

are not collected by default.

You can collect these results if you specifically request them. You will need to give them a name at the time of collection. For instance, if you want to collect `e(V)` after a regression model, you can type

```
. collect V=e(V): regress y x
```

Also see

[TABLES] **collect create** — Create a new collection

[TABLES] **collect layout** — Specify table layout for the current collection

[TABLES] **collect set** — Set the current (active) collection

[TABLES] **collect unget** — Remove results from a collection

[R] **table intro** — Introduction to tables of frequencies, summaries, and command results

Description

`collect unget` removes results from a collection.

Quick start

Remove coefficients and standard errors identified with omitted predictor variable `x1`

```
collect unget _r_b _r_se, fortags(colname[o.x1])
```

Remove all frequency results

```
collect unget frequency
```

Remove frequency results that are tagged with any level of dimension `group`

```
collect unget frequency, fortags(group)
```

Remove frequency results that are tagged with level `_hide` of dimension `group`

```
collect unget frequency, fortags(group[_hide])
```

Remove frequency results that are tagged with `code1[first]` or `code2[last]`

```
collect unget frequency, fortags(code1[first] code2[last])
```

Remove frequency results that are tagged with `code1[first]` and `code2[last]`

```
collect unget frequency, fortags(code1[first]#code2[last])
```

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Remove results

Syntax

`collect unget` *results* [, *options*]

results is a list of one or more `result` levels that identify values to be removed from the collection.

options	Description
Main	
`name(`*cname*`)`	remove results from collection *cname*
Options	
`fortags(`*taglist*`)`	only add tags to values with tags in *taglist*

Options

Main

`name(`*cname*`)` specifies the collection from which to remove results. If this option is not specified, the change is made in the current collection.

Options

`fortags(`*taglist*`)` specifies conditions for selecting the results to be removed from the collection. Results with tags in *taglist* will be removed from the collection.

Within the *taglist*, if tags are joined by `#`, results having all of these tags are selected; if tags are separated by a space, results with any of these tags are selected.

taglist contains

tagspec

tagspec taglist

tagspec contains

tag

tag`#`*tag*[`#`*tag*[...]]

tag contains

dimension

dimension`[`*levels*`]`

dimension is a dimension in the collection.

levels are levels of the corresponding dimension.

Distinguish between `[]`, which are to be typed, and [ ], which indicate optional arguments.

Remarks and examples

After collecting results, we may want to remove or replace some results to accommodate special requirements for the table we wish to create. `collect unget` allows you to remove results from a collection.

One example is a coefficient table with factor-variable coefficients, where we want to identify base levels with the string "`(base)`" in the column of standard errors.

To demonstrate, we use data from the Second National Health and Nutrition Examination Survey (NHANES II) (McDowell et al. 1981). We fit a model for systolic blood pressure, then arrange the fitted coefficients and standard errors into a table.

```
. use https://www.stata-press.com/data/r19/nhanes2l
(Second National Health and Nutrition Examination Survey)
. quietly: collect: regress bpsystol weight i.diabetes i.sex
. collect layout (colname) (result[_r_b _r_se])
Collection: default
      Rows: colname
   Columns: result[_r_b _r_se]
   Table 1: 6 x 2
```

	Coefficient	Std. error
Weight (kg)	.4340474	.0153533
Not diabetic	0	0
Diabetic	14.34115	1.019611
Male	0	0
Female	1.107633	.4710559
Intercept	98.40567	1.235476

We want to replace the zero-valued standard errors with the string "`(base)`". First, we use `collect unget` to remove the standard error results for the base levels of `diabetes` and `sex`. Then, we use `collect preview` to verify our intended change.

```
. collect unget _r_se , fortags(colname[0.diabetes 1.sex])
(2 items removed from collection default)
. collect preview
```

	Coefficient	Std. error
Weight (kg)	.4340474	.0153533
Not diabetic	0	
Diabetic	14.34115	1.019611
Male	0	
Female	1.107633	.4710559
Intercept	98.40567	1.235476

Now let's add some string results to replace the zero values we just removed.

```
. collect get _r_se="(base)", tag(colname[0.diabetes])
. collect get _r_se="(base)", tag(colname[1.sex])
. collect preview
```

	Coefficient	Std. error
Weight (kg)	.4340474	.0153533
Not diabetic	0	(base)
Diabetic	14.34115	1.019611
Male	0	(base)
Female	1.107633	.4710559
Intercept	98.40567	1.235476

Stored results

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

Macros

`s(collection)`	name of collection
`s(k_removed)`	number of items removed

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

[TABLES] **collect get** — Collect results from a Stata command

collect addtags — Add tags to items in a collection

Description

`collect addtags` adds tags to existing items in a collection.

Quick start

Add the tag `newdim[level1]` to values with the current tag `olddim[oldlevel]`

```
collect addtags newdim[level1], fortags(olddim[oldlevel])
```

Add the tag `newdim[level1]` to values with the current tags `dim1[level1]` and `dim2[level2]`

```
collect addtags newdim[level1], fortags(dim1[level1]#dim2[level2])
```

Replace the tag `dim1[oldlevel]` with `dim1[newlevel]`

```
collect addtags dim1[newlevel], fortags(dim1[oldlevel])
```

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Build and style table

Syntax

`collect addtags` *tags* [, *options*]

tags is a dimension and its corresponding level, or it is a space-separated list of dimensions and their corresponding levels:

dimname`[`*levvalue*`]` [*dimname*`[`*levvalue*`]` ...]

If *levvalue* contains spaces, it must be enclosed in double quotes.

options	Description
Main	
`name(`*cname*`)`	add tags to items in collection *cname*
`replace`	replace the items' tags when in conflict
Options	
`fortags(`*taglist*`)`	only add tags to values with tags in *taglist*

Options

Main

`name(`*cname*`)` specifies the collection in which to add tags. If this option is not specified, the change is made in the current collection.

`replace` specifies that the new tags replace the items' tags where they are in conflict.

Options

`fortags(`*taglist*`)` specifies conditions for selecting the values to which tags will be added. Values with tags in *taglist* will have the specified tags added to them.

Within the *taglist*, if tags are joined by `#`, values having all of these tags are selected; if tags are separated by a space, values with any of these tags are selected.

taglist contains

tagspec

tagspec taglist

tagspec contains

tag

tag`#`*tag*[`#`*tag*[...]]

tag contains

dimension

dimension`[`*levels*`]`

dimension is a dimension in the collection.

levels are levels of the corresponding dimension.

Distinguish between `[]`, which are to be typed, and [ ], which indicate optional arguments.

Remarks and examples

After collecting results, we may need to add tags to certain items to lay out the table that we wish to create. `collect addtags` allows you to add tags to existing items in a collection.

One instance where you may need to add tags is when you want to use different levels of a dimension to define both the rows and columns of a table. While you cannot use the same dimension to define both the rows and columns, you can add tags to certain levels, use the new dimension to identify the rows, and use the old dimension to identify the columns and vice versa.

To demonstrate, we use data from the Second National Health and Nutrition Examination Survey (NHANES II) (McDowell et al. 1981). We fit two models for systolic blood pressure:

```
. use https://www.stata-press.com/data/r19/nhanes2l
(Second National Health and Nutrition Examination Survey)
. quietly: collect: regress bpsystol weight i.diabetes i.sex
. quietly: collect: regress bpsystol weight diabetes##sex
```

Our goal is to create the following table, in which coefficients and standard errors are placed on separate columns and model statistics are placed on rows beneath the covariates.

	1		2	
Weight (kg)	0.43	(0.02)	0.43	(0.02)
Diabetic	14.34	(1.02)	12.57	(1.54)
Female	1.11	(0.47)	0.95	(0.48)
Diabetic # Female			3.15	(2.05)
Intercept	98.41	(1.24)	98.52	(1.24)
R-squared	0.0998		0.1000	
N	10349		10349	

The top portion of the table has rows defined by covariate names (`colname`) and columns defined by levels `_r_b` and `_r_se` of dimension `result`, for each model. To align the `N` and `R-squared` values as shown above, they need to be tagged with level `_r_b` of dimension `result` and with levels of another dimension that we can place on the rows.

First, we will add tags to the model statistics corresponding with a new dimension called `extra`.

```
. collect addtags extra[Rsquared], fortags(result[r2])
(2 items changed in collection default)
. collect addtags extra[N], fortags(result[N])
(2 items changed in collection default)
```

The R^2 value is currently tagged with level `r2` of dimension `result`, and now it is also tagged with level `Rsquared` of dimension `extra`. Similarly, the number of observations for each model is now tagged with level `N` of dimension `extra`.

To place these statistics in the same column as the coefficients, we `recode` them so that they are now tagged with the level `_r_b` of dimension `result`.

```
. collect recode result r2=_r_b N=_r_b
(4 items recoded in collection default)
```

Now we can lay out our table with the covariates (`colname`) and extra statistics (`extra`) on the rows and the results for each command on the columns:

```
. collect layout (colname extra) (cmdset#result[_r_b _r_se])
Collection: default
      Rows: colname extra
   Columns: cmdset#result[_r_b _r_se]
   Table 1: 12 x 4
```

	1 Coefficient	1 Std. error	2 Coefficient	2 Std. error
Weight (kg)	.4340474	.0153533	.4335342	.0153559
Not diabetic	0	0	0	0
Diabetic	14.34115	1.019611	12.57211	1.538361
Male	0	0	0	0
Female	1.107633	.4710559	.9520999	.4817911
Not diabetic # Male			0	0
Not diabetic # Female			0	0
Diabetic # Male			0	0
Diabetic # Female			3.146466	2.048958
Intercept	98.40567	1.235476	98.5238	1.237787
Rsquared	.0997888		.099994	
N	10349		10349	

The curious reader might wonder why we did not simply tag these statistics with levels of dimension `colname`. By adding them to a new dimension, we are able to place them at the end of the table, after all the covariates.

We obtained the layout we want, and now we will use a few `collect style` commands to polish the table. First, we load one of the predefined styles that are installed with Stata. This style hides the labels for the levels of dimension `result`, and it centers headers that are repeated across columns. Then, we use `collect style showbase` to hide all the base levels from the table.

```
. collect style use table-reg2-fv1
Collection: default
      Rows: colname extra
   Columns: cmdset#result[_r_b _r_se]
   Table 1: 9 x 4
. collect style showbase off
```

Next, we define a label for the R^2 value:

```
. collect label levels extra Rsquared "R-squared"
```

Below, we format the coefficients to display two digits after the decimal and align them to the right. We also place the standard errors in parentheses.

```
. collect style cell result[_r_b], halign(right) nformat(%6.2f)
. collect style cell result[_r_se], halign(left) nformat(%6.2f) sformat("(%s)")
```

Then, we format the model statistics and align them to the right.

```
. collect style cell extra[Rsquared]#result, halign(right) nformat(%6.4f)
. collect style cell extra[N]#result, halign(right) nformat(%4.0f)
```

Finally, we add a border below the constant to separate the R^2 value and number of observations from the rest of the table.

```
. collect style cell colname[_cons], border(bottom, pattern(single))
```

After these style changes, we have the following table:

```
. collect preview
```

	1		2	
Weight (kg)	0.43	(0.02)	0.43	(0.02)
Diabetic	14.34	(1.02)	12.57	(1.54)
Female	1.11	(0.47)	0.95	(0.48)
Diabetic # Female			3.15	(2.05)
Intercept	98.41	(1.24)	98.52	(1.24)
R-squared	0.0998		0.1000	
N	10349		10349	

Stored results

collect addtags stores the following in s():

Macros

s(collection)	name of collection
s(k_changed)	number of changed items

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

[TABLES] **collect remap** — Remap tags in a collection

[TABLES] **collect recode** — Recode dimension levels in a collection

collect clear — Clear all collections in memory

Description

collect clear clears all collection information from memory. clear collect is a synonym for collect clear.

Syntax

```
collect clear
```

Remarks and examples

collect clear clears all collection information from memory.

Results collected using the collect prefix or the collect get command are stored in memory. To see the collections currently in memory, you can type collect dir. collect clear removes all of these collections from memory. If you wish to remove only some of the collections from memory, see collect drop and collect keep.

Also see

[TABLES] **collect get** — Collect results from a Stata command

[TABLES] **collect drop** — Drop collections from memory

collect combine — Combine collections

Description

`collect combine` combines separate collections into a single new collection.

Quick start

Create new collection `newc` by combining existing collections `c1`, `c2`, and `c3`

```
collect combine newc = c1 c2 c3
```

Same as above, but use the layout defined in the rightmost collection, `c3`

```
collect combine newc = c1 c2 c3, layout(right)
```

Same as above, but use the style defined in the rightmost collection, `c3`

```
collect combine newc = c1 c2 c3, layout(right) style(right)
```

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Combine collections

Syntax

`collect combine` *newcname* = *cnamelist* [, *options*]

where *newcname* is the name of the new collection and *cnamelist* is a list of names of existing collections.

options	Description
Main	
`replace`	overwrite *newcname* if it exists
Options	
`layout(left`|`right)`	specify the collection whose layout is to be used; default is `layout(left)`
`style(left`|`right)`	specify the collection whose style is to be used; default is `style(left)`
`label(left`|`right)`	specify the collection whose labels are to be used; default is `label(left)`
[`no`]`warn`	display notes when encountering unrecognized tags

Options

Main

`replace` permits `collect combine` to overwrite *newcname* if it already exists. This option is required if the new collection already exists and is not empty.

Options

`layout(left`|`right)` specifies the collection whose layout will be used in the new collection.

`layout(left)` is the default; it applies the layout from the leftmost collection in *cnamelist* to the new collection.

`layout(right)` applies the layout from the rightmost collection in *cnamelist* to the new collection.

`style(left`|`right)` specifies the collection whose style will be used in the new collection.

`style(left)` is the default; it applies the style definitions from the leftmost collection in *cnamelist* to the new collection.

`style(right)` applies the style definitions from the rightmost collection in *cnamelist* to the new collection.

`label(left`|`right)` specifies the collection whose labels will be used in the new collection.

`label(left)` is the default; it applies the labels from the leftmost collection in *cnamelist* to the new collection.

`label(right)` applies the labels from the rightmost collection in *cnamelist* to the new collection.

`warn` and `nowarn` control the display of notes when `collect` encounters a tag it does not recognize.

`warn`, the default, specifies that `collect` show the notes.

`nowarn` specifies that `collect` not show the notes.

These options override the `collect_warn` setting; see [TABLES] **set collect_warn**.

Remarks and examples

`collect combine` combines existing collections into a new collection. The new collection becomes the current collection.

The `label()`, `layout()`, and `style()` options specify whether `collect` should apply the labels, layout, and style from the leftmost or rightmost collection specified. The default is to apply the style, layout, and labels from the leftmost collection in *cnamelist* to the new collection. This is equivalent to specifying `label(left)`, `layout(left)`, and `style(left)`. If any of these attributes is not defined in the leftmost collection, `collect` will search for that attribute in the collections listed in *cnamelist*, from left to right. However, if the rightmost collection is specified with any of these options, and that attribute is not defined in the rightmost collection, `collect` will search for that attribute in the collections listed, from right to left.

For example, we create a collection called `new` by combining the collections `c1`, `c2`, `c3`, and `c4`.

```
. collect combine new = c1 c2 c3 c4
```

If collection `c1` has an empty style, `collect` will apply the style from `c2` to the new collection. If `c2` also has an empty style, `collect` will apply the style from `c3`.

Suppose that we instead type the following:

```
. collect combine new = c1 c2 c3 c4, style(right)
```

`collect` will apply the style from collection `c4` to the collection `new`. If collection `c4` has an empty style, `collect` will apply the style from `c3` to the new collection.

Stored results

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

Macros

`s(current)`	name of new collection
`s(collections)`	list of combined collections

Also see

[TABLES] **collect use** — Use a collection from disk

[TABLES] **collect save** — Save a collection to disk

collect composite — Manage composite results in a collection

Description | Quick start | Menu | Syntax
Options | Remarks and examples | Reference | Also see

Description

`collect composite` manages the creation and removal of composite results in a collection. A composite result is composed from a list of levels in dimension `result`.

Quick start

Define a result composed from a coefficient and its standard error estimate

```
collect composite define bse = _r_b _r_se
```

Drop the above composite result

```
collect composite drop bse
```

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Build and style table

Syntax

Define a composite result

collect composite define *composite* = *elements* [, *options*]

Drop a composite result

collect composite drop *composite* [, name(*cname*)]

composite is the name for a composite result.

elements are levels in the result dimension and previously defined composite results.

options	Description
name(*cname*)	add composite result definition to collection *cname*
delimiter(*char*)	use character as delimiter between *elements*
[no]trim	preserve or trim extra spaces from numeric formats
[no]override	preserve or override trim property of elements
replace	overwrite *composite* if it already exists

Options

name(*cname*) specifies the collection to add or remove the composite result. By default, changes are applied to the current collection.

delimiter(*char*) changes the delimiter between elements. The default is delimiter(" ").

notrim and trim control the handling of extra spaces caused by numeric formats applied to the elements. notrim, the default, preserves the extra spaces; trim removes the extra spaces.

Note that the trim and notrim options do not apply to results _r_ci and _r_cri.

nooverride and override control handling of the trim property when an element is a composite result. nooverride, the default, does not change the trim property of elements; override applies the specified trim property to all elements.

replace permits collect composite define to overwrite the definition of an existing composite result.

Remarks and examples

collect composite is used to create a single result from multiple results. This can be useful when you want to display multiple results in a single cell. For example, you may want to place standard deviations right next to means or confidence intervals right next to coefficients. Any formatting applied to the individual results will persist in the composite result. For example, you may format confidence intervals by enclosing them in brackets. If you then create a composite result with coefficients and confidence intervals, the result will contain coefficients followed by the confidence intervals enclosed in brackets.

You can also build a result from existing composite results. For example, you might first decide to create a composite result from means and standard deviations and call it meansd. As you are working, you may decide you want to add on medians to the composite result. You can then create a new composite result with meansd and the medians.

Remarks are presented under the following headings:

Example 1: Table of means and standard deviations

We have data from the Second National Health and Nutrition Examination Survey (NHANES II) (McDowell et al. 1981), and we want to create a table reporting the mean and standard deviation of systolic blood pressure, cholesterol, and triglycerides for males and females. Additionally, we want to display the p-value for the mean-comparison test.

```
. use https://www.stata-press.com/data/r19/nhanes2l
(Second National Health and Nutrition Examination Survey)
```

We can use the ttest command to perform a mean-comparison test and then create a table with its stored results. The table command allows us to issue multiple commands and create a table with their results in a single step. For example, below we create a table with only two variables of interest to prevent the table from wrapping. We specify each ttest command in the command() option, and we place results from each command on a separate column. While ttest reports multiple statistics, we will display only the means (mu_1 and mu_2) and p-values.

```
. table (result[mu_1 mu_2 p]) (command),
> command(ttest bpsystol, by(sex))
> command(ttest tcresult, by(sex))
```

	ttest bpsystol, by(sex)	ttest tcresult, by(sex)
x_1 mean for population 1	132.8877	213.1731
x_2 mean for population 2	129.0679	221.7353
Two-sided p-value	8.03e-17	1.10e-18

Notice that the commands are used to label the output for each variable. We will now create our table with all three variables, but we will use the quietly prefix to suppress our output:

```
. quietly: table () (command),
> command(ttest bpsystol, by(sex))
> command(ttest tcresult, by(sex))
> command(ttest tgresult, by(sex))
```

Below, we use collect label levels to label the output from each command with the variable label. Then, we use collect style header to suppress the labels for the results; we will modify them shortly.

```
. collect label levels command  1 "Systolic BP"
> 2 "Cholesterol (mg/dL)" 3 "Triglycerides (mg/dL)", modify
. collect style header result, level(hide)
```

Next, we will format all results to two decimal places and the p-values to three. Then, we will lay out our table with the results on the columns and the variables on the rows. Each command corresponds to a different variable, so we specify `command` as the row identifier. We are interested only in the means, standard deviations, and p-values, so we list the levels of `result` that we want in our table.

```
. collect style cell result, nformat(%6.2f)
. collect style cell result[p], nformat(%7.3f)
. collect layout (command) (result[mu_1 sd_1 mu_2 sd_2 p])
Collection: Table
      Rows: command
   Columns: result[mu_1 sd_1 mu_2 sd_2 p]
   Table 1: 3 x 5
```

Systolic BP	132.89	20.99	129.07	25.13	0.000
Cholesterol (mg/dL)	213.17	45.98	221.74	51.95	0.000
Triglycerides (mg/dL)	154.63	112.30	133.85	77.58	0.000

We would like to display the standard deviations right next to the means, using a plus–minus sign as the delimiter. Below, we combine the means and standard deviations for males (`mu_1` and `sd_1`) into one result and the means and standard deviations for females (`mu_2` and `sd_2`) into another.

The means and standard deviations have a numeric format of `%6.2f`, which means a total output width of 6 with only 2 digits displayed after the decimal. However, you will notice that all the standard deviations for females in the above fourth column have a total width of five: two digits before the decimal, two digits after, and the decimal point. This means we will have an extra space between the plus–minus sign and those standard deviations. Therefore, we use the `trim` option to remove those extra spaces.

Then, we label the results as `Males` and `Females` and modify the label for the p-values. Now that we have customized labels, we use `collect style header` to display the labels for our results.

```
. collect composite define msd1 = mu_1 sd_1, delimiter(±) trim
. collect composite define msd2 = mu_2 sd_2, delimiter(±) trim
. collect label levels result msd1 "Males" msd2 "Females" p "p-value", modify
. collect style header result, level(label)
```

Finally, we can lay out our table using our composite results, `msd1` and `msd2`:

```
. collect layout (command) (result[msd1 msd2 p])
Collection: Table
      Rows: command
   Columns: result[msd1 msd2 p]
   Table 1: 3 x 3
```

	Males	Females	p-value
Systolic BP	132.89±20.99	129.07±25.13	0.000
Cholesterol (mg/dL)	213.17±45.98	221.74±51.95	0.000
Triglycerides (mg/dL)	154.63±112.30	133.85±77.58	0.000

We could customize this table further by, for example, modifying how p-values are displayed and removing some borders, but our focus here is solely on displaying these combined results.

Example 2: Table of means, medians, standard deviations, and confidence intervals

Suppose we want to create a table similar to the one above, but instead of p-values, we now want to include medians and confidence intervals for the means. We can obtain these statistics from the `summarize` and `ci means` commands. First, we clear the collection information in memory:

```
. collect clear
```

To create our table, we loop over our variables of interest and the values of sex (1 and 2). We will use `collect get` to collect the stored results of interest from each command.

First, we obtain our means and standard deviations from `ttest`. Recall that `ttest` stores the means and standard deviations for the groups separately, in `mu_1` and `sd_1` and `mu_2` and `sd_2`. We wish to store all the means in one result, `mymean`, and all the standard deviations in another result, `mysd`. To identify whether they belong to males or females, we will tag each result with the variable and level of sex.

Next, we will collect the upper (`r(ub)`) and lower (`r(lb)`) bounds of the confidence intervals from `ci means` and the median from `summarize`. Note that we also tag these results with the variable and level of sex.

```
. foreach x of varlist bpsystol tcresult tgresult {
  2.     ttest `x', by(sex)
  3.     collect get mymean=r(mu_1) mysd=r(sd_1), tags(var[`x'] sex[1])
  4.     collect get mymean=r(mu_2) mysd=r(sd_2), tags(var[`x'] sex[2])
  5.
.       forvalues i = 1/2 {
  6.         ci means `x' if sex==`i'
  7.         collect get r(lb) r(ub), tags(var[`x'] sex[`i'])
  8.
.           summarize `x' if sex==`i', detail
  9.         collect get r(p50), tags(var[`x'] sex[`i'])
 10.     }
 11. }
  (output omitted)
```

We have now collected all of our results, and we can begin building our table. First, let's create a composite result from the upper and lower bounds of the confidence intervals. This result, `meanci`, will use a dash as the delimiter:

```
. collect composite define meanci = lb ub, delimiter(-) trim
```

We will format all results to two decimal places and center them. Then, we will place the confidence intervals in brackets:

```
. collect style cell result, nformat(%6.2f) halign(center)
. collect style cell result[meanci], sformat("[%s]")
```

Next, we will join the means and standard deviations, as we did with the previous example. Before we combine the medians with the composite result, `meansd`, we enclose them in parentheses. We then create a new composite result called `msdmed`.

```
. collect composite define meansd = mymean mysd, delimiter(±) trim
. collect style cell result[p50], sformat("(%s)")
. collect composite define msdmed = meansd p50
```

Before we lay out our table, we will hide the labels for the results with `collect style header`. Then, we will use `collect layout` to lay out our table with the mean, standard deviation, and median for each variable on one row and the confidence interval on the following row. The columns will be defined by the levels of `sex`.

```
. collect style header result, level(hide)
. collect layout (var#result[msdmed meanci]) (sex)
Collection: default
      Rows: var#result[msdmed meanci]
   Columns: sex
   Table 1: 6 x 2
```

	Male	Female
Systolic blood pressure	132.89±20.99 (130.00)	129.07±25.13 (124.00)
	[132.30-133.47]	[128.40-129.74]
Serum cholesterol (mg/dL)	213.17±45.98 (209.00)	221.74±51.95 (217.00)
	[211.89-214.46]	[220.35-223.12]
Serum triglycerides (mg/dL)	154.63±112.30 (128.00)	133.85±77.58 (115.00)
	[150.17-159.09]	[130.87-136.83]

Our table looks great; we just need to modify the headers. Below, we center the column headers, add a descriptive title, and then preview our table:

```
. collect style cell cell_type[column-header], halign(center)
. collect title "Mean, SD, median, and confidence interval values for health me
> asures"
. collect preview
```

Mean, SD, median, and confidence interval values for health measures

	Male	Female
Systolic blood pressure	132.89±20.99 (130.00)	129.07±25.13 (124.00)
	[132.30-133.47]	[128.40-129.74]
Serum cholesterol (mg/dL)	213.17±45.98 (209.00)	221.74±51.95 (217.00)
	[211.89-214.46]	[220.35-223.12]
Serum triglycerides (mg/dL)	154.63±112.30 (128.00)	133.85±77.58 (115.00)
	[150.17-159.09]	[130.87-136.83]

Example 3: Table of regression results

Creating composite results can also be useful when creating a table of regression results. For example, we wish to create a table with the coefficients, confidence intervals, and p-values from a linear regression model. Below, we create a new collection and collect the results from our model:

```
. collect create regress
(current collection is regress)
. quietly: collect: regress bpsystol i.agegrp bmi i.sex
```

Before laying out our table, we format the results to two decimal places and center the coefficients and p-values. We also place the confidence intervals in brackets and use a comma to separate the upper and lower bounds.

```
. collect style cell result[_r_b _r_p], nformat(%5.2f) halign(center)
. collect style cell result[_r_ci], nformat(%9.2f) sformat("[%s]")
> cidelimiter(,)
```

Now that we have formatted our confidence intervals and coefficients, we can combine them into a single result. We create a composite result called `coefci` and trim the extra spaces resulting from the numeric format. Then, we lay out our table with our composite result and the p-values.

```
. collect composite define coefci = _r_b _r_ci, trim
. collect layout (colname) (result[coefci _r_p])
Collection: regress
      Rows: colname
   Columns: result[coefci _r_p]
   Table 1: 10 x 2
```

	Coefficient [95% CI]	p-value
20-29	0.00	
30-39	0.87 [-0.37, 2.10]	0.17
40-49	6.69 [5.36, 8.03]	0.00
50-59	14.75 [13.42, 16.09]	0.00
60-69	20.99 [19.91, 22.06]	0.00
70+	28.00 [26.55, 29.45]	0.00
Body mass index (BMI)	1.36 [1.28, 1.44]	0.00
Male	0.00	
Female	-4.07 [-4.82, -3.32]	0.00
Intercept	87.08 [85.03, 89.12]	0.00

Notice that we have a lot of extra space preceding the upper bounds of the confidence intervals. We set the output width to 9 (`%9.2f`), but our bounds require at most a width of 5. We chose this format for `_r_ci` to illustrate that the `trim` option does not work with this result or with `_r_cri` if you are working with credible intervals. The reason is that the `collect` system has already composed these from upper and lower bounds; in a sense, these are composite results. To prevent having this extra space, we have two options. One option is to specify a smaller width, perhaps `%5.2f`, for our intervals. When creating your own tables, you may find that several results have a numeric format that is too large. You can use `collect query cell` to query the appearance styles for your results. The other option is to build your own composite result with the confidence intervals to which you can then apply the `trim` option. This is the method we will use. Below, we create a new composite result with the lower (`_r_lb`) and upper (`_r_ub`) bounds and then enclose these results in brackets.

```
. collect composite define myci = _r_lb _r_ub, delimiter(,)
. collect style cell result[myci], nformat(%9.2f) sformat("[%s]")
```

Now, we can create our composite result with the coefficients and composite result; we call it `coefci2`. We `trim` these results and specify the `override` option; this allows us to apply this trim property to the composite result, `myci`. Then, we label `coefci2` by referring to `__LEVEL__`, which is the confidence level that is recorded when results are consumed; in our case, the level is 95, but our notation avoids us having to check whether that is the case. If you issue `collect label list result`, you will see that this notation is used for `_r_lb`, `_r_ub`, and `_r_ci`. After labeling our new result, we lay out our table with the new composite result:

```
. collect composite define coefci2 = _r_b myci, trim override
. collect label levels result coefci2 "Coef. [__LEVEL__% CI]"
. collect layout (colname) (result[coefci2 _r_p])
Collection: regress
      Rows: colname
   Columns: result[coefci2 _r_p]
   Table 1: 10 x 2
```

	Coef. [95% CI]	p-value
20-29	0.00	
30-39	0.87 [-0.37,2.10]	0.17
40-49	6.69 [5.36,8.03]	0.00
50-59	14.75 [13.42,16.09]	0.00
60-69	20.99 [19.91,22.06]	0.00
70+	28.00 [26.55,29.45]	0.00
Body mass index (BMI)	1.36 [1.28,1.44]	0.00
Male	0.00	
Female	-4.07 [-4.82,-3.32]	0.00
Intercept	87.08 [85.03,89.12]	0.00

Here we can see that the intervals were in fact trimmed of the extra spaces.

Combining our coefficients and confidence intervals into a single result is particularly useful when displaying results from multiple models. For example, below we refit our model but add indicator variables for `race`. Then, we lay out our table once more:

```
. quietly: collect: regress bpsystol i.agegrp bmi i.sex i.race
. collect layout (colname) (cmdset#result[coefci2 _r_p])
Collection: regress
      Rows: colname
   Columns: cmdset#result[coefci2 _r_p]
   Table 1: 13 x 4
```

	1 Coef. [95% CI]	1 p-value	2 Coef. [95% CI]	2 p-value
20-29	0.00		0.00	
30-39	0.87 [-0.37,2.10]	0.17	0.93 [-0.30,2.16]	0.14
40-49	6.69 [5.36,8.03]	0.00	6.80 [5.46,8.13]	0.00
50-59	14.75 [13.42,16.09]	0.00	14.84 [13.51,16.18]	0.00
60-69	20.99 [19.91,22.06]	0.00	21.12 [20.04,22.20]	0.00
70+	28.00 [26.55,29.45]	0.00	28.12 [26.67,29.58]	0.00
Body mass index (BMI)	1.36 [1.28,1.44]	0.00	1.35 [1.27,1.42]	0.00
Male	0.00		0.00	
Female	-4.07 [-4.82,-3.32]	0.00	-4.08 [-4.82,-3.33]	0.00
White			0.00	
Black			2.61 [1.39,3.84]	0.00
Other			2.07 [-0.65,4.78]	0.14
Intercept	87.08 [85.03,89.12]	0.00	87.01 [84.96,89.06]	0.00

Note that we did not have to create another composite result for the coefficients and confidence intervals for the second model we fit. The composite result (`coefci2`) will be built from all existing values of `_r_b`, `_r_lb`, and `_r_ub`.

We can give the table a neater look with just a couple of modifications. First, we suppress the base levels from the output. Then, we specify that duplicate headers should be displayed only once and centered. Finally, we label the group of results from each set of commands as `Model 1` and `Model 2` and preview our table:

```
. collect style showbase off
. collect style column, dups(center)
. collect label levels cmdset 1 "Model 1" 2 "Model 2"
. collect preview
```

	Model 1			Model 2		
	Coef.	[95% CI]	p-value	Coef.	[95% CI]	p-value
30-39	0.87	[-0.37,2.10]	0.17	0.93	[-0.30,2.16]	0.14
40-49	6.69	[5.36,8.03]	0.00	6.80	[5.46,8.13]	0.00
50-59	14.75	[13.42,16.09]	0.00	14.84	[13.51,16.18]	0.00
60-69	20.99	[19.91,22.06]	0.00	21.12	[20.04,22.20]	0.00
70+	28.00	[26.55,29.45]	0.00	28.12	[26.67,29.58]	0.00
Body mass index (BMI)	1.36	[1.28,1.44]	0.00	1.35	[1.27,1.42]	0.00
Female	-4.07	[-4.82,-3.32]	0.00	-4.08	[-4.82,-3.33]	0.00
Black				2.61	[1.39,3.84]	0.00
Other				2.07	[-0.65,4.78]	0.14
Intercept	87.08	[85.03,89.12]	0.00	87.01	[84.96,89.06]	0.00

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

[TABLES] **collect label** — Manage custom labels in a collection

[TABLES] **collect levelsof** — List levels of a dimension

[TABLES] **collect style cell** — Collection styles for cells

[TABLES] **collect query** — Query collection style properties

collect copy — Copy a collection

Description

`collect copy` copies a collection in memory into a new collection.

Quick start

Copy existing collection `c1` into collection `c2`

```
collect copy c1 c2
```

Same as above, but overwrite `c2` if it exists

```
collect copy c1 c2, replace
```

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Copy collection

Syntax

`collect copy` *cname newcname* [, `replace`]

where *cname* is the name of an existing collection and *newcname* is the name of the collection to be created.

Option

`replace` permits `collect copy` to overwrite an existing collection. `replace` is required if *newcname* already exists and is not empty.

Remarks and examples

`collect copy` copies an existing collection into a new collection. This new collection becomes the current collection, the collection to which subsequent `collect` subcommands will be applied.

Stored results

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

Macros

`s(current)`	name of current collection

Also see

[TABLES] **collect create** — Create a new collection

[TABLES] **collect drop** — Drop collections from memory

collect create — Create a new collection

Description

`collect create` creates a new, empty collection.

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Create collection

Syntax

`collect create` *newcname* [, `replace`]

where *newcname* is the name of the new collection.

Option

`replace` permits `collect create` to overwrite an existing collection.

Remarks and examples

`collect create` creates a new, empty collection. This new collection becomes the current collection. After creating the collection, you can use `collect get` or the `collect` prefix to store results from Stata commands into the new collection and then build, customize, and export tables.

Stored results

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

Macros
`s(current)` name of current collection

Also see

[TABLES] **collect combine** — Combine collections

[TABLES] **collect copy** — Copy a collection

collect dims — List dimensions in a collection

Description

collect dims lists the dimensions in a collection.

Syntax

collect dims [, name(*cname*)]

Option

name(*cname*) specifies the collection for which dimensions should be listed, instead of the current collection.

Remarks and examples

After you use the collect get command or collect prefix, the values stored from the command results into the collection are categorized according to their tags. For example, a regression coefficient of 5.36 on variable x1 would have tags including result[_r_b] and colname[x1]. Here result and colname are known as dimensions, and they contain the type of results and the covariate names respectively. Within each dimension, there are multiple levels. These tags correspond to the _r_b level of the result dimension and the x1 level of the colname dimension.

Once you have collected results, you can see a list of all the dimensions in your collection using collect dims. For instance, after typing

```
. use https://www.stata-press.com/data/r19/nhanes2
. collect _r_b _r_se: regress bpsystol age weight i.region i.sex
  (output omitted)
```

you see a list of dimensions as follows:

```
. collect dims

Collection dimensions
Collection: default
-----------------------------------------
                   Dimension   No. levels
-----------------------------------------
Layout, style, header, label
                      cmdset   1
                       coleq   1
                     colname   9
           colname_remainder   1
               program_class   1
                      region   4
                      result   33
                 result_type   3
                     rowname   1
                         sex   2
Style only
                border_block   4
                   cell_type   4
-----------------------------------------
```

These are the dimensions in your collection. You will often need to know their names to specify them in other `collect` subcommands. The output is divided into sections, which tell you the types of `collect` subcommands that each dimension will be useful with.

For example, when arranging the collected values into a table by using `collect layout`, you can look at the section of the output labeled `Layout, style, header, label` to determine which dimensions can be used with this command. To build a table, you specify the dimensions that correspond to the rows and columns of your table. With this collection, you could type

```
. collect layout (colname) (result)
```

After you look at the list provided by `collect dims`, it might not have been obvious that you wanted `result` and `colname`. After learning the names of the dimensions, you may want to further explore each one. You can use `collect levelsof` to list the levels of a particular dimension. You can also use `collect label list` to list the label for the dimension and the labels for its levels.

Occasionally, you may want to explore the dimensions of another collection without making it the current collection. `collect dims` with option `name()` lists the dimensions of the collection specified within this option.

Stored results

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

Macros

`s(collection)`	name of collection
`s(dimnames)`	list of dimension names in collection
`s(dimsizes)`	list of dimension sizes in collection

Reference

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/.

Also see

[TABLES] **collect label** — Manage custom labels in a collection

[TABLES] **collect levelsof** — List levels of a dimension

collect dir — Display names of all collections in memory

Description

`collect dir` lists all collections in memory and identifies the current collection. `collect` by itself is a synonym for `collect dir`.

Syntax

`collect dir`

`collect`

Remarks and examples

After creating one or more collections, you can use `collect dir` to learn about the collections you have in memory.

`collect dir` provides a report such as

```
. collect dir

Collections in memory
Current: means

-------------------------
       Name   No. items
-------------------------
    default   14
      means   12
regressions   134
-------------------------
```

Above the table, we find that `means` is the name of our current collection. In the table, we learn that we have three collections in memory named `default`, `means`, and `regressions`. For each of these collections, `collect dir` reports the number of items in the collection. These items are the values collected from results of Stata commands, values that we can include in the cells of a table that we construct.

By default, new results collected by the `collect` prefix or `collect get` command will be stored in the current collection. Likewise, new table layouts, styles, and labels will apply to the current collection. Also, any exported tables will be based on the current collection. If you would like to work with a different collection, you can use `collect set` to specify that another collection from this list become the current collection.

To learn about commands for exploring the contents of one of these collections, see [TABLES] **collect dims** and [TABLES] **collect levelsof**.

Also see

[TABLES] **collect dims** — List dimensions in a collection

[TABLES] **collect get** — Collect results from a Stata command

[TABLES] **collect set** — Set the current (active) collection

collect drop — Drop collections from memory

Description | Quick start | Menu | Syntax
Remarks and examples | Stored results | Also see

Description

`collect drop` eliminates collections from memory.

`collect keep` works the same way as `collect drop`, except that you specify the collections to be kept rather than the collections to be deleted.

Quick start

Drop collection `c1` from memory

    collect drop c1

Same as above, and drop collection `c2`

    collect drop c1 c2

Drop all collections from memory

    collect drop _all

Drop all collections, except for `c2`

    collect keep c2

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Drop or keep collections

Syntax

Drop specified collections

`collect drop` *cname* [*cname* [...]]

Drop all but the specified collections

`collect keep` *cname* [*cname* [...]]

where *cname* is the name of an existing collection.

Remarks and examples

Results collected using the `collect` prefix or the `collect get` command are stored in memory. To see the collections currently in memory, you can type `collect dir`. `collect drop` is used to remove selected collections from memory. Alternatively, `collect keep` removes all but the specified collections from memory.

When `collect drop` or `collect keep` drops the current collection from memory, a new collection named `default` will be created and will become the current collection.

You can drop all collections by typing `collect drop _all`. `collect clear` can also be used to remove all collections from memory. The difference between `collect clear` and `collect drop _all` is that `collect drop _all` will post the list of dropped collections to macro `s(collections)`.

Stored results

`collect drop` and `collect keep` store the following in `s()`:

Macros

`s(current)`	name of current collection
`s(collections)`	list of dropped collections

Also see

[TABLES] **collect clear** — Clear all collections in memory

collect label — Manage custom labels in a collection

Description

`collect label dim` defines the label for a dimension. If the label is specified as an empty string, any existing label for the dimension is removed.

`collect label levels` defines labels for the levels of a dimension. If the label is specified as an empty string, any existing label for the level is removed.

`collect label save` saves label definitions to the specified file.

`collect label use` adds the labels defined in an external file to a collection.

`collect label drop` drops the dimension label and the labels for its levels.

`collect label list` lists the dimension label and the labels for its levels.

All `collect label` subcommands operate on the current collection by default.

Quick start

In the current collection, label dimension `dim1` as "First dimension", overwriting the default label

```
collect label dim dim1 "First dimension", modify
```

Clear the label for dimension `dim1`

```
collect label dim dim1 ""
```

Redefine the labels for the levels of dimension `dim1`

```
collect label levels dim1 level1 "Level 1" level2 "Level 2", modify
```

Save all defined labels from the current collection to `mylabels.stjson` for use with other collections

```
collect label save mylabels
```

Same as above, but save labels from the collection `c2` to `mylabels2.stjson`

```
collect label save mylabels2, name(c2)
```

Drop the dimension label and level labels from dimension `dim1`

```
collect label drop dim1
```

List all labels for dimension `dim1`

```
collect label list dim1
```

Menu

collect label dim

Statistics > Summaries, tables, and tests > Tables and collections > Build and style table

collect label levels

Statistics > Summaries, tables, and tests > Tables and collections > Build and style table

collect label save

Statistics > Summaries, tables, and tests > Tables and collections > Collection labels > Save labels

collect label use

Statistics > Summaries, tables, and tests > Tables and collections > Collection labels > Use labels

collect label drop

Statistics > Summaries, tables, and tests > Tables and collections > Collection labels > Drop labels

collect label list

Statistics > Summaries, tables, and tests > Tables and collections > Collection labels > List labels

Syntax

Label a dimension

`collect label dim` *dim* "*label*" [, `name(`*cname*`)` `modify`]

Label dimension levels

`collect label levels` *dim level* "*label*" [*level* "*label*" ...]
[, `name(`*cname*`)` `modify` `replace`]

Save labels to disk

`collect label save` *filename* [, `name(`*cname*`)` `replace`]

Use labels from disk

`collect label use` *filename* [, `name(`*cname*`)` `modify` `replace`]

Drop labels for a dimension

`collect label drop` *dim* [, `name(`*cname*`)`]

List labels for a dimension

`collect label list` *dim* [, `name(`*cname*`)` `all`]

where *cname* is a collection name, *dim* is a dimension in the specified collection, and *level* is a level of this dimension.

If *filename* is specified without an extension, `.stjson` is assumed for both `collect label save` and `collect label use`. If *filename* contains embedded spaces, enclose it in double quotes.

Options

Options are presented under the following headings:

Options for collect label dim

`name(`*cname*`)` specifies the collection to which the dimension label is to be applied. By default, the dimension label is applied to the current collection.

`modify` specifies that an existing (nonempty) dimension label is to be modified. By default, `collect label dim` applies labels only to dimensions with empty labels.

Options for collect label levels

`name(`*cname*`)` specifies the collection to which the level labels are to be applied. By default, the level labels are applied to the current collection.

`modify` specifies that existing (nonempty) labels on dimension levels are to be modified. By default, new labels specified using `collect label levels` are applied only to levels without existing labels, and those with existing labels will remain unchanged.

`replace` specifies that all existing labels for the levels of dimension *dim* be dropped and only the new level labels be applied.

Options for collect label save

`name(`*cname*`)` specifies the collection from which labels are to be saved. By default, the labels from the current collection are saved.

`replace` permits `collect label save` to overwrite an existing file.

Options for collect label use

`name(`*cname*`)` specifies the collection to which the labels in *filename* will be applied. By default, the labels in *filename* will be applied to the current collection.

`modify` specifies that existing (nonempty) labels on dimensions and dimension levels are to be modified. By default, the labels in *filename* are applied only to dimensions and levels of dimensions without existing labels, and those with existing labels remain unchanged.

`replace` specifies that all existing labels on dimensions and dimension levels be dropped and only the new labels in *filename* be used.

Option for collect label drop

name(*cname*) specifies the collection for which labels are to be dropped. By default, labels are dropped for the dimension in the current collection.

Options for collect label list

name(*cname*) specifies the collection for which labels are to be listed. By default, labels are listed for the dimension in the current collection.

all specifies that all levels of *dim* be shown in the list, including those without a label.

Remarks and examples

Labels of dimensions and their levels allow us to quickly understand what values are being presented in a table.

When you use collect get or the collect prefix, some labels are automatically applied to dimensions and levels of dimensions in your collection. When you have variable and value labels on the variables in your dataset, those labels will be included in your collection as well, and they will be displayed in tables created from it. In addition, Stata provides default labels for dimensions such as the result dimension, which means that the statistics reported on the table will have meaningful labels. However, for a given table, it is often necessary to modify labels. For instance, value labels from the dataset may be longer than desired for a column header, or you may prefer a different description than the default provided by Stata for the statistics in your table. collect label allows you to make these changes.

To demonstrate, we use data from the Second National Health and Nutrition Examination Survey (NHANES II) (McDowell et al. 1981) to fit two models. We collect the coefficients (_r_b) and use the quietly prefix to suppress the output. Then, we arrange the items in our collection with collect layout. We place the variable names on the rows and the statistics (result) from each command (cmdset) on the columns:

```
. use https://www.stata-press.com/data/r19/nhanes2
. quietly: collect _r_b: regress bpsystol bmi
. quietly: collect _r_b: regress bpsystol bmi age
. collect layout (colname) (cmdset#result)
Collection: default
      Rows: colname
   Columns: cmdset#result
   Table 1: 3 x 2
```

	1 Coefficient	2 Coefficient
Body mass index (BMI)	1.656894	1.304128
Age (years)		.5883367
Intercept	88.56855	69.58451

Each covariate is a level of the dimension colname, and the levels of colname are labeled with the variable labels from the current dataset in memory, by default. The constant is labeled as Intercept. Say that we want to change that label to Constant. We can use collect label levels to change the label. We will first need to know the name of this level within the colname dimension. To determine the name, we can simply list all the levels of the dimension and their corresponding labels as follows:

```
. collect label list colname

  Collection: default
   Dimension: colname
       Label: Covariate names and column names
Level labels:
       _cons  Intercept
         age  Age (years)
         bmi  Body mass index (BMI)
```

Here we see the label for the dimension as well as each of its levels. By default, the dimension label is omitted from the table. We find that the name of the level for the constant is `_cons`. Because this level already has a label, we need the `modify` option with `collect label levels` to override the existing label. In the table above, we also see `1` and `2` in the column headers. These identify our two regression commands; they are the levels of the `cmdset` dimension, which are unlabeled. We could learn this by typing

```
. collect label list cmdset, all
```

We will label the columns as `Model 1` and `Model 2`. Because the levels of `cmdset` do not have labels, we do not need to specify any options. After applying our labels, we get a preview of the table:

```
. collect label levels colname _cons "Constant", modify

. collect label levels cmdset 1 "Model 1" 2 "Model 2"

. collect preview
```

	Model 1 Coefficient	Model 2 Coefficient
Body mass index (BMI)	1.656894	1.304128
Age (years)		.5883367
Constant	88.56855	69.58451

There may be certain labels that you find yourself modifying for many tables. For example, if you create many tables of estimation results, you might change the label `Coefficient` to simply `Coef.` in each table. Instead of doing this for each collection separately, you can save a set of labels with `collect label save`. Then, you can use `collect label use` with option `modify` to apply these labels to other collections.

Stored results

All `collect label` subcommands store the following in `s()`:

Macros

`s(collection)`	name of collection
`s(dimname)`	dimension name

`collect label list` additionally stores the following in `s()`:

Macros

`s(label)`	dimension label
`s(level#)`	level of dimension
`s(label#)`	label for level of dimension
`s(k)`	number of dimension levels with a label

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

[TABLES] **collect dims** — List dimensions in a collection

[TABLES] **collect levelsof** — List levels of a dimension

Description Quick start Syntax Option
Remarks and examples Stored results Reference Also see

Description

`collect levelsof` lists the levels of a dimension in a collection.

Quick start

List the levels of dimension `dim1` in the current collection

```
collect levelsof dim1
```

List the levels of dimension `dim2` in the collection `c2`

```
collect levelsof dim2, name(c2)
```

Syntax

`collect levelsof` *dim* [, `name(`*cname*`)`]

where *dim* is a dimension in the collection and *cname* is the name of a collection.

Option

`name(`*cname*`)` specifies the collection for which dimension levels are to be listed. If this option is not specified, the levels of the specified dimension will be listed for the current collection.

Remarks and examples

`collect levelsof` lists the levels of the specified dimension.

Exploring dimension levels is often helpful in the process of table building because many `collect` subcommands take levels of dimensions as arguments. For example, the list provided by `collect levelsof` can be useful when arranging values in the collection into a table by using `collect layout`. If you wish to report only a subset of the results that were collected, you can list the desired levels in `collect layout`, and you can first determine the names of those levels from the results of `collect levelsof`.

Stored results

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

Macros

`s(collection)`	name of collection
`s(dimname)`	specified dimension
`s(levels)`	list of levels for the specified dimension

Reference

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/.

Also see

[TABLES] **collect label** — Manage custom labels in a collection

[TABLES] **collect dims** — List dimensions in a collection

collect notes — Add table notes in a collection

Description | Quick start | Menu | Syntax
Options | Remarks and examples | Stored results | Reference
Also see

Description

`collect notes` manages the creation of table notes in a collection.

Quick start

Add a table note that describes the cell information for continuous variables

```
collect notes "mean (sd)"
```

Add a table note that describes the cell information for indicator variables

```
collect notes "count percent%"
```

Insert a note at the beginning of the list of notes

```
collect notes 0: "Intercept estimates are baseline odds."
```

Replace the first note with a custom note for labels on significant results

```
collect notes 1: "*** p<0.001, ** p<0.01, * p<0.05"
```

Clear all table notes

```
collect notes, clear
```

Remove the third note

```
collect notes 3:, clear
```

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Build and style table

Syntax

Add a table note at the end of the list of notes

collect notes "*string*" [, *options*]

Add, replace, or remove a table note in a specified position

collect notes #: ["*string*"] [, *options*]

Remove all table notes

collect notes, clear [name(*cname*)]

The table note is added to the end of the list of notes unless #: is specified.

When # is 0, the note is added to the beginning of the list of notes.

options	Description
name(*cname*)	specify custom table note for collection *cname*
clear	remove custom table note
fortags(*taglist*)	specify tags identifying tables for which to target the note

Options

name(*cname*) specifies the collection to which the table note is added or removed. By default, the change is applied to the current collection.

clear removes all notes or the note in the specified position.

fortags(*taglist*) specifies conditions for selecting which tables the note is added or removed.

Within the *taglist*, if tags are joined by #, tables having all of these tags are selected; if tags are separated by a space, tables with any of these tags are selected.

taglist contains

tagspec

tagspec taglist

tagspec contains

tag

tag#*tag*[#*tag*[. . .]]

tag contains

dimension

dimension[*levels*]

dimension is a dimension in the collection.

levels are levels of the corresponding dimension.

Distinguish between [], which are to be typed, and [ ], which indicate optional arguments.

Remarks and examples

Remarks are presented under the following headings:

Introduction

Many times, you will want to add notes to a table to provide the reader with crucial information, such as details about the sample or an explanation of the notation used in the table. With `collect notes`, you can add notes to your table with any information you feel is necessary. Notes will be added to the table in plain text, but you can customize the appearance style of the notes with `collect style notes`. For example, you can specify the font, text color, and more.

After adding several notes to a table, you may decide that one or more of those notes is not needed. You can remove all the notes in a collection or a specific note by referring to the number of the note. If you are unsure about the order of the notes, you can list them with `collect query notes`.

Additionally, you might be labeling significant results with `collect stars`. If you add an explanatory note with `collect stars`, that note will be listed before any notes you specify with `collect notes`.

Adding notes to a table

Notes are a great place to tell the reader about groups that may not be present in our sample and consequently in our table. For example, we use data from the Second National Health and Nutrition Examination Survey (NHANES II) (McDowell et al. 1981). We wish to fit a model for systolic blood pressure as a function of agegrp (age group) and race.

First, we create an empty cell by replacing agegrp with a missing value for individuals in the third level of race and sixth level of agegrp. Then, we collect the regression results and format our coefficients (_r_b) to two decimal places with collect style cell. Finally, we lay out our table with our variables on the rows and coefficients in a single column:

```
. use https://www.stata-press.com/data/r19/nhanes2l
(Second National Health and Nutrition Examination Survey)
. replace agegrp = . if race==3 & agegrp==6
(11 real changes made, 11 to missing)
. quietly: collect _r_b: regress bpsystol agegrp#race
. collect style cell result[_r_b], nformat(%5.2f)
. collect layout (colname) (result)
Collection: default
      Rows: colname
   Columns: result
   Table 1: 19 x 1
```

	Coefficient
20-29 # White	0.00
20-29 # Black	-0.27
20-29 # Other	-5.15
30-39 # White	2.45
30-39 # Black	5.46
30-39 # Other	-0.26
40-49 # White	8.44
40-49 # Black	18.07
40-49 # Other	10.92
50-59 # White	17.43
50-59 # Black	25.62
50-59 # Other	8.40
60-69 # White	23.26
60-69 # Black	29.49
60-69 # Other	34.57
70+ # White	30.25
70+ # Black	33.69
70+ # Other	0.00
Intercept	117.51

We see that there are no individuals in the 70+ age group who belong to the Other category. By default, empty cells are included in the table. Instead of displaying the empty row, we would like to add a note to point out that our sample did not include any individuals 70 years old, or older, that were in the Other race category. First, we omit the empty cells from our table with collect style showempty. Then, we add a note about the reference category and a note about the empty cell. Because we want to place the Other category between quotes, we use compound double quotes for our second note; see [U] 18.3.5 Double quotes.

```
. collect style showempty off

. collect notes
> "The reference category is white individuals in the 20–29 age group."

. collect notes
> `"There were not any 70+ individuals in the "Other" category in our sample."'
```

Now, we can preview our table with the notes:

```
. collect preview

----------------------------------
                 |   Coefficient
-----------------+----------------
20-29 # White    |          0.00
20-29 # Black    |         -0.27
20-29 # Other    |         -5.15
30-39 # White    |          2.45
30-39 # Black    |          5.46
30-39 # Other    |         -0.26
40-49 # White    |          8.44
40-49 # Black    |         18.07
40-49 # Other    |         10.92
50-59 # White    |         17.43
50-59 # Black    |         25.62
50-59 # Other    |          8.40
60-69 # White    |         23.26
60-69 # Black    |         29.49
60-69 # Other    |         34.57
70+ # White      |         30.25
70+ # Black      |         33.69
Intercept        |        117.51
----------------------------------
The reference category is white individuals in the 20-29 age group.
There were not any 70+ individuals in the "Other" category in our sample.
```

We did not number our notes, so each note is numbered sequentially.

To complete this table, we will add stars to represent p-values that are less than 0.01, less than 0.05, and less than 0.1. We added extra spaces so that the stars are left-aligned. Additionally, we attach the stars to the coefficients and use the `shownote` option to add a note explaining what the stars represent:

```
. collect stars _r_p 0.01 "***" 0.05 "** " 0.1 "*  " 1 "   ",
> attach(_r_b) shownote

. collect preview
```

	Coefficient
20-29 # White	0.00
20-29 # Black	-0.27
20-29 # Other	-5.15*
30-39 # White	2.45***
30-39 # Black	5.46***
30-39 # Other	-0.26
40-49 # White	8.44***
40-49 # Black	18.07***
40-49 # Other	10.92***
50-59 # White	17.43***
50-59 # Black	25.62***
50-59 # Other	8.40*
60-69 # White	23.26***
60-69 # Black	29.49***
60-69 # Other	34.57***
70+ # White	30.25***
70+ # Black	33.69***
Intercept	117.51***

```
*** p<.01, ** p<.05, * p<.1, p<1
The reference category is white individuals in the 20-29 age group.
There were not any 70+ individuals in the "Other" category in our sample.
```

The note regarding the stars will always be placed before any notes we add with `collect notes`.

It seems we forgot to note the source of the data. We would like to add another note, but this time we will add it to the beginning of the list of notes by making it the zeroth note. Then, we will preview the table once more:

```
. collect notes 0: "Source: NHANES II"
. collect preview
```

	Coefficient
20-29 # White	0.00
20-29 # Black	-0.27
20-29 # Other	-5.15*
30-39 # White	2.45***
30-39 # Black	5.46***
30-39 # Other	-0.26
40-49 # White	8.44***
40-49 # Black	18.07***
40-49 # Other	10.92***
50-59 # White	17.43***
50-59 # Black	25.62***
50-59 # Other	8.40*
60-69 # White	23.26***
60-69 # Black	29.49***
60-69 # Other	34.57***
70+ # White	30.25***
70+ # Black	33.69***
Intercept	117.51***

```
*** p<.01, ** p<.05, * p<.1, p<1
Source: NHANES II
The reference category is white individuals in the 20-29 age group.
There were not any 70+ individuals in the "Other" category in our sample.
```

Removing notes

In the section above, we added three notes to our table. However, we now decide that the note regarding the reference category is not needed. To remove just this note, we can preview our table and count from the first note to find that we want to remove the second note. But suppose we are preparing our notes before laying out our table. In this case, we can simply list the notes in a collection, and the appearance styles for these notes, with `collect query notes`:

```
. collect query notes

Table note styles
      Collection: default
          Note 1: Source: NHANES II
          Note 2: The reference category is white individuals in the 20-29 age
> group.
          Note 3: There were not any 70+ individuals in the "Other" category in
> our sample.
     Font family:
       Font size:
      Font color:
     Font weight: normal
      Font style: normal
    Font variant: normal
  Font strikeout: normal
  Font underline: none
        SMCL tag:
     LaTeX macro:
Background color:
Foreground color:
 Shading pattern:
```

Now we see that we want to remove the second note. Below, we remove it and then list our note information once more:

```
. collect notes 2:, clear

. collect query notes

Table note styles
      Collection: default
          Note 1: Source: NHANES II
          Note 2: There were not any 70+ individuals in the "Other" category in
> our sample.
     Font family:
       Font size:
      Font color:
     Font weight: normal
      Font style: normal
    Font variant: normal
  Font strikeout: normal
  Font underline: none
        SMCL tag:
     LaTeX macro:
Background color:
Foreground color:
 Shading pattern:
```

Here we see that our remaining notes are automatically renumbered.

Targeting notes to a specific table

Suppose your layout involves arranging more than one table, but some notes should show up only with a specific table. Use the fortags() option to target a note to a specific table.

In the following, we add a group dimension so that we can arrange our estimation results into two tables. The first table will report the fitted model's F test, and the second table will report the fitted coefficients as in the previous examples.

First, we use collect get to define the p-value result named p_F corresponding with the F statistic result named F, collected from scalar e(F). Note that collect get tags custom expressions like the one for p_F with program_class[nclass] but that the scalar e(F) is tagged with program_class[eclass]. We want to use collect stars to add significance labels to the F result as we did with the coefficients. However, collect stars requires that all tag elements match when attaching significance labels to other results in the collection, so we must use collect recode to make the p-value and F results have the same program_class tag element. This allows us to apply our collect stars rules to results _r_p and p_F and then attach the stars labels to the matching items with results _r_b and F. We plan to show the F statistic with its degrees of freedom, so we apply some numeric formats to their results.

```
. collect get p_F = Ftail(e(df_m),e(df_r),e(F)), tags(cmdset[1])
. collect recode program_class nclass = eclass, fortags(program_class[nclass])
(1 items recoded in collection default)
. collect stars _r_p p_F 0.01 "***" 0.05 "** " 0.1 "*  " 1 "   ",
> attach(_r_b F) shownote
. collect style cell result[F], nformat(%9.2f)
. collect style cell result[df_m df_r], nformat(%12.0fc)
```

Next, we add new tag elements to help target the results for our two tables. We want the model's F test to go in the first table, so we tag those results with group[1]. We want the coefficients to go in the second table, so we tag those results with group[2]. Note that we include the p-value results in these calls to collect addtags so that collect stars can attach the significance labels properly.

```
. collect addtags group[1], fortags(result[df_m df_r F p_F])
(4 items changed in collection default)
. collect addtags group[2], fortags(result[_r_b _r_p])
(36 items changed in collection default)
. collect addtags colname[_hide], fortags(group[1])
(4 items changed in collection default)
```

Now we can use `collect layout` to arrange our tables using the new `group` dimension. We also use `collect style header` to hide the title and levels of dimension `group`. At this point, our notes are all common. None of our notes are targeted to a specific table, so they show up under each table.

```
. collect style header group, title(hide) level(hide)
. collect layout (colname) (result[_r_b df_m df_r F]) (group)
Collection: default
      Rows: colname
   Columns: result[_r_b df_m df_r F]
    Tables: group
   Table 1: 1 x 3
   Table 2: 18 x 1
```

Model DF	Residual DF	F statistic
16	10,323	196.61***

```
*** p<.01, ** p<.05, * p<.1, p<1
Source: NHANES II
There were not any 70+ individuals in the "Other" category in our sample.
```

	Coefficient
20-29 # White	0.00
20-29 # Black	-0.27
20-29 # Other	-5.15*
30-39 # White	2.45***
30-39 # Black	5.46***
30-39 # Other	-0.26
40-49 # White	8.44***
40-49 # Black	18.07***
40-49 # Other	10.92***
50-59 # White	17.43***
50-59 # Black	25.62***
50-59 # Other	8.40*
60-69 # White	23.26***
60-69 # Black	29.49***
60-69 # Other	34.57***
70+ # White	30.25***
70+ # Black	33.69***
Intercept	117.51***

```
*** p<.01, ** p<.05, * p<.1, p<1
Source: NHANES II
There were not any 70+ individuals in the "Other" category in our sample.
```

Finally, we can add notes targeted to specific tables. Recall that the second note in our collection—the note about individuals 70 years old or older that were in the `Other` race category—is not targeted, so we remove it and then add it back as a targeted note for the second table showing the coefficients. We also add a note about the null hypothesis for the table reporting the F test.

```
. collect notes 2:, clear
. collect notes
> `"There were not any 70+ individuals in the "Other" category in our sample."'
> , fortags(group[2])
. collect notes "Null hypothesis is all coefficients are zero.",
> fortags(group[1])
. collect preview
```

Model DF	Residual DF	F statistic
16	10,323	196.61***

```
*** p<.01, ** p<.05, * p<.1, p<1
Source: NHANES II
Null hypothesis is all coefficients are zero.
```

	Coefficient
20-29 # White	0.00
20-29 # Black	-0.27
20-29 # Other	-5.15*
30-39 # White	2.45***
30-39 # Black	5.46***
30-39 # Other	-0.26
40-49 # White	8.44***
40-49 # Black	18.07***
40-49 # Other	10.92***
50-59 # White	17.43***
50-59 # Black	25.62***
50-59 # Other	8.40*
60-69 # White	23.26***
60-69 # Black	29.49***
60-69 # Other	34.57***
70+ # White	30.25***
70+ # Black	33.69***
Intercept	117.51***

```
*** p<.01, ** p<.05, * p<.1, p<1
Source: NHANES II
There were not any 70+ individuals in the "Other" category in our sample.
```

Here we call `collect query notes` to get a report on the common notes, targeted notes, and the notes styles.

```
. collect query notes
Table note styles
      Collection: default
    Common notes:
          Note 1: Source: NHANES II
   Notes for tag: group[1]
          Note 1: Null hypothesis is all coefficients are zero.
   Notes for tag: group[2]
          Note 1: There were not any 70+ individuals in the "Other" category in
> our sample.
     Font family:
       Font size:
      Font color:
     Font weight: normal
      Font style: normal
    Font variant: normal
  Font strikeout: normal
  Font underline: none
        SMCL tag:
     LaTeX macro:
Background color:
Foreground color:
 Shading pattern:
```

Stored results

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

Macros
    `s(collection)`   name of collection

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

[TABLES] **collect style notes** — Collection styles for table notes

[TABLES] **collect query** — Query collection style properties

[TABLES] **collect stars** — Add stars for significant results in a collection

collect query — Query collection style properties

Description | Quick start | Syntax | Option
Remarks and examples | Stored results | Also see

Description

`collect query` displays the settings of various collection parameters.

Quick start

Display the current layout specification

    collect query layout

Display the current row header style properties, such as the binder used to separate factor variables from their levels and whether row header elements are stacked or placed in separate columns

    collect query row

Display the current `putdocx` settings, such as the table width and cell spacing

    collect query putdocx

Display the levels of dimension `dim1` that will be automatically displayed when `dim1` is included in a table

    collect query autolevels dim1

Syntax

Query the current layout specification

`collect query layout` [, `name(`*cname*`)`]

Query automatic levels for a dimension

`collect query autolevels` *dim* [, `name(`*cname*`)`]

Query row header style properties

`collect query row` [, `name(`*cname*`)`]

Query column header style properties

`collect query column` [, `name(`*cname*`)`]

Query table header style properties

`collect query table` [, `name(`*cname*`)`]

Query dimension header style properties

collect query header [*dim* [*level*]] [, name(*cname*)]

Query collection styles for HTML files

collect query html [, name(*cname*)]

Query collection styles for Microsoft Word files

collect query putdocx [, name(*cname*)]

Query collection styles for PDF files

collect query putpdf [, name(*cname*)]

Query collection styles for LaTeX files

collect query tex [, name(*cname*)]

Query collection styles for displaying base levels

collect query showbase [, name(*cname*)]

Query collection styles for displaying empty cells

collect query showempty [, name(*cname*)]

Query collection styles for displaying omitted coefficients

collect query showomit [, name(*cname*)]

Query cell appearance styles

collect query cell [*tag*] [, name(*cname*)]

Query collection style for intercept position

collect query _cons [, name(*cname*)]

Query collection styles for identifying significant results

collect query stars [, name(*cname*)]

Query collection styles for table titles

collect query title [, name(*cname*)]

Query collection styles for table notes

collect query notes [, name(*cname*)]

Query collection composite results

`collect query composite` [, `name(`*cname*`)`]

Query collection composite result definition

`collect query composite` *clevel* [, `name(`*cname*`)`]

where *cname* is a collection name, *dim* is a dimension in the specified collection, *level* is a level of the specified dimension, *clevel* is a composite result, and *tag* identifies cells in the table. *tag* is one or more *dim*[*level*] joined by #.

Option

`name(`*cname*`)` specifies the collection for which settings should be displayed. By default, settings are displayed for the current collection.

Remarks and examples

`collect query` provides information on collection style properties. This can be useful when you want information on style properties without having to preview the table in a collection or lay out a table. For example, perhaps you have adopted a colleague's layout and style with `collect style use`. You can type

```
. collect query layout
```

to see the layout specification, or type

```
. collect query autolevels result
```

to see what levels of dimension `result` will be automatically displayed.

Or perhaps you are building a table, and you want to check whether the column header style properties are to your liking. You can type

```
. collect query column
```

and see which characters will be used to delimit interaction terms, whether extra spaces will be inserted between columns, and more.

Stored results

`collect query layout` stores the following in `s()`:

Macros

`s(collection)`	name of collection
`s(rows)`	rows specification
`s(columns)`	columns specification
`s(tables)`	tables specification
`s(k_tables)`	number of tables
`s(table#)`	layout for the #th table

collect query autolevels stores the following in s():

Macros

s(dimname)	name of dimension
s(levels)	levels of dimension to be automatically displayed
s(collection)	name of collection

collect query row after collect style row split stores the following in s():

Macros

s(span)	true or false
s(position)	left or right
s(dups)	repeat, first, or center
s(spacer)	on or off; whether a blank line is added between stacked row dimensions
s(binder)	binder used to separate factor variables from their levels
s(nobinder)	on or off; whether to bind factor variables and their levels
s(bardelimiter)	delimiter for interaction terms containing the | symbol
s(atdelimiter)	delimiter for interaction terms containing the @ symbol
s(delimiter)	delimiter for interaction terms composed in a single cell
s(nodelimiter)	on or off
s(arrangement)	split
s(collection)	name of collection

collect query row after collect style row stack stores the following in s():

Macros

s(abbreviate)	on or off
s(truncate)	tail, middle, or head
s(wrap)	# of lines allowed for long headers, 0 if not specified
s(wrapon)	word or length
s(length)	maximum display length for stacked headers
s(indent)	on or off; whether stacked headers are to be indented
s(spacer)	on or off; whether a blank line is added between stacked row dimensions
s(binder)	binder used to separate factor variables from their levels
s(nobinder)	on or off; whether to bind factor variables and their levels
s(bardelimiter)	delimiter for interaction terms containing the | symbol
s(atdelimiter)	delimiter for interaction terms containing the @ symbol
s(delimiter)	delimiter for interaction terms composed in a single cell
s(nodelimiter)	on or off
s(arrangement)	stack
s(collection)	name of collection

collect query column stores the following in s():

Macros

s(width)	asis or equal
s(extraspace)	# of spaces used to pad columns
s(position)	top or bottom
s(dups)	repeat, first, or center
s(binder)	binder used to separate factor variables from their levels
s(bardelimiter)	delimiter for interaction terms containing the | symbol
s(atdelimiter)	delimiter for interaction terms containing the @ symbol
s(delimiter)	delimiter for interaction terms composed in a single cell
s(nodelimiter)	on or off
s(collection)	name of collection

collect query table stores the following in s():

Macros

s(dimbinder)	delimiter used to separate dimensions from their levels
s(dimdelimiter)	delimiter for table headers
s(binder)	binder used to separate factor variables from their levels
s(bardelimiter)	delimiter for interaction terms containing the | symbol
s(atdelimiter)	delimiter for interaction terms containing the @ symbol
s(delimiter)	delimiter for interaction terms composed in a single cell
s(nodelimiter)	on or off
s(collection)	name of collection

collect query header stores the following in s():

Macros

s(dimname)	name of dimension, if specified
s(dimlevel)	level of dimension, if specified
s(title)	label, name, or hide
s(level)	label, value, or hide
s(collection)	name of collection

collect query html stores the following in s():

Macros

s(useth)	true or false
s(bcollapse)	true or false
s(collection)	name of collection

collect query putdocx stores the following in s():

Macros

s(title)	table title
s(width)	table width
s(indent)	table indentation from left margin
s(cellspacing)	spacing between adjacent cells and edges of the table
s(halign)	left, right, or center
s(layout)	fixed, autofitwindow, or autofitcontents
s(cellmargin_top)	margin between text and top cell border
s(cellmargin_bottom)	margin between text and bottom cell border
s(cellmargin_left)	margin between text and left cell border
s(cellmargin_right)	margin between text and right cell border
s(note#)	note #
s(collection)	name of collection

`collect query putpdf` stores the following in `s()`:

Macros

`s(title)`	table title
`s(width)`	table width
`s(indent)`	table indentation from left margin
`s(halign)`	`left`, `right`, or `center`
`s(spacing_before)`	spacing before table
`s(spacing_after)`	spacing after table
`s(note#)`	note #
`s(collection)`	name of collection

`collect query tex` stores the following in `s()`:

Macros

`s(centering)`	`true` or `false`
`s(begintable)`	`true` or `false`
`s(collection)`	name of collection

`collect query showbase` stores the following in `s()`:

Macros

`s(showbase)`	`off`, `factor`, or `all`
`s(collection)`	name of collection

`collect query showempty` stores the following in `s()`:

Macros

`s(showempty)`	`on` or `off`
`s(collection)`	name of collection

`collect query showomit` stores the following in `s()`:

Macros

`s(showomit)`	`on` or `off`
`s(collection)`	name of collection

`collect query cell` stores the following in `s()`:

Macros

`s(cridelimiter)`	delimiter for credible intervals
`s(cidelimiter)`	delimiter for confidence intervals
`s(sformat)`	string format
`s(nformat)`	numeric format
`s(valign)`	vertical alignment
`s(halign)`	horizontal alignment
`s(shading_pattern)`	fill pattern
`s(shading_foreground)`	foreground color
`s(shading_background)`	background color
`s(latex)`	LaTeX macro
`s(smcl)`	SMCL directive
`s(font_underline)`	underline pattern
`s(font_strikeout)`	`normal` or `strikeout`
`s(font_variant)`	`allcaps`, `smallcaps`, or `normal`
`s(font_style)`	whether text is formatted as italic
`s(font_weight)`	whether text is formatted as bold
`s(font_color)`	font color
`s(font_size)`	font size
`s(font_family)`	font family
`s(dborder_color)`	color for diagonal cell borders
`s(dborder_pattern)`	pattern for diagonal cell borders
`s(dborder_direction)`	`down`, `up`, or `both`
`s(margin_left_width)`	margin between text and left cell border
`s(margin_right_width)`	margin between text and right cell border
`s(margin_top_width)`	margin between text and top cell border
`s(margin_bottom_width)`	margin between text and bottom cell border
`s(border_left_color)`	color for left cell borders
`s(border_left_pattern)`	pattern for left cell borders
`s(border_left_width)`	width for left cell borders
`s(border_right_color)`	color for left cell borders
`s(border_right_pattern)`	pattern for left cell borders
`s(border_right_width)`	width for left cell borders
`s(border_top_color)`	color for left cell borders
`s(border_top_pattern)`	pattern for left cell borders
`s(border_top_width)`	width for left cell borders
`s(border_bottom_color)`	color for left cell borders
`s(border_bottom_pattern)`	pattern for left cell borders
`s(border_bottom_width)`	width for left cell borders
`s(tag)`	tags identifying table cells to which styles were applied
`s(collection)`	name of collection

`collect query _cons` stores the following in `s()`:

Macros

`s(position)`	`first` or `last`
`s(collection)`	name of collection

collect query stars stores the following in s():

Macros

s(type)	result or dimension
s(results)	*resultID*
s(attach)	*attachres*
s(fortags)	*taglist*
s(value1)	star cutoff value 1
s(label1)	star label 1
s(value2)	star cutoff value 2
s(label2)	star label 2
s(value3)	star cutoff value 3
s(label3)	star label 3
s(value4)	star cutoff value 4
s(label4)	star label 4
s(value5)	star cutoff value 5
s(label5)	star label 5
s(shownote)	on or off; whether to show the stars note after the table
s(pvname)	note *p*-value name
s(nformat)	note numeric format
s(delimiter)	note delimiter between labels
s(prefix)	note prefix, if specified
s(suffix)	note suffix, if specified
s(note)	the stars note
s(collection)	name of collection

collect query title stores the following in s():

Macros

s(shading_pattern)	fill pattern
s(shading_foreground)	foreground color
s(shading_background)	background color
s(latex)	LaTeX macro
s(smcl)	SMCL directive
s(font_underline)	underline pattern
s(font_strikeout)	normal or strikeout
s(font_variant)	allcaps, smallcaps, or normal
s(font_style)	whether text is formatted as italic
s(font_weight)	whether text is formatted as bold
s(font_color)	font color
s(font_size)	font size
s(font_family)	font family
s(title)	custom title
s(collection)	name of collection

collect query notes stores the following in s():

Macros

s(shading_pattern)	fill pattern
s(shading_foreground)	foreground color
s(shading_background)	background color
s(latex)	LaTeX macro
s(smcl)	SMCL directive
s(font_underline)	underline pattern
s(font_strikeout)	normal or strikeout
s(font_variant)	allcaps, smallcaps, or normal
s(font_style)	whether text is formatted as italic
s(font_weight)	whether text is formatted as bold
s(font_color)	font color
s(font_size)	font size

s(font_family)	font family
s(note1)	first note
s(note2)	second note
s(note3)	third note
s(note#)	#th note
s(k_notes)	number of notes
s(tag1_note1)	first note for the first tag
s(tag1_note2)	second note for the first tag
s(tag1_note#)	#th note for the first tag
s(k_tag1_notes)	number of notes for the first tag
s(tag2_note1)	first note for the second tag
s(tag2_note2)	second note for the second tag
s(tag2_note#)	#th note for the second tag
s(k_tag2_notes)	number of notes for the second tag
s(tag$\#_t$_note$\#_n$)	$\#_n$th note for the $\#_t$th tag
s(k_tag#_notes)	number of notes for the #th tag
s(k_notes_tags)	number of notes tags
s(notes_tags)	tags for notes
s(tag1_notes_tag)	first tag
s(tag2_notes_tag)	second tag
s(tag$\#_t$_notes_tag)	$\#_t$th tag
s(collection)	name of collection

`collect query composite` stores the following in `s()`:

Macros

s(levels)	composite result levels
s(collection)	name of collection

`collect query composite` *level* stores the following in `s()`:

Macros

s(override)	on or off; whether to override trim properties of elements
s(trim)	on or off; whether to remove extra spaces from numeric format
s(delimiter)	delimiter placed between elements
s(elements)	composite result elements
s(composite)	composite result level
s(collection)	name of collection

Also see

[TABLES] **collect layout** — Specify table layout for the current collection

[TABLES] **collect style autolevels** — Collection styles for automatic dimension levels

[TABLES] **collect style row** — Collection styles for row headers

[TABLES] **collect style column** — Collection styles for column headers

collect recode — Recode dimension levels in a collection

Description

`collect recode` recodes dimension levels attached to values in the current collection.

Quick start

Recode level `lev1` to `newlevel` and level `lev2` to `newlevel2` in dimension `dim1`

```
collect recode dim1 lev1=newlevel lev2=newlevel2
```

Same as above, but apply the recoded levels only to values in the collection tagged with `dim2[lev3]`

```
collect recode dim1 lev1=newlevel lev2=newlevel2, fortags(dim2[lev3])
```

Recode levels `2.catvar` and `3.catvar` in dimension `dim2`

```
collect recode dim2 2.catvar=catvar2 3.catvar=catvar3
```

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Build and style table

Syntax

`collect recode` *dim oldlevel* = *newlevel* [*oldlevel* = *newlevel* ...]

[, `name(`*cname*`)` `fortags(`*taglist*`)`]

where *dim* is the name of a dimension in the collection, *oldlevel* is the name of an existing level in the dimension, and *newlevel* is the name of the level to which *oldlevel* is to be set.

Levels `_r_ci` and `_r_cri` of dimension `result` are not allowed in *oldlevel*.

Options

Main

`name(`*cname*`)` specifies the collection in which to recode the levels of the dimension. If this option is not specified, the change is made in the current collection.

Options

`fortags(`*taglist*`)` specifies conditions for selecting the values to which the recoded levels will be applied. Values with tags in *taglist* will have their levels recoded.

Within the *taglist*, if tags are joined by `#`, values having all of these tags are selected; if tags are separated by a space, values with any of these tags are selected.

taglist contains

tagspec

tagspec taglist

tagspec contains

tag

tag#*tag*[#*tag*[...]]

tag contains

dimension

dimension`[`*levels*`]`

dimension is a dimension in the collection.

levels are levels of the corresponding dimension.

Levels `_r_ci` and `_r_cri` of dimension `result` are not allowed in *taglist*.

Distinguish between `[]`, which are to be typed, and [ ], which indicate optional arguments.

Remarks and examples

After collecting results, we occasionally need to recode levels of a dimension to lay out the table that we wish to create. `collect recode` replaces the existing levels of a dimension with newly specified levels.

To demonstrate, we use data from the Second National Health and Nutrition Examination Survey (NHANES II) (McDowell et al. 1981). With the `table` command, we create a table with two regression results as well as the means for each dependent variable.

```
. use https://www.stata-press.com/data/r19/nhanes2
. quietly table (result colname) (statcmd),
>     command(regress bpsystol age weight)
>     command(regress bpdiast age weight)
>     statistic(mean bpsystol bpdiast) nformat(%6.3f)
. collect style header statcmd, level(value)
. collect preview
```

	1	2	3
Mean			
Systolic blood pressure			130.882
Diastolic blood pressure			81.715
Coefficient			
Age (years)	0.638	0.188	
Weight (kg)	0.407	0.312	
Intercept	71.271	50.376	

The statcmd dimension is used to identify the columns of the table. The regression results are tagged with statcmd[1] and statcmd[2] for bpsystol and bpdiast, respectively. The means of the dependent variables are tagged with statcmd[3]. We can use collect recode to recode the levels of statcmd so that the mean of each dependent variable has the same level as the corresponding regression results.

```
. collect recode statcmd 3 = 1, fortags(var[bpsystol])
(1 items recoded in collection Table)

. collect recode statcmd 3 = 2, fortags(var[bpdiast])
(1 items recoded in collection Table)

. collect preview

-------------------------------------------------
                           |        1          2
---------------------------+---------------------
Mean                       |
  Systolic blood pressure  |  130.882
  Diastolic blood pressure |              81.715
Coefficient                |
  Age (years)              |    0.638      0.188
  Weight (kg)              |    0.407      0.312
  Intercept                |   71.271     50.376
-------------------------------------------------
```

Because we wanted to recode only statcmd[3] to statcmd[1] for the mean value of bpsystol, we specify fortags(var[bpsystol]), which indicates that the recode will be performed only for values with this tag. Likewise, we recode statcmd[3] to statcmd[2] only for values with the tag var[bpdiast]. This produced a table with only two columns, one for each dependent variable.

Our rows are identified by the result and colname dimensions. Because our means have different levels of colname, they appear on separate rows. We can place them on the same row by recoding the separate bpsystol and bpdiast levels to one level, say, mean.

```
. collect recode colname bpsystol = mean
(1 items recoded in collection Table)

. collect recode colname bpdiast = mean
(1 items recoded in collection Table)

. collect preview

----------------------------------
              |        1        2
--------------+-------------------
Mean          |
  mean        |  130.882   81.715
Coefficient   |
  Age (years) |    0.638    0.188
  Weight (kg) |    0.407    0.312
  Intercept   |   71.271   50.376
----------------------------------
```

Now, we have the values arranged where we would like them in our table. We can clean up the row and column headers of our table by typing

```
. collect label levels statcmd 1 "Systolic BP" 2 "Diastolic BP", modify

. collect style header statcmd, level(label)

. collect label levels result mean "Mean of dependent variable"
> _r_b "Coefficients", modify

. collect style header colname[mean], level(hide)
```

```
. collect preview
```

	Systolic BP	Diastolic BP
Mean of dependent variable	130.882	81.715
Coefficients		
Age (years)	0.638	0.188
Weight (kg)	0.407	0.312
Intercept	71.271	50.376

See [TABLES] **collect label** and [TABLES] **collect style header** for more information on these commands.

Stored results

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

Macros

`s(collection)`	name of collection
`s(dimname)`	name of dimension
`s(k_recoded)`	number of recoded items

References

Huber, C. 2021. Customizable tables in Stata 17, part 3: The classic table 1. *The Stata Blog: Not Elsewhere Classified.* https://blog.stata.com/2021/06/24/customizable-tables-in-stata-17-part-3-the-classic-table-1/.

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

[TABLES] **collect addtags** — Add tags to items in a collection

[TABLES] **collect remap** — Remap tags in a collection

collect remap — Remap tags in a collection

Description Quick start Menu Syntax
Options Remarks and examples Stored results Reference
Also see

Description

`collect remap` remaps tags associated with values in a collection. Remapping tags can be useful when you need to specify a table layout but the original tags do not allow you to place values from different commands that are tagged differently into the same rows, columns, or tables.

With `collect remap`, you can remap tags for levels of an existing dimension to a new dimension with the same levels, remap tags for levels of an existing dimension to a new dimension with new levels, or remap tags for levels of an existing dimension to new levels within the existing dimension.

Quick start

Remap all tags with dimension `olddim` to new dimension `newdim`, with the level unchanged

```
collect remap olddim = newdim
```

Remap tags with levels `lev1` and `lev2` in dimension `olddim` to `newdim`, with the level unchanged

```
collect remap olddim[lev1 lev2] = newdim
```

Same as above, but remap tags to the specified levels of the new dimension

```
collect remap olddim[lev1 lev2] = newdim[lev4 lev3]
```

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Build and style table

Syntax

Remap tags from an existing dimension to a new dimension, with the level unchanged

`collect remap` *olddim* = *newdim* [, `name(`*cname*`)` `fortags(`*taglist*`)`]

Remap tags with specified levels of an existing dimension to a new dimension, with the level unchanged

`collect remap` *olddim*`[`*oldlevels*`]` = *newdim* [, `name(`*cname*`)` `fortags(`*taglist*`)`]

Remap tags with specified levels of an existing dimension to new levels of a new dimension

`collect remap` *olddim*`[`*oldlevels*`]` = *newdim*`[`*newlevels*`]`
[, `name(`*cname*`)` `fortags(`*taglist*`)`]

where *olddim* is the name of an existing dimension in the collection, *newdim* is the name of a dimension into which levels of *oldim* are to be mapped, *oldlevels* are the names of existing levels in the dimension, and *newlevels* are the names of the levels to which *oldlevels* are to be set.

Levels `_r_ci` and `_r_cri` of dimension `result` are not allowed in *oldlevels*.

Distinguish between `[]`, which are to be typed, and [ ], which indicate optional arguments.

Options

Main

`name(`*cname*`)` specifies the collection in which to remap items. If this option is not specified, the change is made in the current collection.

Options

`fortags(`*taglist*`)` specifies conditions for selecting the values to which remapped tags will be applied. Values with tags in *taglist* will have their tags remapped.

Within the *taglist*, if tags are joined by `#`, values having all of these tags are selected; if tags are separated by a space, values with any of these tags are selected.

taglist contains

tagspec

tagspec taglist

tagspec contains

tag

tag`#`*tag*[`#`*tag*[...]]

tag contains

dimension

dimension`[`*levels*`]`

dimension is a dimension in the collection.

levels are levels of the corresponding dimension.

Levels _r_ci and _r_cri of dimension result are not allowed in *taglist*.

Distinguish between [ ], which are to be typed, and [ ], which indicate optional arguments.

Remarks and examples

After collecting results, we occasionally need to remap tags to lay out the table that we wish to create. collect remap allows you to remap tags from the existing levels of an existing dimension to new tags, possibly with new dimensions and new levels.

To demonstrate, we use data from the Second National Health and Nutrition Examination Survey (NHANES II) (McDowell et al. 1981). With the table command, we create a table with two regression results as well as the means for each dependent variable.

```
. use https://www.stata-press.com/data/r19/nhanes2
. quietly table (result colname) (statcmd),
>     command(regress bpsystol age weight)
>     command(regress bpdiast age weight)
>     statistic(mean bpsystol bpdiast) nformat(%6.3f)
. collect style header statcmd, level(value)
. collect preview
```

	1	2	3
Mean			
Systolic blood pressure			130.882
Diastolic blood pressure			81.715
Coefficient			
Age (years)	0.638	0.188	
Weight (kg)	0.407	0.312	
Intercept	71.271	50.376	

The statcmd dimension is used to identify the columns of the table. The regression results are tagged with statcmd[1] and statcmd[2] for bpsystol and bpdiast, respectively. The means of the dependent variables are tagged with statcmd[3]. We can use collect remap to remap the statcmd[3] tags so that the mean of each dependent variable has the same level as the corresponding regression results.

```
. collect remap statcmd[3]=statcmd[1], fortags(var[bpsystol])
(1 items remapped in collection Table)
. collect remap statcmd[3]=statcmd[2], fortags(var[bpdiast])
(1 items remapped in collection Table)
. collect preview
```

	1	2
Mean		
Systolic blood pressure	130.882	
Diastolic blood pressure		81.715
Coefficient		
Age (years)	0.638	0.188
Weight (kg)	0.407	0.312
Intercept	71.271	50.376

Because we wanted to remap only `statcmd[3]` to `statcmd[1]` for the mean value of `bpsystol`, we specify `fortags(var[bpsystol])`, which indicates that the remapping will be performed only for values with this tag. Likewise, we remap `statcmd[3]` to `statcmd[2]` only for values with the tag `var[bpdiast]`. This produced a table with only two columns, one for each dependent variable.

Our rows are identified by the `result` and `colname` dimensions. Because our means have different levels of `colname`, they appear on separate rows. We can place them on the same row by remapping the separate `bpsystol` and `bpdiast` levels to one level, say, `mean`.

```
. collect remap colname[bpsystol bpdiast] = colname[mean mean]
(2 items remapped in collection Table)
. collect preview
```

	1	2
Mean		
mean	130.882	81.715
Coefficient		
Age (years)	0.638	0.188
Weight (kg)	0.407	0.312
Intercept	71.271	50.376

Now, we have the values arranged where we would like them in our table. We can clean up the row and column headers of our table by typing

```
. collect label levels statcmd 1 "Systolic BP" 2 "Diastolic BP", modify
. collect style header statcmd, level(label)
. collect label levels result mean "Mean of dependent variable"
>     _r_b "Coefficients", modify
. collect style header colname[mean], level(hide)
. collect preview
```

	Systolic BP	Diastolic BP
Mean of dependent variable	130.882	81.715
Coefficients		
Age (years)	0.638	0.188
Weight (kg)	0.407	0.312
Intercept	71.271	50.376

See [TABLES] **collect label** and [TABLES] **collect style header** for more information on these commands.

In the examples above, we remapped tags to new levels within the same dimension. We could have performed these same remappings using `collect recode`. However, `collect remap` can do more. We could, for instance, type

```
. collect remap colname[bpsystol bpdiast] = mycol
```

to remap the existing tags to tags with new dimension `mycol` but with the existing level names. We could also type

```
. collect remap colname[bpsystol bpdiast] = mycol[mean mean]
```

and remap the existing tags to tags with new dimension `mycol` and level `mean`.

Stored results

collect remap stores the following in s():

Macros

s(collection)	name of collection
s(k_remapped)	number of remapped items

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

[TABLES] **collect addtags** — Add tags to items in a collection

[TABLES] **collect recode** — Recode dimension levels in a collection

collect rename — Rename a collection

Description

`collect rename` changes the name associated with a collection.

Quick start

Change name of existing collection `c1` to `collection1`

```
collect rename c1 collection1
```

Same as above, but overwrite `collection1` if it exists

```
collect rename c1 collection1, replace
```

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Rename collection

Syntax

`collect rename` *cname newcname* [`, replace`]

where *cname* is the name of an existing collection and *newcname* is the new name for the existing collection.

Option

`replace` permits `collect rename` to overwrite an existing collection.

Remarks and examples

`collect rename` changes the name associated with a collection. This is useful when you have multiple collections in memory. For example, you may be collecting results into the collection named `default`. You may want to give this collection a more descriptive name, based on the results you have collected. For example, you might call this collection `means`:

```
. collect rename default means
```

Now it is clear that the collection `means` contains means.

Stored results

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

Macros
`s(current)` name of current collection

Also see

[TABLES] **collect create** — Create a new collection

[TABLES] **collect copy** — Copy a collection

collect save — Save a collection to disk

Description

`collect save` saves a collection to a file.

Quick start

Save current collection to a file, with the filename constructed using the collection name with the `.stjson` extension

```
collect save
```

Same as above, but save collection to `myfile.stjson`

```
collect save myfile
```

Save collection `c1` to `c1.stjson`, replacing it if it exists

```
collect save, name(c1) replace
```

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Save collection

Syntax

`collect save` [*filename*] [`,` `replace` `name(`*cname*`)`]

If *filename* is specified without an extension, `.stjson` is assumed. If *filename* contains embedded spaces, enclose it in double quotes.

Options

`replace` permits `collect save` to overwrite an existing file.

`name(`*cname*`)` specifies the name of the collection to be saved. By default, the current collection is saved.

Remarks and examples

With `collect save`, you can save your collection to a file. This might be useful if, for example, you have collected the results you want and you are about to exit Stata but you plan to continue working with this collection in a future session. Suppose your current collection is named `default` and you simply type

```
. collect save
```

Your collection will be stored in a file called `default.stjson`. You might instead specify a more descriptive filename, as follows:

```
. collect save means
```

The default behavior is to save the current collection. If you have multiple collections in memory and you wish to save a collection other than the current collection, specify the collection name with `name()`.

Whenever you are ready to resume working with the collection `means`, you can type

```
. collect use means
```

to load the collection from that file into memory; see [TABLES] **collect use**.

Stored results

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

Macros

`s(collection)`	name of saved collection
`s(filename)`	name of the file

Also see

[TABLES] **collect use** — Use a collection from disk

collect set — Set the current (active) collection

Description

`collect set` identifies a collection to be the current (active) collection.

Quick start

Set `c1` as the current collection

```
collect set c1
```

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Set current collection

Syntax

`collect set` *cname*

where *cname* is the name of an existing collection.

Remarks and examples

`collect set` identifies a collection to be the current collection. This means that any results collected with the `collect` prefix or the `collect get` command will be stored in this collection, by default. Also, any style specifications and labels will apply to this collection. And any exported tables will be based on this collection as well.

When you start Stata, you will have a single collection called `default`. This will be the current collection until you specify another collection to be the current one or until you issue a `table`, `dtable`, or `etable` command. These commands all create tables and automatically create a collection with their results.

If you want to set another collection to be the current collection but cannot remember the collection name, you can list all the collections in memory by typing

```
. collect dir
```

You can then specify your selected collection name in `collect set`.

If you would instead like to create a new collection and make it the current one, you can use `collect create`.

Stored results

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

Macros

`s(current)`	name of current collection

Also see

[TABLES] **collect create** — Create a new collection

collect stars — Add stars for significant results in a collection

Description Quick start Menu Syntax
Options Remarks and examples Stored results Also see

Description

collect stars manages the creation of stars for indicating the significance of results in a collection. You can also add a note to the table to explain the significance represented by the stars.

Quick start

Specify three levels of stars based on p-values in _r_p in the result dimension

```
collect stars _r_p 0.01 "***" 0.05 "**" 0.1 "*"
```

Same as above, and attach the stars to coefficients stored in _r_b

```
collect stars _r_p 0.01 "***" 0.05 "**" 0.1 "*", attach(_r_b)
```

Same as above, and display a note explaining what the stars represent

```
collect stars _r_p 0.01 "***" 0.05 "**" 0.1 "*", attach(_r_b) shownote
```

Clear the current stars specification

```
collect stars, clear
```

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Build and style table

Syntax

`collect stars` *resultIDs* [*#1* "*label1*"
[*#2* "*label2*"
[*#3* "*label3*"
[*#4* "*label4*"
[*#5* "*label5*"]]]]] [, *options*]

resultIDs are levels in the `result` dimension whose values determine the stars to be applied.

Value-label pairs are rearranged such that $\#1 < \#2 < \#3 < \#4 < \#5$.

For value *v* corresponding to one of the results in *resultIDs*,

if $v \geq \#5$, no new stars value is created

if $v < \#5$, the new stars value is set to "*label5*"

if $v < \#4$, the new stars value is set to "*label4*"

if $v < \#3$, the new stars value is set to "*label3*"

if $v < \#2$, the new stars value is set to "*label2*"

if $v < \#1$, the new stars value is set to "*label1*"

options	Description
Main	
`name(`*cname*`)`	specify stars for collection *cname*
`clear`	remove previous style properties
`attach(`*attachres*`)`	specify that stars be appended to items in *attachres*
`fortags(`*taglist*`)`	specify tags identifying items for which to generate new star items
`result`	tag new items with `result[stars]`
`dimension`	tag new items using `stars[label]`
Options	
[`no`]`shownote`	specify whether to display the stars note
`increasing`	compose stars note with increasing *p*-values; the default
`decreasing`	compose stars note with decreasing *p*-values
`pvname(`*string*`)`	specify a name for *p*-value in the stars note
`delimiter(`*string*`)`	specify label delimiter in the stars note
`nformat(`%*fmt*`)`	specify numeric format for cutoff values in the stars note
`prefix(`*string*`)`	specify a prefix for the stars note
`suffix(`*string*`)`	specify a suffix for the stars note

Options

Main

`name(`*cname*`)` specifies the collection to which stars are to be applied. By default, stars are applied to the current collection.

`clear` removes existing `collect stars` properties.

`attach(`*attachres*`)` specifies that items with result levels in *attachres* be appended with the associated star label when rendered to a table.

`fortags(`*taglist*`)` specifies conditions for selecting which values are to be used to create stars. Values with tags in *taglist* will be used to generate new stars.

Within the *taglist*, if tags are joined by `#`, values having all of these tags are selected; if tags are separated by a space, values with any of these tags are selected.

taglist contains

tagspec

tagspec taglist

tagspec contains

tag

tag`#`*tag* [`#`*tag* [...]]

tag contains

dimension

dimension`[`*levels*`]`

dimension is a dimension in the collection.

levels are levels of the corresponding dimension.

Distinguish between `[]`, which are to be typed, and [ ], which indicate optional arguments.

`result` and `dimension` control how `collect stars` adds items when labeling significant results. These options are mutually exclusive.

`result` specifies the factory default behavior, and this option is necessary only if the following `dimension` behavior is in effect and you want to change back to the `result` behavior.

`dimension` specifies that dimension `stars` be added to the collection. Items will be tagged with `stars[value]`, and the labels will be tagged with `stars[label]`. Use this option for layouts where results are to be stacked within columns, and use new dimension `stars` in the column specification of the layout.

Options

`noshownote` and `shownote` control whether to show the stars note.

`noshownote` suppresses the stars note. This is the default.

`shownote` specifies to show the stars note.

`increasing` and `decreasing` control the order of p-values in the stars note.

`increasing` specifies that the stars note be composed with increasing p-values. This is the default.

`decreasing` specifies that the stars note be composed with decreasing p-values.

`pvname(`*string*`)` specifies a name for p-value in the stars note. The default is `pvname(p)`.

`delimiter(`*string*`)` specifies the delimiter between labels in the stars note. The default is `delimiter(",")`.

`nformat(`*%fmt*`)` specifies the numeric format for the cutoff values in the stars note. The default is `nformat(%9.0g)`.

`prefix(`*string*`)` specifies the prefix for the stars note. The prefix is empty by default.

`suffix(`*string*`)` specifies the suffix for the stars note. The suffix is empty by default.

Remarks and examples

Stars are often used in tables to denote significance. `collect stars` allows you to include stars in your table based on other values, typically p-values that are already in your collection. You can attach the stars to a statistic, typically coefficients, or present them separately. Whichever style you choose, you will likely want to use the `shownote` option to add a note to your table explaining the significance level that the stars represent.

`collect stars` stores its specification among the collection's style properties. This means that the creation of stars for significant results can be made before collecting results, can be saved to disk via `collect style save`, and can be applied to other collections via `collect style use`. Additionally, you can check the current stars specification with `collect query stars`.

To demonstrate, we first create a table of regression results displaying coefficients and p-values.

```
. use https://www.stata-press.com/data/r19/nhanes2
. quietly: collect: regress bpsystol bmi i.region age
. collect style showbase off
. collect layout (colname) (result[_r_b _r_p])
Collection: default
      Rows: colname
   Columns: result[_r_b _r_p]
   Table 1: 6 x 2
```

	Coefficient	p-value
Body mass index (BMI)	1.303719	0.000
MW	-.0659707	0.907
S	-.5170085	0.356
W	-.6045511	0.289
Age (years)	.5887217	0.000
Intercept	69.89029	0.000

Rather than showing the p-values, we can use `collect stars` to define the levels of the p-values stored in `_r_p` for which stars should be shown. Here we will use three stars for values less than 0.01, two stars for values less than 0.05, and one star for values less than 0.1. A new `stars` level in the `result` dimension is created and can be used in our table layout.

```
. collect stars _r_p 0.01 "***"  0.05 "**" 0.1 "*"
. collect layout (colname) (result[_r_b stars])
Collection: default
      Rows: colname
   Columns: result[_r_b stars]
   Table 1: 6 x 2
```

	Coefficient	stars
Body mass index (BMI)	1.303719	***
MW	-.0659707	
S	-.5170085	
W	-.6045511	
Age (years)	.5887217	***
Intercept	69.89029	***

It is unlikely that we want the level name `stars` to show in the column header. It would also be helpful to left-align the stars to be closer to the reported coefficients. We can do this with `collect style header` and `collect style cell`.

```
. collect style header result[stars], level(hide)
. collect style cell result[stars], halign(left)
. collect preview
```

	Coefficient	
Body mass index (BMI)	1.303719	***
MW	-.0659707	
S	-.5170085	
W	-.6045511	
Age (years)	.5887217	***
Intercept	69.89029	***

Alternatively, we can directly attach the stars to the coefficient by specifying the `attach()` option and naming the result (`_r_b`) that we want the stars attached to.

```
. collect stars _r_p 0.01 "***"  0.05 "** " 0.1 "*  " 1 "   ", attach(_r_b)
. collect layout (colname) (result[_r_b])
Collection: default
      Rows: colname
   Columns: result[_r_b]
   Table 1: 6 x 1
```

	Coefficient
Body mass index (BMI)	1.303719***
MW	-.0659707
S	-.5170085
W	-.6045511
Age (years)	.5887217***
Intercept	69.89029***

Here we added extra spaces to force three characters following the number in each cell. This gives nice alignment when we are looking at the results in formats such as plain text and the Stata Markup and Control Language format. However, if you are exporting your table to other formats, you may prefer to leave the stars in a separate column and apply alignment and margin styles to achieve your desired look.

It is common to want to stack coefficients and their standard errors in a single column of an estimation table. While the default `collect stars` behavior tends to yield ugly tables for this layout, the solution is to use option `dimension` and put the new `stars` dimension in the column specification of the layout.

```
. collect stars _r_p 0.01 "***"  0.05 "**" 0.1 "*", attach(_r_b) dimension

. collect layout (colname#result[_r_b _r_se]) (stars)

Collection: default
      Rows: colname#result[_r_b _r_se]
   Columns: stars
   Table 1: 18 x 2

-----------------------+--------------
Body mass index (BMI)  |
  Coefficient          |  1.303719 ***
  Std. error           |  .0395032
MW                     |
  Coefficient          | -.0659707
  Std. error           |  .5633352
S                      |
  Coefficient          | -.5170085
  Std. error           |   .559805
W                      |
  Coefficient          | -.6045511
  Std. error           |  .5698782
Age (years)            |
  Coefficient          |  .5887217 ***
  Std. error           |  .0112852
Intercept              |
  Coefficient          |  69.89029 ***
  Std. error           |  1.142773
-----------------------+--------------
```

Although the stars we have chosen to represent the p-values may be common in practice, to be clear, we add an explanatory note below. Note that we do not have to respecify our labels; we are simply adding on to our current stars specification:

```
. collect stars, shownote
. collect preview
```

Body mass index (BMI)	
Coefficient	1.303719 ***
Std. error	.0395032
MW	
Coefficient	-.0659707
Std. error	.5633352
S	
Coefficient	-.5170085
Std. error	.559805
W	
Coefficient	-.6045511
Std. error	.5698782
Age (years)	
Coefficient	.5887217 ***
Std. error	.0112852
Intercept	
Coefficient	69.89029 ***
Std. error	1.142773

```
*** p<.01, ** p<.05, * p<.1
```

Stored results

collect stars stores the following in s():

Macros

s(collection)	name of collection
s(value1)	star cutoff value 1
s(label1)	star label 1
s(value2)	star cutoff value 2
s(label2)	star label 2
s(value3)	star cutoff value 3
s(label3)	star label 3
s(value4)	star cutoff value 4
s(label4)	star label 4
s(value5)	star cutoff value 5
s(label5)	star label 5

Also see

[TABLES] **collect clear** — Clear all collections in memory

[TABLES] **collect drop** — Drop collections from memory

[TABLES] **collect query** — Query collection style properties

collect title — Add a custom table title in a collection

Description

`collect title` manages the creation of a custom table title in a collection.

Quick start

Specify a custom table title

```
collect title "Model comparison"
```

Clear the current custom table title

```
collect title, clear
```

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Build and style table

Syntax

`collect title "`*string*`"` [, *options*]

options	Description
`name(`*cname*`)`	specify custom table title for collection *cname*
`clear`	remove custom table title

Options

`name(`*cname*`)` specifies the collection to which the custom table title is added or removed. By default, the custom table title change is applied to the current collection.

`clear` removes the existing custom table title.

Remarks and examples

When creating a table, you will likely want to add a custom title. With `collect title`, you can specify your custom title, and it will be added to the table in plain text. If your table layout generates multiple tables, the title added with `collect title` will be displayed for each table. You can also customize the appearance style of the title with `collect style title`. For example, you can specify the font, text color, and more.

▷ Example 1: Adding a custom table title

Suppose that we want to create a table comparing regression results. First, we load data from the Second National Health and Nutrition Examination Survey (NHANES II) (McDowell et al. 1981), and we fit two models for systolic blood pressure:

```
. use https://www.stata-press.com/data/r19/nhanes2l
(Second National Health and Nutrition Examination Survey)
. quietly: collect _r_b: regress bpsystol i.agegrp
. quietly: collect _r_b: regress bpsystol i.agegrp i.sex
```

Then, we add our custom title and lay out our table:

```
. collect title "Models for systolic blood pressure"
. collect layout (colname) (cmdset#result)
Collection: default
      Rows: colname
   Columns: cmdset#result
   Table 1: 9 x 2
Models for systolic blood pressure
```

	1 Coefficient	2 Coefficient
20-29	0	0
30-39	2.89081	2.916153
40-49	9.597631	9.603552
50-59	18.32889	18.38803
60-69	24.17618	24.18566
70+	30.82992	30.93702
Male		0
Female		-4.015163
Intercept	117.3466	119.4303

◁

Stored results

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

Macros
`s(collection)` name of collection

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

[TABLES] **collect style title** — Collection styles for table titles

[TABLES] **collect query** — Query collection style properties

Description Quick start Menu Syntax
Options Remarks and examples Stored results Reference
Also see

Description

`collect use` loads a collection into memory from a saved file, making it the current collection. The collection name in the file is used for the collection in memory unless a new collection name is assigned.

Quick start

Load the collection from `myfile.stjson`, using the collection name in the file

    collect use myfile

Same as above, but name this collection `c2`

    collect use c2 myfile

Same as above, but replacing collection `c2` if it exists

    collect use c2 myfile, replace

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Use collection

Syntax

Load a new collection from a file, using the collection name in the file

`collect use` *filename* [, *options*]

Load a new collection from a file, using a specified collection name

`collect use` *newcname* *filename* [, *options*]

where *newcname* is the new name to be assigned to the collection.

If *filename* is specified without an extension, `.stjson` is assumed. If *filename* contains embedded spaces, enclose it in double quotes.

options	Description
`replace`	overwrite existing collection
[`no`]`warn`	display or suppress notes about tags that are not recognized; default is to display

Options

replace permits collect use to overwrite an existing collection in memory. Whether you load a collection using the collection name in the file or specify a new collection name, replace is required if that collection already exists and is not empty.

warn and nowarn control the display of notes when collect encounters a tag it does not recognize.

warn, the default, specifies that collect show the notes.

nowarn specifies that collect not show the notes.

These options override the default behavior controlled by set collect_warn; see [TABLES] **set collect_warn**.

Remarks and examples

At times, you may not finalize your table of results in a single Stata session. In addition, you may wish to return to a collection to build another table from the same collected results or to export a table you have created to a publication-ready format. Saving a collection allows you to easily resume working with it later.

To demonstrate, we use data from the Second National Health and Nutrition Examination Survey (NHANES II) (McDowell et al. 1981). Below, we fit a simple linear regression model and collect the coefficients (_r_b). We also use the quietly prefix to suppress the output.

```
. use https://www.stata-press.com/data/r19/nhanes2
. quietly: collect _r_b: regress bpsystol age
```

Then, we arrange the items in our collection with collect layout. We place the variable names on the rows and the statistics (result) on the columns:

```
. collect layout (colname) (result)
Collection: default
      Rows: colname
   Columns: result
   Table 1: 2 x 1
```

	Coefficient
Age (years)	.6520775
Intercept	99.85603

Then, we format the results to display only two digits after the decimal and shorten the label for the coefficients. We save our collection under the filename bp:

```
. collect style cell, nformat(%5.2f)
. collect label levels result _r_b "Coef.", modify
. collect save bp
(collection default saved to file bp.stjson)
```

Now, let's clear all collection information from memory:

```
. collect clear
```

To resume working on our collection, we load it into memory and then get a preview of the contents:

```
. collect use bp
(collection default read from file bp.stjson)
. collect preview
```

	Coef.
Age (years)	0.65
Intercept	99.86

Notice that the layout, appearance style, and labels are all saved with the collection. Also, while the filename is `bp.stjson`, the collection we saved was named `default`. `collect use` issues a note informing us of the collection and file name.

If you want to load the collection with the name `bp`, you can instead type

```
. collect use bp bp
```

You can also rename the collection at a later time with `collect rename`.

Stored results

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

Macros

`s(current)`	name of current collection
`s(filename)`	name of the file used

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

[TABLES] **collect save** — Save a collection to disk

collect layout — Specify table layout for the current collection

Description

`collect layout` builds a table from the current collection. With `collect layout`, you specify which of the values that were collected from other Stata commands are to appear in the table, and you specify how the table is to be arranged.

The values in a collection are categorized into dimensions. These dimensions may represent types of statistics and covariate names. To specify a table layout, you specify which of these dimensions go on the rows and which of these dimensions go on the columns of your table.

As you specify the table layout, you can also determine which levels of a dimension are to be included in the table. For instance, if a collection includes three types of statistics—means, standard deviations, and frequencies—you may specify that only means are to appear in the table.

Table layouts can go beyond a single table with rows and columns. You can also specify dimensions that identify multiple tables.

`collect layout`, typed without arguments, reports the current layout.

Quick start

Table with dimension `dim1` on the rows and `dim2` on the columns

```
collect layout (dim1) (dim2)
```

Same as above, but include levels `lev1` and `lev2` of `dim1` instead of the levels automatically determined by `collect`

```
collect layout (dim1[lev1 lev2]) (dim2)
```

Table with interacted levels of dimension `dim1` and `dim2` on the rows and `dim3` on the columns

```
collect layout (dim1#dim2) (dim3)
```

Table with levels of dimension `dim1` and then the levels of `dim2` on the rows and `dim3` on the columns

```
collect layout (dim1 dim2) (dim3)
```

Separate tables for each level of dimension `dim3`, each with `dim1` on the rows and `dim2` on the columns

```
collect layout (dim1) (dim2) (dim3)
```

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Build and style table

Syntax

Basic syntax for specifying the table layout

Single column layout with specified rows

`collect layout (`*rows*`)`

Single row layout with specified columns

`collect layout () (`*cols*`)`

Single table layout with specified rows and columns

`collect layout (`*rows*`) (`*cols*`)`

Multiple tables layout with specified rows and columns

`collect layout (`*rows*`) (`*cols*`) (`*tabs*`)`

Full syntax for specifying the table layout

`collect layout (`[*rows*]`)` [`(`[*cols*]`)` [`(`*tabs*`)`]] [`,` `warn` | `nowarn`]

Report the current layout

`collect layout`

Clear the layout information

`collect layout, clear`

rows, *cols*, and *tabs* are each composed of a *taglist* that selects dimensions and possibly levels within a dimension. Within the *taglist*, if tags are joined by `#`, their levels are interacted to identify rows, columns, or separate tables; if tags are separated by a space, their levels are appended to identify rows, columns or separate tables.

taglist contains

tagspec

tagspec taglist

tagspec contains

tag

tag`#`*tag*[`#`*tag*[...]]

tag contains

dimension

dimension`[`*levels*`]`

dimension is a dimension in the collection.

levels are levels of the corresponding dimension.

Distinguish between `[]`, which are to be typed, and [ ], which indicate optional arguments.

Options

`clear` resets the collection's layout information.

`warn` and `nowarn` control the display of notes when `collect` encounters a tag it does not recognize.

`warn`, the default, specifies that `collect` display notes when it encounters a tag it does not recognize.

`nowarn` specifies that `collect` not show the notes.

These options override the `collect_warn` setting; see [TABLES] **set collect_warn**.

Remarks and examples

After collecting results from Stata commands using `collect get` or the `collect` prefix, we can arrange results into a table using `collect layout`.

The values in a collection are organized by their associated tags. These tags allow us to lay out a table by specifying which tags we wish to put on the rows and columns. More specifically, tags have two parts, the dimension and the level within the dimension. For example, the dimensions may represent types of statistics and covariate names. The levels within those dimensions may be coefficients, standard errors, and test statistics and `x1`, `x2`, and `x3`. The dimension for our statistics is called `result`, and the dimension for our covariates is called `colname`, so we can type

```
. collect layout (colname) (result)
```

to lay out a table with the covariates on the rows and the statistics on the columns. In this case, we specified only the dimension name, so the levels of each dimension that appear in the table are those that are selected as automatic levels. These automatic levels may be decided by the default collection style by specifying the levels you are interested in at the time you `collect` the results or by specifying automatic levels using `collect style autolevels`. If you want to see levels other than the automatic levels in your table, you can specify both the dimensions and their levels in your layout. For instance, we might type

```
. collect layout (colname[x1 x2]) (result[_r_b _r_se])
```

to lay out a table with variables `x1` and `x2` appearing on the rows and with statistics `_r_b` and `_r_se`, the coefficients and standard errors, appearing on the columns.

To demonstrate, we use data from the Second National Health and Nutrition Examination Survey (NHANES II) (McDowell et al. 1981). Below, we fit a model for systolic blood pressure as a function of age and weight. We use the `collect` prefix to collect the results, and we specify the `quietly` prefix to suppress the output.

```
. use https://www.stata-press.com/data/r19/nhanes2
. quietly: collect: regress bpsystol age weight
. collect dims
Collection dimensions
Collection: default
─────────────────────────────────────────
                  Dimension   No. levels
─────────────────────────────────────────
Layout, style, header, label
                     cmdset   1
                      coleq   1
                    colname   3
              program_class   1
                     result   33
                result_type   3
                    rowname   1
Style only
               border_block   4
                  cell_type   4
─────────────────────────────────────────
```

After collecting the results, we used `collect dims` to list the dimensions in our collection. These can be used to specify the rows and columns of our table. Let's put `colname`, the dimension containing covariates, on the rows and `result`, the dimension containing types of statistics, on the columns. If we type

```
. collect layout (colname) (result)
```

all covariates will be placed on the rows and all covariate-specific statistics will be reported on the columns. These statistics include the coefficient, confidence interval, test statistic, and p-value and more. This creates a wide table, and we likely want only a subset of these statistics reported.

Say that we instead wanted to include both the coefficient and its standard error. We could use `collect label list` to find that the levels of `result` that represent the reported coefficients and standard errors are `_r_b` and `_r_se`. We request these levels by typing

```
. collect layout (colname) (result[_r_b _r_se])
Collection: default
      Rows: colname
   Columns: result[_r_b _r_se]
   Table 1: 3 x 2
────────────────────────────────────
             │ Coefficient Std. error
─────────────┼──────────────────────
Age (years)  │    .6379892   .0111315
Weight (kg)  │    .4069041   .0124786
Intercept    │    71.27096   1.041742
────────────────────────────────────
```

Often we will want more than one dimension on the rows or columns. To demonstrate, we first collect results from another regression that includes the interaction between age and weight.

```
. collect: quietly: regress bpsystol age weight c.age#c.weight
```

Now we can place both the `colname` and `result` dimensions on the rows. We separate the dimension names by # to specify that the interacted levels should form the rows. If they were separated by a space, this would mean that we want to first see the levels of `colname` followed by the levels of `result`, but this is not what we want and would not uniquely identify the values corresponding to coefficients and standard errors in our collection. After collecting the results from the second regression, we have two levels of the dimension cmdset that identify the two commands we ran. We will put this dimension on the columns.

```
. collect layout (colname#result[_r_b _r_se]) (cmdset)

Collection: default
      Rows: colname#result[_r_b _r_se]
   Columns: cmdset
   Table 1: 12 x 2

--------------------------------------------------
                          |         1         2
--------------------------+-----------------------
Age (years)               |
  Coefficient             |  .6379892  .8898576
  Std. error              |  .0111315  .0536198
Weight (kg)               |
  Coefficient             |  .4069041  .5733109
  Std. error              |  .0124786  .0368295
Age (years) # Weight (kg) |
  Coefficient             |            -.003581
  Std. error              |            .0007458
Intercept                 |
  Coefficient             |  71.27096  59.60983
  Std. error              |  1.041742   2.64211
--------------------------------------------------
```

collect layout also allows you to build multiple tables. With this collection, we could, for instance, create a separate table for each of the models and again put variable names on the rows and statistics on the columns.

```
. collect layout (colname) (result[_r_b _r_se]) (cmdset)

Collection: default
      Rows: colname
   Columns: result[_r_b _r_se]
    Tables: cmdset
   Table 1: 3 x 2
   Table 2: 4 x 2

1
------------------------------------
            | Coefficient Std. error
------------+-----------------------
Age (years) |    .6379892   .0111315
Weight (kg) |    .4069041   .0124786
Intercept   |    71.27096   1.041742
------------------------------------

2
--------------------------------------------------
                          | Coefficient Std. error
--------------------------+-----------------------
Age (years)               |    .8898576   .0536198
Weight (kg)               |    .5733109   .0368295
Age (years) # Weight (kg) |    -.003581   .0007458
Intercept                 |    59.60983    2.64211
--------------------------------------------------
```

If you have a layout that you prefer to use for many of the tables you create, you can save the layout along with any preferred styles with collect style save. Then, after collecting new results, you can use collect style use to apply the same layout to the new collection.

Stored results

collect layout stores the following in s():

Macros

s(collection)	name of collection
s(rows)	rows specification
s(columns)	columns specification
s(tables)	tables specification
s(k_tables)	number of tables
s(table#)	layout for the #th table

References

Huber, C. 2021. Customizable tables in Stata 17, part 3: The classic table 1. *The Stata Blog: Not Elsewhere Classified.* https://blog.stata.com/2021/06/24/customizable-tables-in-stata-17-part-3-the-classic-table-1/.

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

[TABLES] **collect get** — Collect results from a Stata command

[TABLES] **collect preview** — Preview the table in a collection

[TABLES] **collect style save** — Save collection styles to disk

[TABLES] **collect style use** — Use collection styles from disk

collect preview — Preview the table in a collection

Description

`collect preview` displays a preview of a table in a collection based on the specified layout, labels, and styles.

Syntax

`collect preview` [, `name(`*cname*`)` `noblank`]

where *cname* is the name of an existing collection.

Options

`name(`*cname*`)` specifies the collection for which a preview should be displayed. By default, the preview is for a table in the current collection.

`noblank` prevents `collect preview` from putting a blank line before its output. By default, `collect preview` will insert a blank line before its output.

Remarks and examples

Building and customizing a table is often an iterative process. `collect preview` allows you to see what your table looks like at each step of that process.

To demonstrate, we first collect results using the `collect` prefix and lay out a table using `collect layout`.

```
. use https://www.stata-press.com/data/r19/nhanes2
. quietly: collect _r_b: regress bpsystol bmi
. quietly: collect _r_b: regress bpsystol bmi age
. collect layout (colname) (cmdset#result)
Collection: default
      Rows: colname
   Columns: cmdset#result
   Table 1: 3 x 2

-------------------------------------------------
                      |           1           2
                      | Coefficient Coefficient
----------------------+--------------------------
Body mass index (BMI) |    1.656894    1.304128
Age (years)           |                .5883367
Intercept             |    88.56855    69.58451
-------------------------------------------------
```

When we use `collect layout`, we automatically get a preview of our table. However, after other commands that we use to make changes to this table, we need to request a preview to see the results. For instance, we might change labels.

```
. collect label levels cmdset 1 "Model 1" 2 "Model 2"
```

To see the effect of that change, we type

```
. collect preview

----------------------------------------------------
                      |     Model 1     Model 2
                      | Coefficient Coefficient
----------------------+-----------------------------
Body mass index (BMI) |    1.656894    1.304128
Age (years)           |                .5883367
Intercept             |    88.56855    69.58451
----------------------------------------------------
```

There are many other style and label changes that we could make. For instance, here we hide `Coefficient` from column headers, add extra space between columns, and specify a numeric format. Following those changes, we again preview the table to see the results.

```
. collect style header result, level(hide)
. collect style column, extraspace(1)
. collect style cell, nformat(%6.2f)
. collect preview

-------------------------------------------
                      |  Model 1   Model 2
----------------------+--------------------
Body mass index (BMI) |     1.66      1.30
Age (years)           |               0.59
Intercept             |    88.57     69.58
-------------------------------------------
```

In fact, we might have typed `collect preview` after each of these commands to see the result.

Alternatively, we could point and click to make changes to the table and automatically see a preview of the table with modifications in the Tables Builder.

For information on style and label commands we used here, see [TABLES] **collect label**, [TABLES] **collect style header**, [TABLES] **collect style column**, and [TABLES] **collect style cell**.

Stored results

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

Macros

`s(collection)`	name of collection
`s(rows)`	rows specification
`s(columns)`	columns specification
`s(tables)`	tables specification
`s(k_tables)`	number of tables
`s(table#)`	layout for the #th table

Also see

[TABLES] **Tables Builder** — Tables Builder

Description Quick start Menu Syntax Options
Remarks and examples Stored results Also see

Description

collect export exports a table from a collection to a specified document type.

You can also include a table in a report created by putdocx, putpdf, or putexcel; see [RPT] **putdocx collect**, [RPT] **putpdf collect**, and [RPT] **putexcel**.

Quick start

Export a table from the current collection to myfile.xlsx

```
collect export myfile.xlsx
```

Export a table from the collection c2 to myfile2.xlsx

```
collect export myfile2.xlsx, name(c2)
```

Same as above, but export the table to sheet Table1, instead of the default Sheet1

```
collect export myfile2.xlsx, name(c2) sheet(Table1)
```

Export a table from the current collection to table1.docx, replacing the file if it exists

```
collect export table1.docx, replace
```

Same as above, and save the putdocx commands used to export the table in table1.do

```
collect export table1.docx, replace dofile(table1)
```

Export a table from the current collection to table1.tex

```
collect export table1.tex
```

Same as above, but export the table only, instead of creating a complete LaTeX document

```
collect export table1.tex, tableonly
```

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Build and style table

Syntax

`collect export` *filename*.*suffix* [, *export_options document_options*]

export_options	Description
`name(`*cname*`)`	export collection *cname*
`as(`*fileformat*`)`	specify document type
`replace`	overwrite existing file

By default, `collect export` will try to determine the document type from *suffix* using the following table:

suffix	Implied option	File type
`docx`	`as(docx)`	Microsoft Word
`html`	`as(html)`	HTML 5 with CSS
`pdf`	`as(pdf)`	PDF (Portable Document Format)
`xlsx`	`as(xlsx)`	Microsoft Excel 2007/2010 or newer
`xls`	`as(xls)`	Microsoft Excel 1997/2003
`tex`	`as(tex)`	LaTeX
`smcl`	`as(smcl)`	SMCL (Stata Markup and Control Language)
`txt`	`as(txt)`	plain text
`markdown`	`as(markdown)`	Markdown
`md`	`as(markdown)`	Markdown

document_options	Description
docx_options	when exporting to `.docx` files
html_options	when exporting to `.html` files
pdf_options	when exporting to `.pdf` files
excel_options	when exporting to `.xls` and `.xlsx` files
tex_options	when exporting to `.tex` files
smcl_option	when exporting to `.smcl` files
txt_option	when exporting to `.txt` files
md_option	when exporting to `.markdown` and `.md` files

docx_options	Description
`noisily`	show the `putdocx` commands used to export to the `.docx` file
`dofile(`*filename*[`, replace`]`)`	save the `putdocx` commands used for exporting to the named do-file

html_options	Description
`prefix(`*prefix*`)`	use *prefix* to identify style classes
`cssfile(`*cssfile*`)`	define the styles in *cssfile* instead of *filename*
`tableonly`	export only the table to the specified file
`append`	append to an existing file

pdf_options	Description
noisily	show the putpdf commands used to export to the PDF file
dofile(*filename*[, replace])	save the putpdf commands used for exporting to the named do-file

excel_options	Description
noisily	show the putexcel commands used to export to the Excel file
dofile(*filename*[, replace])	save the putexcel commands used for exporting to the named do-file
cell(*cell*)	specify the Excel upper-left cell as the starting position to export the table; the default is cell(A1)
modify	modify Excel file
sheet(*sheetname* [, replace])	specify the worksheet to use; the default sheet name is Sheet1
noopen	do not open Excel file in memory

noopen does not appear in the dialog box.

tex_options	Description
tableonly	export only the table to the specified file
append	append to an existing file

smcl_option	Description
append	append to an existing file

txt_option	Description
append	append to an existing file

md_option	Description
append	append to an existing file

Options

Options are presented under the following headings:

export_options
docx_options
html_options
pdf_options
excel_options
tex_options
smcl_option
txt_option
md_option

export_options

`name(`*cname*`)` specifies a collection to export instead of the current collection.

`as(`*fileformat*`)` specifies the file format to which the collection is to be exported. This option is rarely specified because, by default, `collect export` determines the format from the suffix of the file being created.

`replace` permits `collect export` to overwrite an existing file.

docx_options

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

`dofile(`*filename*[`, replace`]`)` specifies that `collect export` 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.

html_options

`prefix(`*prefix*`)` specifies that `collect export` use *prefix* to identify style classes for the exported HTML table.

`cssfile(`*cssfile*`)` specifies that `collect export` define the styles in *cssfile* instead of *filename*.

`tableonly` specifies that only the table be exported to the specified file. With this option, the produced file may be included in other HTML documents. By default, `collect export` produces a complete HTML document.

If option `cssfile()` is not specified, a CSS filename is constructed from *filename*, with the extension replaced with `.css`.

`append` specifies that `collect export` append the table to an existing file.

This option implies option `tableonly`.

If the target CSS file already exists, `collect export` will also append to it.

pdf_options

`noisily` specifies that `collect export` show the `putpdf` commands used to export to the PDF file.

`dofile(`*filename*[`, replace`]`)` specifies that `collect export` 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.

excel_options

`noisily` specifies that `collect export` show the `putexcel` commands used to export to the `.xls` or `.xlsx` file.

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

`cell(`*cell*`)` specifies an Excel upper-left cell as the starting position to publish the table. The default is `cell(A1)`.

`modify` permits `putexcel set` to modify an Excel file. For more information about this option, see [RPT] **putexcel**.

`sheet(`*sheetname* [`, replace`]`)` saves to the worksheet named *sheetname*. For more information about this option, see [RPT] **putexcel**.

`noopen` prevents `collect export` from using option `open` in the call to `putexcel set`.

By default, `collect export` uses the `putexcel set` option `open` to open the Excel file in memory for modification. This tends to improve the speed of the export if many cells or style edits are in the collection. For more information about the `open` option, see [RPT] **putexcel**.

`noopen` is necessary only when you need to force `collect export` to produce do-files as it did when `collect export` was first introduced in Stata 17.

tex_options

`tableonly` specifies that only the table be exported to the specified file. With this option, the produced file may be included in other LaTeX documents via the `\input` or `\include` macro.

`append` specifies that `collect export` append the table to an existing file.

This option implies option `tableonly`.

smcl_option

`append` specifies that `collect export` append the table to an existing file.

txt_option

`append` specifies that `collect export` append the table to an existing file.

md_option

`append` specifies that `collect export` append the table to an existing file.

Remarks and examples

Remarks are presented under the following headings:

Introduction

One goal of creating a customized table may be to present your findings to others. With `collect export`, you can export a collection to a variety of file types. For example, after creating a table from a collection of results and making styling edits to obtain the look you want, you can export a table from the current collection to an Excel file by typing the following:

```
. collect export myfile.xlsx
```

By specifying the `.xlsx` suffix, we have indicated that we want to export our work to a Microsoft Excel file. Equivalently, we could have instead specified the document type as follows:

```
. collect export myfile, as(xlsx)
```

Either way, we would have exported a table from the current collection to the file `myfile.xlsx`.

Styles for different documents

The `collect` suite of commands has many formatting features that can be applied to any collection of results, regardless of the document you may be exporting your table to. For example, you may specify the numeric formatting for your results or modify the labels for the dimensions. But there are also some style specifications that are specific to the type of document you will be exporting to. For example, you can use `collect style html` to specify whether adjacent cell borders should be collapsed in the resulting HTML file. If you will be exporting a table to a `.docx` or `.pdf` file, you can see `collect style putdocx` and `collect style putpdf` for some style specifications specific to those types of documents.

Additionally, you can add a custom table title with `collect title` and notes with `collect notes`. These commands add the title and notes in plain text, but you can specify the appearance style for the title and notes, respectively, with `collect style title` and `collect style notes`. Certain appearance edits can be rendered only on certain types of documents. For example, you can specify the font size and color if you will export your table to a Microsoft Excel, an HTML, or a LaTeX file.

Creating more extensive documents

With `collect export`, you can export a customized table to the file format of your choice. If you are exporting to an HTML, a LaTeX, a SMCL, a Markdown, or a plain text file, you can export multiple tables to a single file with the `append` option. But you may want to create documents with more than these customized tables.

For example, when you export a table to an HTML or a LaTeX file, `collect export` creates a complete document with the table from the current collection. If you want to incorporate this table in a more extensive document, you can use the `tableonly` option to export just the table to the specified file.

Additionally, suppose that you wish to export your table to a `.docx` or `.pdf` file but you want that table to be part of a report that also includes graphs, text, and other results from Stata. By using the `putdocx` and `putpdf` suites, you can incorporate the customized table in the active document with `putdocx collect` and `putpdf collect`.

Similarly, if you want to create an Excel file with the table from the current collection and other Stata results or graphs, you can incorporate the table in the active Excel file by using the `collect` output type with `putexcel`.

Stored results

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

Macros

`s(filename)`	name of the file
`s(collection)`	name of collection
`s(noisily)`	1 if option `noisily` specified, 0 otherwise
`s(dofile)`	name of the new do-file
`s(cssfile)`	name of the new css-file

Also see

[TABLES] **collect style html** — Collection styles for HTML files

[TABLES] **collect style putdocx** — Collection styles for putdocx

[TABLES] **collect style putpdf** — Collection styles for putpdf

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

[RPT] **putexcel** — Export results to an Excel file

[RPT] **putpdf collect** — Add a table from a collection to a PDF file

collect style autolevels — Collection styles for automatic dimension levels

Description Quick start Menu Syntax
Options Remarks and examples Stored results Also see

Description

`collect style autolevels` specifies the levels of a dimension that will be automatically displayed when the dimension is included in a table.

Quick start

Display levels `lev1`, `lev2`, and `lev3` automatically when dimension `dim1` is included in the table

```
collect style autolevels dim1 lev1 lev2 lev3
```

Display coefficients `_r_b` and standard errors `_r_se` automatically when dimension `result` is included in the table after collecting results from a regression model

```
collect style autolevels result _r_b _r_se
```

Display means `mu_1` and `mu_2` and *p*-value `p` automatically when dimension `result` is included in the table, and clear previous automatic results

```
collect style autolevels result mu_1 mu_2 p, clear
```

Clear automatic levels for dimension result

```
collect style autolevels result, clear
```

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Build and style table

Syntax

`collect style autolevels` *dim* [*levels*] [, `name(`*cname*`)` `clear`]

where *cname* is a collection name, *dim* is a dimension in the specified collection, and *levels* specifies one or more levels of this dimension.

Options

`name(`*cname*`)` specifies a collection *cname* to which the style is applied. By default, the style is applied to the current collection.

`clear` removes existing `collect style autolevels` properties.

Remarks and examples

`collect style autolevels` determines the levels of a dimension to be included in a table when no levels are specified in `collect layout` and when no automatic levels were specified using `collect get` or the `collect` prefix at the time results were collected.

When results are collected using `collect get` or the `collect prefix`, no automatic levels are applied to the dimensions in the collection by default.

When you use the `table` command to create a table, its results are automatically stored in a collection. When the `command()` option is specified with `table`, it will run another Stata command and include the results in the table. If the specified command is an r-class command, all scalars stored in `r()` are set as automatic levels. If the specified command is an estimation (e-class) command, the reported coefficients (`_r_b`) are set as automatic levels.

As an example, we consider results collected from `regress`. At the time we collect results, we can specify automatic levels for the `result` dimension. For instance, we could type

```
. collect _r_b _r_ci: regress y x1 x2
```

or

```
. regress y x1 x2
. collect get _r_b _r_ci
```

to specify that the reported coefficients and confidence intervals should be reported in the table.

However, we may instead collect results without specifying automatic levels. We might type

```
. collect: regress y x1 x2
```

or

```
. regress y x1 x2
. collect get e()
```

Now, there are no automatic levels for the `result` dimension. Therefore, if we include this dimension in a table layout by typing, for instance,

```
. collect layout (colname) (result)
```

all levels of `result` with values that can be identified by the `colname` and `result` dimensions will be included in the table. If we want only the coefficients and confidence intervals in our tables, we can specify this with `collect layout`.

```
. collect layout (colname) (result[_r_b _r_ci])
```

This is convenient enough if we are building a single table. However, if we plan to build multiple tables from this collection and we want to display coefficients and confidence intervals in each one, we could instead type

```
. collect style autolevels result _r_b _r_ci
```

to specify the automatic levels to be included for this dimension.

Now, we can simply type

```
. collect layout (colname) (result)
```

to create the desired table.

Moreover, if we create many similar tables even with different collections of results, we can use `collect style save` to save a file with this autolevels style along with any others we prefer. Then, with future collections, we can use `collect style use` to apply this style to future collections and tables.

Stored results

`collect style autolevels` stores the following in `s()`:

Macros

`s(collection)`	name of collection
`s(dimname)`	specified dimension
`s(levels)`	specified dimension levels

Also see

[TABLES] **collect get** — Collect results from a Stata command

[TABLES] **collect layout** — Specify table layout for the current collection

[TABLES] **collect query** — Query collection style properties

[TABLES] **collect style save** — Save collection styles to disk

[TABLES] **collect style use** — Use collection styles from disk

collect style cell — Collection styles for cells

Description

`collect style cell` specifies the cell appearance styles in the collection. This includes numeric formats, borders, bolding, italics, font, text color, cell color, margins, justification, and more. These styles can be applied to all cells in the collection, to cells of a particular dimension, or to specific cells of a particular dimension. Certain appearance edits can be rendered only on certain export formats.

Quick start

Use a comma as the delimiter for the upper and lower bounds of confidence intervals

```
collect style cell result[_r_ci], cidelimiter(,)
```

Set the cell margin for all cells in the collection to 10 points on the left and right

```
collect style cell, margin(left right, width(10))
```

Format standard errors and coefficients with two decimal places

```
collect style cell result[_r_se _r_b], nformat(%8.2f)
```

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Build and style table

Syntax

`collect style cell` [*taglist*] [, *options*]

taglist is a list of tags that identify table cells to which styles are to be applied. Within the *taglist*, if tags are joined by #, cells identified by all of these specified tags are selected; if tags are separated by a space, cells identified by any of these tags are selected. If no *taglist* is specified, styles are applied to all cells.

taglist contains

tagspec

tagspec taglist

tagspec contains

tag

tag#*tag*[#*tag*[...]]

tag contains

dimension

dimension[*levels*]

dimension is a dimension in the collection.

levels are levels of the corresponding dimension.

Distinguish between [ ], which are to be typed, and [ ], which indicate optional arguments.

options	Description
Main	
name(*cname*)	apply cell appearance styles to collection *cname*
basestyle	update base style
[no]warn	display or suppress notes about tags that are not recognized; default is to display
Borders	
border(*bspec*)	set pattern, color, and width for cell border; option may be repeated
Diagonals	
dborder(*dbspec*)	set direction, pattern, and color for cell diagonal border
Fonts	
font([*fontfamily*][, *font_opts*])	set font style for cell text
smcl(*smcl*)	specify formatting for SMCL files
latex(*latex*)	specify LaTeX macro
Shading	
shading(*sspec*)	set background color, foreground color, and fill pattern for cells
Margins	
margin(*mspec*)	set margins for cells; option may be repeated
Alignments	
halign(*hvalue*)	set horizontal alignment for cells
valign(*vvalue*)	set vertical alignment for cells
Formats	
nformat(*%fmt*)	specify numeric format for cell text
sformat(*sfmt*)	specify string format for cell text
minimum([#][, label(*string*)])	specify minimum value to report
maximum([#][, label(*string*)])	specify maximum value to report
empty(*string*)	specify text to place in empty cells
cidelimiter(*char*)	use character as delimiter for confidence interval limits
cridelimiter(*char*)	use character as delimiter for credible interval limits

font_opts	Description
size(# [*unit*])	specify font size
color(*color*)	specify font color
variant(*variant*)	specify font variant and capitalization
[no]bold	specify whether to format text as bold
[no]italic	specify whether to format text as italic
[no]strikeout	specify whether to strike out text
underline(*upattern*)	specify whether to underline text

bspec is

[*borders*] [, width(*bwidth*) pattern(*bpattern*) color(*bcolor*)]

borders specifies one or more border locations and identifies where to apply the border style edits.

bwidth is defined as # [*unit*] and specifies the border line width. If # is specified without the optional *unit*, points is assumed.

bpattern is a keyword specifying the look of the border. The default is single. For a complete list of border patterns, see *Border patterns* of [TABLES] **Appendix**. To remove an existing border, specify nil as the *bpattern*.

bcolor specifies the border color.

dbspec is

direction [, pattern(*dbpattern*) color(*dbcolor*)]

direction specifies the diagonal border direction and may be one of down, up, or both.

dbpattern is a keyword specifying the look of the diagonal border. The default is thin. For a complete list of diagonal border patterns, see *Diagonal border patterns* of [TABLES] **Appendix**.

dbcolor specifies the diagonal border line color.

sspec is

[background(*bgcolor*) foreground(*fgcolor*) pattern(*fpattern*)]

bgcolor specifies the background color.

fgcolor specifies the foreground color.

fpattern specifies the fill pattern. A complete list of fill patterns is shown in *Shading patterns* of [TABLES] **Appendix**.

fontfamily specifies a valid font family.

unit may be in (inch), pt (point), or cm (centimeter). An inch is equivalent to 72 points and 2.54 centimeters. The default is pt.

variant may be allcaps, smallcaps, or normal.

variant(allcaps) changes the text to all uppercase letters; applicable when publishing items from a collection to Microsoft Word, PDF, LaTeX, and HTML files.

variant(smallcaps) changes the text to use large capitals for uppercase letters and smaller capitals for lowercase letters; applicable when publishing items from a collection to Microsoft Word, LaTeX, and HTML files.

variant(normal) changes the font variant back to normal and leaves the capitalization unchanged from the original text; applicable when publishing items from a collection to Microsoft Word, PDF, LaTeX, and HTML files.

bcolor, *dbcolor*, *bgcolor*, *fgcolor*, and *color* may be one of the colors listed in *Colors* of [TABLES] **Appendix**; 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

Main

`name(`*cname*`)` specifies a collection *cname* to which appearance styles are applied.

`basestyle` indicates that the appearance styles be applied to the base style, instead of overriding the current style for the specified cells.

Each cell begins with baseline style properties. (You can view your table with these baseline style properties by first clearing out the collection styles with `collect style clear`.) The appearance of the cells is then updated with any changes specified in the default style used by `collect` and `table`. Any `collect style cell` command you issue will override the current style for the specified cells. If you specify the `basestyle` option, the style changes will instead apply to the baseline style and they will not override any current style edits targeted to specific tags.

For example, suppose you have created a table with coefficients, standard errors, p-values, confidence intervals, and R^2 values. You then format your coefficients to display only two digits after the decimal. If you then decide to format all other statistics to display only three digits, you can type `collect style cell result, basestyle nformat(%9.3f)` to apply this change while retaining the formatting you applied to the coefficients.

`warn` and `nowarn` control the display of notes when `collect` encounters a tag it does not recognize. The notes are displayed by default unless you used `set collect_warn off` to suppress them. `warn` specifies that `collect` show the notes. `nowarn` specifies that `collect` not show the notes. These options override the `collect_warn` setting; see [TABLES] **set collect_warn**.

Borders

`border(`[*borders*] [`,` `width(`*bwidth*`)` `pattern(`*bpattern*`)` `color(`*bcolor*`)`]`)` specifies line styles for cell borders. *borders* specifies one or more border locations and identifies where to apply the border style edits. The border locations are `left`, `right`, `top`, `bottom`, or `all`. If *borders* is not specified, `all` is assumed. You may change the width, pattern, and color for the border by specifying *bwidth*, *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. Additionally, these border style properties are applicable when publishing items from a collection to all file types, except Markdown.

Diagonals

`dborder(`*direction* [`,` `pattern(`*dbpattern*`)` `color(`*dbcolor*`)`]`)` specifies line styles for diagonal cell borders. The direction of the diagonal border is specified by *direction*, which may be `down`, `up`, or `both`. Optionally, you may change the pattern and color for the border by specifying *dbpattern* and *dbcolor*.

These diagonal border style properties are applicable when publishing items from a collection to a Microsoft Excel file.

Fonts

`font(`[*fontfamily*] [`,` `size(`# [*unit*]`)` `color(`*color*`)` `variant(`*variant*`)` [`no`]`bold` [`no`]`italic` [`no`]`strikeout` [`no`]`underline` `underline(`*upattern*`)`]`)` specifies the font style for the cell text.

These font style properties are applicable when publishing items from a collection to Microsoft Word, Microsoft Excel, PDF, LaTeX, and HTML files, unless otherwise specified.

fontfamily specifies a valid font family. This font style property is applicable when publishing items from a collection to Microsoft Word, Microsoft Excel, PDF, and HTML files.

`size(#` [*unit*]`)` specifies the font size as a number optionally followed by units. If # is specified without the optional *unit*, points is assumed. This font style property is applicable when publishing items from a collection to Microsoft Word, Microsoft Excel, PDF, and HTML files.

`variant(`*variant*`)` specifies the font variant and capitalization.

`bold` and `nobold` specify the font weight. `bold` changes the font weight to bold; `nobold` changes the font weight back to normal.

`italic` and `noitalic` specify the font style. `italic` changes the font style to italic; `noitalic` changes the font style back to normal.

`strikeout` and `nostrikeout` specify whether to add a strikeout mark to the text. `strikeout` adds a strikeout mark to the text; `nostrikeout` changes the text back to normal.

Only one of `strikeout` or `underline` is allowed when publishing to HTML files.

`underline(`*upattern*`)`, `underline`, and `nounderline` specify how to underline the text.

`underline(`*upattern*`)` adds an underline to the text using a specified pattern. *upattern* may be any of the patterns listed in *Underline patterns* of [TABLES] **Appendix**. For example, `underline(none)` removes the underline from the text, and `underline(single)` underlines the test. All other *upattern*s are available only when publishing items from a collection to Microsoft Word.

Only one of `strikeout` or `underline` is allowed when publishing to HTML files; underline patterns are not allowed when publishing to HTML files.

`smcl(`*smcl*`)` specifies how to render cell text for SMCL output. The supported SMCL directives are `input`, `error`, `result`, and `text`. This style property is applicable only when publishing items from a collection to a SMCL file.

`latex(`*latex*`)` specifies the name of a LaTeX macro to render cell text for LaTeX output. This style property is applicable only when publishing items from a collection to a LaTeX file.

Example LaTeX macro names are `textbf`, `textsf`, `textrm`, and `texttt`. Custom LaTeX macros are also allowed. If *value* is the value for a given cell, then *latex* is translated to the following when exporting to LaTeX:

latex{*value*}

Shading

`shading(`[`background(`*bgcolor*`) foreground(`*fgcolor*`) pattern(`*fpattern*`)`]`)` sets the background color, foreground color, and fill pattern for cells. The background color is applicable when exporting the table to Microsoft Word, Microsoft Excel, PDF, HTML, and LaTeX files. The foreground color and fill pattern are applicable when exporting the table to Microsoft Word and Microsoft Excel.

Margins

`margin(`[*margins*] [`, width(#` [*unit*]`)`]`)` specifies margins inside the cell.

These margin style properties are applicable when publishing items from a collection to PDF and HTML files.

margins specifies one or more margin locations and identifies where to apply the margin style edits. The margin locations are `left`, `right`, `top`, `bottom`, and `all`. If *margins* is not specified, `all` is assumed.

`width(#` [*unit*]`)` specifies the margin width as a number optionally followed by units.

Alignments

`halign(`*hvalue*`)` specifies the horizontal alignment for the cell text. *hvalue* may be `left`, `center`, and `right`.

These alignment style properties are applicable when publishing items from a collection to all file types, except Markdown.

`valign(`*vvalue*`)` specifies the vertical alignment for the cell text. *vvalue* may be `top`, `bottom`, or `center`.

These alignment style properties are applicable when publishing items from a collection to all file types, except Markdown.

Formats

`nformat(`*%fmt*`)` applies the Stata numeric format *%fmt* to cell text constructed from numeric items.

`sformat(`*sfmt*`)` applies a string format to cell text. You can, for instance, add symbols or text to the values reported in the collection by modifying the string format.

sfmt may contain a mix of text and `%s`. Here `%s` refers to the numeric value that is formatted as specified using `nformat()`. The text will be placed around the numeric values in the collection as it is placed around `%s` in this option. For instance, to place parentheses around results, you can specify `sformat("(%s)")`.

Two text characters must be specified using a special character sequence if you want them to be displayed in your collection. To include %, type %%. To include \, type \\. For instance, to place a percent sign after results, you can specify `sformat("%s%%")`.

`minimum(`[#][, `label(`*string*`)`]`)` specifies that numeric items less than # be displayed as "<#", where # is formatted according to `nformat()`.

If suboption `label(`*string*`)` is specified, then "*string*" is used instead of "<#". If *string* contains `%s`, then `%s` is replaced by # formatted according to `nformat()`.

If suboption `label()` is not specified, it effectively defaults to `label("<%s")`.

`maximum(`[#][, `label(`*string*`)`]`)` specifies that numeric items greater than # be displayed as ">#", where # is formatted according to `nformat()`.

If suboption `label(`*string*`)` is specified, then "*string*" is used instead of ">#". If *string* contains `%s`, then `%s` is replaced by # formatted according to `nformat()`.

If suboption `label()` is not specified, it effectively defaults to `label(">%s")`.

`empty(`*string*`)` specifies text to place in empty cells.

`cidelimiter(`*char*`)` changes the delimiter between confidence interval limits. The default is `cidelimiter(" ")`.

`cridelimiter(`*char*`)` changes the delimiter between credible interval limits. The default is `cridelimiter(" ")`.

Remarks and examples

`collect style cell` allows you to specify the cell appearance styles for tables built from the collection. These styles include the numeric format for results, borders around cells, font, and much more. If you do not specify a tag, your appearance style will be applied to all cells in the table, including those in the body of the table and the headers.

Stored results

`collect style cell` stores the following in `s()`:

Macros
`s(collection)` name of collection

References

Huber, C. 2021a. 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/.

———. 2021b. Customizable tables in Stata 17, part 3: The classic table 1. *The Stata Blog: Not Elsewhere Classified*. https://blog.stata.com/2021/06/24/customizable-tables-in-stata-17-part-3-the-classic-table-1/.

Also see

[TABLES] **collect query** — Query collection style properties

[TABLES] **collect style column** — Collection styles for column headers

[TABLES] **collect style row** — Collection styles for row headers

collect style clear — Clear all collection styles

Description

`collect style clear` clears all collection styles defined in the current (active) collection, including the default style, and specifies that `collect` use an empty style.

Syntax

`collect style clear`

Remarks and examples

`collect style clear` clears all collection styles defined in the current collection, including the default style, and specifies that `collect` use an empty style. Note that this command does not affect the styles for other collections you have in memory. This command is rarely used because a table produced using the empty style will typically need many style edits to be complete. For instance, with the empty style, a table will display the title for each dimension, which is typically not needed.

If you are in the process of creating a table and have made several changes to the appearance style but you wish to return to the default style, you can load the default style with `collect style use`.

Stored results

`collect style clear` stores the following in `s()`:

Macros
    `s(collection)`   name of collection

Also see

[TABLES] **collect style use** — Use collection styles from disk

collect style column — Collection styles for column headers

Description Quick start Menu Syntax Options
Remarks and examples Stored results Reference Also see

Description

`collect style column` specifies column header style properties. `collect style column` determines how factor variables are displayed in column headers, how duplicates are reported, whether headers are filled from top to bottom or from bottom to top, and the width and spacing of columns.

Quick start

Specify that repeating headers be displayed only once and centered horizontally

```
collect style column, dups(center)
```

Same as above, and set all the columns to have the same width

```
collect style column, dups(center) width(equal)
```

Use an `x` to delimit interaction terms

```
collect style column, delimiter(" x ")
```

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Build and style table

Syntax

`collect style column` [, *options*]

options	Description
Main	
`name(`*cname*`)`	specify column header styles for collection *cname*
`nodelimiter`	place factor-variable and interaction elements in separate cells without a delimiter
`delimiter(`*delim*`)`	use *delim* to delimit interaction terms composed in a single cell
`atdelimiter(`*atdelim*`)`	use *atdelim* to delimit interaction terms containing the `@` symbol
`bardelimiter(`*bardelim*`)`	use *bardelim* to delimit interaction terms containing the `|` symbol
`binder(`*binder*`)`	use *binder* to separate factor variables from their levels
`dups(`*dups*`)`	specify how duplicate headers are displayed
`position(`*colpos*`)`	specify the position of the column header to be filled first
SMCL/Text	
`extraspace(#)`	specify the number of extra spaces between columns in SMCL and plain text
`width(`*widthspec*`)`	specify how to distribute column widths

Options

Main

`name(`*cname*`)` specifies the collection to which column header style properties are to be applied. By default, properties are applied to the current collection.

`nodelimiter`, `delimiter()`, `atdelimiter()`, and `bardelimiter()` control how to compose factor-variable and interaction terms in headers.

`nodelimiter` specifies that factor-variable and interaction term elements (matrix stripe elements) be split into separate cells.

`delimiter(`*delim*`)` specifies that factor-variable and interaction term elements (matrix stripe elements) be composed in a single cell.

The variables in an interaction term are composed in a single cell using *delim* as the delimiter.

Factor-variable terms serve as their own dimension nested within the stripe dimensions `coleq`, `colname`, `roweq`, and `rowname`. Option `binder()` controls how levels of factor variables are composed within a single cell.

`atdelimiter(`*atdelim*`)` specifies that *atdelim* be used to delimit interaction terms containing the `@` symbol. This option is applicable when, for example, working with results from `contrast`, `mean`, `proportion`, `ratio`, and `total`.

`bardelimiter(`*bardelim*`)` specifies that *bardelim* be used to delimit interaction terms containing the `|` symbol. This option is applicable when, for example, working with results from `anova` and `manova`.

`binder(`*binder*`)` specifies how to compose levels of factor variables within a single cell. *binder* will be used to separate factor variables from their levels.

The binder will be applied as long as the factor variable and its levels are not hidden. Note that the default style used by `collect`, which is `style-default.stjson`, will hide the dimension title from the headers. You can use `collect style header` to specify whether to display the label or name for a dimension and whether to display the label or value for the level of a dimension.

`dups(`*dups*`)` controls how to handle duplicate header elements. *dups* is one of `repeat`, `first`, or `center`.

`dups(repeat)`, the default, specifies that `collect` repeat duplicate header elements.

`dups(first)` specifies that `collect` hide all duplicate header elements, except the first.

`dups(center)` specifies that `collect` horizontally center duplicate header elements, where the header element spans the duplicate header cell locations. When this style is not supported, such as when exporting to Markdown, `dups(first)` is used instead.

`position(`*colpos*`)` specifies how column headers are filled in when one or more levels of a dimension occupy more than one cell. This option is used when factor variables are displayed in the column headers. *colpos* may be `top` or `bottom`.

`position(top)` is the default and specifies that `collect` fill in column headers starting with the topmost cell. This will result in some empty cells on the bottom for unbalanced column dimensions.

`position(bottom)` specifies that `collect` shift the column header cells to the bottom so that the cells in the last row are all filled in. This will result in some empty cells on the top for unbalanced column dimensions.

SMCL/Text

`extraspace(`*#*`)` specifies extra spaces to pad columns when exporting to SMCL and plain text. The first and middle columns get *#* extra spaces added on both sides. The last column gets *#* extra spaces added on the left. The default is `extraspace(0)`.

`width(`*widthspec*`)` specifies how to distribute the column widths for the items. Row header widths are not affected by this option.

`width(asis)`, the default, specifies that column widths be determined separately, with each column being as wide as necessary to accommodate the widest cell contents within that column.

`width(equal)` specifies that the item column widths all be equal to the widest cell contents among the items and column headers in all columns.

Remarks and examples

`collect style column` determines how factor variables are displayed in column headers, how duplicates are reported, whether headers are filled from top to bottom or from bottom to top, and the width and spacing of columns.

In the following examples, we explore some styles for column headers that may be of interest when working with factor variables and interactions.

▷ Example 1: Working with factor variables

Below, we use data from the Second National Health and Nutrition Examination Survey (NHANES II) (McDowell et al. 1981). We begin by fitting a model for systolic blood pressure as a function of agegrp. We collect the coefficients (_r_b) and use the quietly prefix to suppress the output. Then, we arrange the items in our collection with collect layout. We place the variable names (colname) on the columns and the statistics (result) on the rows:

```
. use https://www.stata-press.com/data/r19/nhanes2
. quietly: collect _r_b: regress bpsystol i.agegrp
. collect layout (result) (colname)
Collection: default
      Rows: result
   Columns: colname
   Table 1: 1 x 7
```

	Age group 20–29	Age group 30–39	Age group 40–49	Age group 50–59	Age group 60–69	Age group 70+	Intercept
Coefficient	0	2.89081	9.597631	18.32889	24.17618	30.82992	117.3466

Instead of having the repeated header for agegrp, let's specify that it be displayed only once and centered horizontally across the columns it applies to. Then, we will get a preview of our table:

```
. collect style column, dups(center)
. collect preview
```

	Age group						Intercept
	20–29	30–39	40–49	50–59	60–69	70+	
Coefficient	0	2.89081	9.597631	18.32889	24.17618	30.82992	117.3466

◁

▷ Example 2: Working with interactions

When working with models that contain interactions, you may want to specify the delimiter for the interaction terms. For example, below we create a new collection called interaction, which then becomes the current collection. Then, we fit a model with an interaction between race and sex, collecting only the coefficients. To keep the table from becoming too wide, we use collect style cell to format the coefficients to display only two digits after the decimal, and we suppress the display of the base levels.

```
. collect create interaction
(current collection is interaction)
. quietly: collect _r_b: regress bpsystol sex##race
. collect style cell, nformat(%6.2f)
. collect style showbase off
```

Then, we specify the same layout as we did in the previous example:

```
. collect layout (result) (colname)

Collection: interaction
      Rows: result
   Columns: colname
   Table 1: 1 x 6
```

	Sex Female	Race Black	Race Other	Sex Female Race Black	Sex Female Race Other	Intercept
Coefficient	-4.32	0.84	-2.18	4.48	0.37	132.85

Instead of having the levels of `sex` and `race` in separate cells, we may prefer to place them in a single cell, delimited by an `x`. We make that change below and center the results horizontally:

```
. collect style column, delimiter(" x ")
. collect style cell result, halign(center)
. collect preview
```

	Female	Black	Other	Female x Black	Female x Other	Intercept
Coefficient	-4.32	0.84	-2.18	4.48	0.37	132.85

One last thing we can do to make this table even better would be to set the columns to have equal widths. Currently, the width of each column is determined by its contents. We will not make the change here, because it would make the table wrap. But you can experiment by adding the `width(equal)` option to the `collect style column` command from above.

◁

▷ Example 3: Binders for factor variables and their levels

For some tables, you may want to present the label for the factor variable and its level in a single cell. For example, in example 1 we may have wanted to display `Age group: 20–29`, `Age group: 30–39`, and such. To make this change, you may be tempted to simply type

```
. collect style column, binder(":")
```

However, you will not see the change applied, because dimension titles are hidden with the default style used by `collect`. Factor variables are treated as their own dimension, so you will not see the title for the factor variables. To obtain headers such as `Age group: 20–29`, first we make the collection `default` the current collection. (Our first example was executed in the collection called `default`.) Then, we need to specify that we want to see the title for the age group dimension; specifically, we want to see its label. Then, we can get a preview of the table.

```
. collect set default
. collect style column, binder(":")
. collect style header agegrp, title(label)
. collect preview
```

We do not include the output here, because the resulting table is rather wide. However, you can run these commands to view the resulting table.

◁

Stored results

`collect style column` stores the following in `s()`:

Macros
`s(collection)` name of collection

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

[TABLES] **collect query** — Query collection style properties

[TABLES] **collect style header** — Collection styles for hiding and showing header components

[TABLES] **collect style row** — Collection styles for row headers

collect style _cons — Collection styles for intercept position

Description Menu Syntax Option
Remarks and examples Stored results Reference Also see

Description

collect style _cons specifies the position of the intercept for estimation results included in tables. The intercept may be placed at the end of the list of covariates or at the beginning.

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Build and style table

Syntax

collect style _cons { first | last } [, name(*cname*)]

where *cname* is a collection name.

Option

name(*cname*) specifies a collection *cname* to which the style for the intercept position is applied. By default, the style is applied to the current collection.

Remarks and examples

collect style _cons specifies whether the intercept is displayed at the beginning of the list of covariates or at the end of the list of covariates. This appearance style is applicable when the results of estimation commands are included in the table produced by collect.

For example, we have data from the Second National Health and Nutrition Examination Survey (NHANES II) (McDowell et al. 1981). Below, we fit a simple model with a single independent variable, and we collect the coefficients (_r_b). We use the quietly prefix to suppress the output. Then, we arrange the results with the variable names on the rows and the statistics (result) on the columns:

```
. use https://www.stata-press.com/data/r19/nhanes2
. quietly: collect _r_b: regress bpsystol bmi
. collect layout (colname) (result)
Collection: default
      Rows: colname
   Columns: result
   Table 1: 2 x 1
```

	Coefficient
Body mass index (BMI)	1.656894
Intercept	88.56855

Here we see that by default the intercept is displayed at the end of the list of covariates. Below, we specify that we want it listed first, and then we get a preview of the table:

```
. collect style _cons first
. collect preview
```

	Coefficient
Intercept	88.56855
Body mass index (BMI)	1.656894

Stored results

collect style _cons stores the following in s():

Macros

s(collection) name of collection

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

[TABLES] **collect query** — Query collection style properties

[TABLES] **collect style cell** — Collection styles for cells

Description | Quick start | Menu | Syntax
Options | Remarks and examples | Stored results | Also see

Description

`collect style header` edits the content of the table headers. With this style, you can specify how the dimensions and levels are displayed in row, column, and table headers. For each dimension, the name of the dimension, the label of the dimension, or nothing may be displayed. Likewise, for levels within a dimension, the label of that level, the value of the level, or nothing may be displayed.

Quick start

For all dimensions, display the dimension label and the level labels in row, column, and table headers

    collect style header, title(label) level(label)

For dimension `d1`, hide the dimension title

    collect style header d1, title(hide)

For dimension `d1`, display the level values

    collect style header d1, level(value)

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Build and style table

Syntax

`collect style header` [*dimlist*] [, *options*]

dimlist is a dimension in a collection, a dimension and its corresponding level or list of levels, or a combination of these:

dimension

dimension`[`*level*`]`

dimension`[`*level1 level2* `...]`

options	Description
`name(`*cname*`)`	specify header styles for collection *cname*
`title(`*tstyle*`)`	specify dimension title header style
`level(`*lstyle*`)`	specify dimension level header style
`nformat(`%*fmt*`)`	specify numeric format for numeric levels
`fvlevels(`*fvlstyle*`)`	specify factor-variable level header style
`basestyle`	update basestyle properties

`basestyle` does not appear in the dialog box.

Options

`name(`*cname*`)` specifies the collection to which header style properties are to be applied. By default, properties are applied to the current collection.

`title(`*tstyle*`)` specifies the dimension title header style to be used in row, column, and table headers. *tstyle* may be `label`, `name`, or `hide`.

`label` specifies that `collect` use the dimension's label for headers. If a dimension does not have a label, then `collect` will use the dimension's name.

`name` specifies that `collect` use the dimension's name for headers.

`hide` specifies that `collect` not show the dimension's label or name in the headers.

The default is `title(hide)`.

`level(`*lstyle*`)` specifies the dimension's level header style to be used in row, column, and table headers. *lstyle* may be `label`, `value`, or `hide`.

`label` specifies that `collect` use the level's value labels for headers. If a level does not have a label, then `collect` will use the level's value.

`value` specifies that `collect` use the level's values for headers.

`hide` specifies that `collect` not show the level's labels or values in the headers.

The default is `level(label)`.

`nformat(`*%fmt*`)` applies the Stata numeric format *%fmt* to numeric levels for a given dimension in the header.

`fvlevels(`*fvlstyle*`)` specifies the dimension's factor-variable level header style to be used in row, column, and table headers. *fvlstyle* may be `show` or `hide`.

`show` specifies that `collect` show factor-variable levels in the headers.

`hide` specifies that `collect` not show factor-variable levels in the headers.

The default is `fvlevel(show)`.

`basestyle` indicates that the header style edits be applied to the base header style properties, instead of overriding the current style for the headers.

Each header begins with baseline header style properties. (You can view your table with these baseline properties by first clearing out the collection styles with `collect style clear`.) The appearance of the headers is then updated with any changes specified in the default style used by `collect` and `table`. Any `collect style header` command you issue will override the current style for that header. If you specify the `basestyle` option, the style changes will instead apply to the baseline style and they will not override any current header style edits targeted to specific dimensions.

For example, suppose your current style displays values for levels of dimensions. You then decide to display the labels for the levels of the dimension `result`, by typing `collect style header result, level(label)`. After you preview your table, you choose to hide the values and labels for all other dimension levels. You can type `collect style header, level(hide) basestyle` to make this change while continuing to display the labels for levels of the dimension `result`.

Remarks and examples

`collect style header` specifies the way that dimensions and their levels be displayed in row, column, and table headers. `collect style header` is often used in combination with `collect label dim` and `collect label levels` to get the desired wording in the headers.

To demonstrate, we first collect results using the `collect` prefix and lay out a table using `collect layout`.

```
. use https://www.stata-press.com/data/r19/nhanes2
. quietly: collect _r_b: regress bpsystol bmi
. quietly: collect _r_b: regress bpsystol bmi age
. collect layout (colname) (cmdset#result)
Collection: default
      Rows: colname
   Columns: cmdset#result
   Table 1: 3 x 2

-----------------------------------------------
                      |           1           2
                      | Coefficient Coefficient
----------------------+------------------------
Body mass index (BMI) |    1.656894    1.304128
Age (years)           |                .5883367
Intercept             |    88.56855    69.58451
-----------------------------------------------
```

By default, we do not see names or labels for the dimensions. However, we do see the labels for all levels that are labeled—the variable labels are the labels for the levels of `colname`, and `Coefficient` is the label for the `_r_b` level of the dimension `result`. The levels of the `cmdset` dimension do not have labels, so we see the values of these levels.

Because the coefficient is the only statistic in the table, we could hide its label by specifying the `level(hide)` option for the `result` dimension.

```
. collect style header result, level(hide)
. collect preview

---------------------------------------
                      |        1        2
----------------------+------------------
Body mass index (BMI) | 1.656894 1.304128
Age (years)           |          .5883367
Intercept             | 88.56855 69.58451
---------------------------------------
```

If the levels of `cmdset` had labels, they would show because `level(label)` is the default for all dimensions. Here we add labels to the levels of this dimension, and they automatically appear in the column headers.

```
. collect label levels cmdset 1 "Model 1" 2 "Model 2"
. collect preview

-----------------------------------------
                      |  Model 1  Model 2
----------------------+------------------
Body mass index (BMI) | 1.656894 1.304128
Age (years)           |          .5883367
Intercept             | 88.56855 69.58451
-----------------------------------------
```

Suppose we wanted to see the names of our variables (the values of the levels of the colname dimension) rather than their labels on the rows. We can request this with option level(value). We can also specify a new label for the colname dimension and show this label in the row headers by specifying the title(label) option with collect style header.

```
. collect label dim colname "Covariates", modify
. collect style header colname, level(value) title(label)
. collect preview
```

```
-----------------------------
           |  Model 1  Model 2
-----------+-----------------
Covariates |
  bmi      | 1.656894 1.304128
  age      |          .5883367
  _cons    | 88.56855 69.58451
-----------------------------
```

Let's add estimation results using a factor variable.

```
. quietly: collect _r_b: regress bpsystol bmi age i.female
. collect label values cmdset 3 "Model 3"
. collect preview
```

```
----------------------------------------
           |  Model 1  Model 2   Model 3
-----------+----------------------------
Covariates |
  bmi      | 1.656894 1.304128  1.305773
  age      |          .5883367   .589271
  Male     |                           0
  Female   |                   -4.063319
  _cons    | 88.56855 69.58451  71.63197
----------------------------------------
```

Recall that dimension titles are hidden by default. This applies to factor variables, too, so we see the value labels of the female variable but not its name or label. Because female is binary, we can hide the base level and show its name instead of the value labels. We use collect style showbase off to hide the base level. We then use collect style header to show female's variable name in the header's title and to hide the factor-variable levels in the header for dimension colname.

```
. collect style row stack, nobinder
. collect style showbase off
. collect style header female, title(name)
. collect style header colname, fvlevels(hide)
. collect preview
```

```
----------------------------------------
           |  Model 1  Model 2   Model 3
-----------+----------------------------
Covariates |
  bmi      | 1.656894 1.304128  1.305773
  age      |          .5883367   .589271
  female   |                   -4.063319
  _cons    | 88.56855 69.58451  71.63197
----------------------------------------
```

The fvlevels() style property works on interactions, too, and is used in the anova predefined style; see [TABLES] **Predefined styles** and [TABLES] **Example 8**.

Here we add estimation results using an interaction and preview the table using the same header styles as in the previous table.

```
. quietly: collect _r_b: regress bpsystol bmi c.age##i.female
. collect label values cmdset 4 "Model 4"
. collect preview
```

	Model 1	Model 2	Model 3	Model 4
Covariates				
bmi	1.656894	1.304128	1.305773	1.264658
age		.5883367	.589271	.4390171
female			-4.063319	-17.78472
female # age				.288516
_cons	88.56855	69.58451	71.63197	79.8064

In the examples above, we have modified our header styles for a selected dimension. However, `collect style header` is not limited to modifying only one dimension. If we wish to make a change for all dimensions, we can simply omit the dimension names from the command. For instance, we could type

```
. collect style header, title(label)
```

Alternatively, we could specify a header style for multiple dimensions. For instance, we could type

```
. collect style header cmdset colname, title(label)
```

If you have a preferred method of displaying the dimensions and their levels for many of the tables you create, you can use `collect style save` to save a file with this style along with any others you like. Then, with future collections, you can use `collect style use` to apply this header style to future collections and tables.

Stored results

`collect style header` stores the following in `s()`:

Macros
`s(collection)` name of collection

Also see

[TABLES] **collect label** — Manage custom labels in a collection

[TABLES] **collect query** — Query collection style properties

[TABLES] **collect style save** — Save collection styles to disk

[TABLES] **collect style use** — Use collection styles from disk

Description | Quick start | Menu | Syntax
Options | Remarks and examples | Stored results | Also see

Description

`collect style html` specifies styles to be used when exporting a table from a collection to an HTML file.

`collect style html`, typed without any options, will clear the existing HTML appearance styles for the current collection.

Quick start

Specify that tables exported from the current collection to a HyperText Markup Language (HTML) file use tag `<th>` for header cells.

```
collect style html, useth
```

Clear the current HTML appearance styles

```
collect style html
```

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Collect styles > Styles for HTML

Syntax

Specify styles to be used when exporting a collection to an HTML *file*

`collect style html` [, *options*]

Clear existing HTML *appearance styles*

`collect style html` [, `name(`*cname*`)`]

options	Description
`name(`*cname*`)`	apply HTML styles to collection *cname*
[`no`]`bcollapse`	collapse adjacent cell borders
[`no`]`useth`	use HTML tag `<th>` for header cells

Options

`name(`*cname*`)` specifies that the HTML styles be applied to collection *cname*.

When `name(`*cname*`)` is specified without any other options, HTML styles are cleared from collection *cname*.

The default in both cases is to apply the style changes to the current collection.

`bcollapse` and `nobcollapse` control whether adjacent cell borders are collapsed into a single border.

`bcollapse`, the default, specifies that `collect export` collapse adjacent cell borders into a single border.

`nobcollapse` specifies that `collect export` not collapse adjacent cell borders into a single border.

`useth` and `nouseth` control which HTML tag to use for header cells.

`useth` specifies that `collect export` use the HTML tag `<th>` for header cells.

`nouseth`, the default, specifies that `collect export` use the HTML tag `<td>` rather than `<th>` for header cells.

Remarks and examples

`collect style html` allows you to specify styles for the table that you will export to an HTML file with `collect export`. If you do not like the change you have made, you can clear the HTML appearance styles by typing the following:

```
. collect style html
```

This change will be applied to the current collection. To make this change for another collection, specify the collection name with the `name()` option.

Stored results

`collect style html` stores the following in `s()`:

Macros

`s(collection)`	name of collection

Also see

[TABLES] **collect export** — Export table from a collection

[TABLES] **collect query** — Query collection style properties

collect style notes — Collection styles for table notes

Description

`collect style notes` specifies appearance styles for table notes in the collection. This includes bolding, italics, font, text color, and shading. Certain appearance edits can be rendered only on certain export formats.

Quick start

Make the table notes italic

```
collect style notes, font(, italic)
```

When exporting the table to a LaTeX file, use the LaTeX macro `textbf` to render the notes

```
collect style notes, latex(textbf)
```

Clear the table notes styles

```
collect style notes, clear
```

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Build and style table

Syntax

`collect style notes` [, *options*]

options	Description
Main	
`name(`*cname*`)`	apply appearance styles to collection *cname*
`clear`	remove table note styles
Fonts	
`font(`[*fontfamily*] [, *font_opts*]`)`	set font style for table notes
`smcl(`*smcl*`)`	specify formatting for SMCL files
`latex(`*latex*`)`	specify LaTeX macro
Shading	
`shading(`*sspec*`)`	set background color, foreground color, and fill pattern for notes

font_opts	Description
`size(`# [*unit*]`)`	specify font size
`color(`*color*`)`	specify font color
`variant(`*variant*`)`	specify font variant and capitalization
[`no`]`bold`	specify whether to format notes as bold
[`no`]`italic`	specify whether to format notes as italic
[`no`]`strikeout`	specify whether to strike out notes
[`no`]`underline`	specify whether to underline notes

fontfamily specifies a font family.

sspec is

[`background(`*bgcolor*`)` `foreground(`*fgcolor*`)` `pattern(`*fpattern*`)`]

bgcolor specifies the background color.

fgcolor specifies the foreground color.

fpattern specifies the fill pattern. A complete list of fill patterns is shown in *Shading patterns* of [TABLES] **Appendix**.

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

unit may be `in` (inch), `pt` (point), or `cm` (centimeter). An inch is equivalent to 72 points and 2.54 centimeters. The default is `pt`.

Options

Main

`name(`*cname*`)` specifies the collection to which note appearance styles are to be applied. By default, the table note styles are applied to the current collection.

`clear` removes existing table note style properties.

Fonts

`font(`[*fontfamily*] [`, size(`# [*unit*]`) color(`*color*`) variant(`*variant*`)` [`no`]`bold` [`no`]`italic` [`no`]`strikeout` [`no`]`underline`]`)` specifies the font style for the table notes.

These font style properties are applicable when publishing items from a collection to Microsoft Word, Microsoft Excel, PDF, HTML, and LaTeX files.

fontfamily specifies a font family.

`size(`# [*unit*]`)` specifies the font size as a number optionally followed by units. If # is specified without the optional *unit*, points is assumed.

`color(`*color*`)` specifies the font color.

`variant(`*variant*`)` specifies the font variant and capitalization. *variant* may be `allcaps`, `smallcaps`, or `normal`. `variant(allcaps)` changes the text to all uppercase letters. `variant(smallcaps)` changes the text to use large capitals for uppercase letters and smaller capitals for lowercase letters. `variant(normal)` changes the font variant back to normal; capitalization is unchanged from the original text.

This style property is applicable only when publishing to HTML and LaTeX files.

`bold` and `nobold` specify the font weight. `bold` changes the font weight to bold; `nobold` changes the font weight back to normal.

`italic` and `noitalic` specify the font style. `italic` changes the font style to italic; `noitalic` changes the font style back to normal.

`strikeout` and `nostrikeout` specify whether to add a strikeout mark to the notes. `strikeout` adds a strikeout mark to the notes; `nostrikeout` changes the notes back to normal.

Only one of `strikeout` or `underline` is allowed when publishing to HTML files.

`underline` and `nounderline` specify whether to underline the notes. `underline` adds a single line under each note; `nounderline` removes the underline.

Only one of `strikeout` or `underline` is allowed when publishing to HTML files.

`smcl(`*smcl*`)` specifies how to render notes for SMCL output. The supported SMCL directives are `input`, `error`, `result`, and `text`.

This style property is applicable only when publishing to a SMCL file.

`latex(`*latex*`)` specifies the name of a LaTeX macro to render notes for LaTeX output. This style property is applicable only when publishing to a LaTeX file.

Example LaTeX macro names are `textbf`, `textsf`, `textrm`, and `texttt`. Custom LaTeX macros are also allowed. If *note* is the table note, then *latex* is translated to the following when exporting to LaTeX:

latex{*note*}

Shading

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

These shading style properties are applicable when publishing to Microsoft Excel, Microsoft Word, PDF, and HTML.

Remarks and examples

collect style notes allows you to customize the appearance of the notes added with collect notes and collect stars. For example, if you export your table to a Microsoft Excel, a Microsoft Word, a PDF, an HTML, or a LaTeX file, you can customize the font for the notes. Additionally, you can use SMCL directives and LaTeX macros to specify how to render your table notes.

To list the current appearance styles for table notes, you can use collect query notes.

▷ Example 1: Customizing the font for the table note

Suppose that we want to export a table of regression results to an HTML file. Below, we load data from the Second National Health and Nutrition Examination Survey (NHANES II) (McDowell et al. 1981). We fit a model for systolic blood pressure as a function of age group and include a note with the data source:

```
. use https://www.stata-press.com/data/r19/nhanes2l
(Second National Health and Nutrition Examination Survey)

. quietly: collect _r_b: regress bpsystol i.agegrp

. collect notes "Data source: NHANES II"
```

Then, we lay out our table with collect layout:

```
. collect layout (colname) (result)

Collection: default
      Rows: colname
   Columns: result
   Table 1: 7 x 1
```

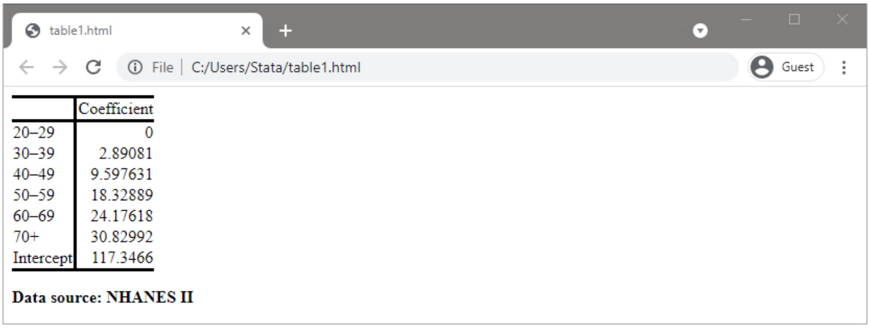
	Coefficient
20-29	0
30-39	2.89081
40-49	9.597631
50-59	18.32889
60-69	24.17618
70+	30.82992
Intercept	117.3466

Data source: NHANES II

Here we see that our note is added in plain text. We wish to make our note bold, so we use the font(, bold) option and then export our table to a file called table1.html.

```
. collect style notes, font(, bold)

. collect export table1.html
(collection default exported to file table1.html)
```

When we open the file, we see the following:



	Coefficient
20–29	0
30–39	2.89081
40–49	9.597631
50–59	18.32889
60–69	24.17618
70+	30.82992
Intercept	117.3466

Data source: NHANES II

If we had included a note with `collect stars`, that note would have been displayed with a bold font as well.

◁

Stored results

`collect style notes` stores the following in `s()`:

Macros
    `s(collection)`   name of collection

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

[TABLES] **collect notes** — Add table notes in a collection

[TABLES] **collect query** — Query collection style properties

[TABLES] **collect stars** — Add stars for significant results in a collection

[TABLES] **collect title** — Add a custom table title in a collection

Description Quick start Menu Syntax
Options Remarks and examples Stored results Reference
Also see

Description

`collect style putdocx` specifies styles to be used by `putdocx` when exporting a table from a collection with `putdocx collect`.

`collect style putdocx`, typed without any options, will clear the current styles for the current collection.

Quick start

Specify that tables exported with `putdocx collect` have a width 80% of the default and be right-aligned

```
collect style putdocx, width(80%) halign(right)
```

Same as above, but for the collection `c2`

```
collect style putdocx, width(80%) halign(right) name(c2)
```

Clear the current styles for `putdocx collect`

```
collect style putdocx
```

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Collect styles > Styles for putdocx

Syntax

Specify styles to be used when exporting a collection with `putdocx collect`

`collect style putdocx` [, *options*]

Clear the current styles for `putdocx collect`

`collect style putdocx` [, `name(`*cname*`)`]

cname is the name of an existing collection.

options	Description
Main	
`name(`*cname*`)`	apply styles to collection *cname*
`halign(`*hvalue*`)`	set table horizontal alignment
`width(#`[*unit* | `%`] | *matname*`)`	set table width; option may be repeated
`layout(`*layouttype*`)`	adjust column width
`indent(#`[*unit*]`)`	set table indentation
`cellspacing(#`[*unit*]`)`	set spacing between adjacent cells and the edges of the table
`headerrow(#)`	set number of the top rows that constitute the table header
Cell margins	
`cellmargin(`*cmarg*`, #`[*unit*]`)`	set margins for each table cell; option may be repeated

unit may be `in` (inch), `pt` (point), `cm` (centimeter), or `twip` (20th of a point). An inch is equivalent to 72 points, 2.54 centimeters, or 1440 twips. The default is `in`.

Options

Main

`name(`*cname*`)` specifies that styles for `putdocx collect` be applied to collection *cname*.

When `name(`*cname*`)` is specified without any other options, styles for `putdocx collect` are cleared from collection *cname*.

The default in both cases is to apply the style changes to the current collection.

`halign(`*hvalue*`)` sets the horizontal alignment of the table within the page. *hvalue* may be `left`, `right`, or `center`. The default is `halign(left)`.

`width(#`[*unit* | `%`]`)` and `width(`*matname*`)` set the table width. Any two of the types of width specifications can be combined.

`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. When specifying the table width as a percentage, it cannot be greater than 100%. For example, `width(50%)` sets the table width to 50% of the default table width. The default is `width(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 percent 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.

`width()` may be specified multiple times in a single command to control the table width and individual column widths simultaneously.

`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)`.

`indent(#[` *unit* `])` specifies the table indentation from the left margin of the current document.

`cellspacing(#[` *unit* `])` sets the spacing between adjacent cells and the edges of the table.

`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.

Cell margins

`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.

Remarks and examples

After finalizing your table of results, you can export it to a Word document (`.docx` file) in two ways. One is to simply use `collect export`—this command will create a document with a table from a collection. The other method is to incorporate the table into a larger report created by `putdocx`. In this case, you create an active `.docx` file, to which you can add a table from a collection along with formatted text, graphs, and other results created in Stata.

The second method utilizing `putdocx` allows you to take advantage of additional formatting features for the table you are exporting to the `.docx` file. Before you export your table, you use `collect style putdocx` to apply your desired styles to your collection. For example, you can specify the option `layout(autofitcontents)` so that the width of the columns of the table will automatically be resized to fit the contents. Then, as you are creating your report with `putdocx`, you can export the customized table to your document with `putdocx collect`.

Stored results

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

Macros
`s(collection)` name of collection

Reference

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/.

Also see

[TABLES] **collect export** — Export table from a collection

[TABLES] **collect query** — Query collection style properties

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

collect style putpdf — Collection styles for putpdf

Description

`collect style putpdf` specifies the styles to be used by `putpdf` when exporting a table from a collection with `putpdf collect`.

`collect style putpdf`, typed without any options, will clear the current styles for the current collection.

Quick start

Specify that tables exported with `putpdf collect` be right-aligned

```
collect style putpdf, halign(right)
```

Same as above, but for the collection `c2`

```
collect style putpdf, halign(right) name(c2)
```

Clear the current styles for `putpdf collect`

```
collect style putpdf
```

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Collect styles > Styles for putpdf

Syntax

Specify styles to be used when exporting a collection with `putpdf collect`

`collect style putpdf` [, `name(`*cname*`)` *options*]

Clear the current styles for `putpdf collect`

`collect style putpdf` [, `name(`*cname*`)`]

cname is the name of an existing collection.

options	Description
Main	
`name(`*cname*`)`	apply styles to collection *cname*
`width(#`[*unit* | `%`] | *matname*`)`	set table width; option may be repeated
`indent(#`[*unit*]`)`	set table indentation
`halign(`*hvalue*`)`	set table horizontal alignment
Spacing	
`spacing(`*position*`, #`[*unit*]`)`	set spacing before or after table; option may be repeated

unit may be `in` (inch), `pt` (point), `cm` (centimeter), or `twip` (20th of a point). An inch is equivalent to 72 points, 2.54 centimeters, or 1440 twips. The default is `in`.

Options

Main

`name(`*cname*`)` specifies that styles for `putpdf collect` be applied to collection *cname*.

When `name(`*cname*`)` is specified without any other options, styles for `putpdf collect` are cleared from collection *cname*.

The default in both cases is to apply the style changes to the current collection.

`width(#`[*unit* | `%`]`)` and `width(`*matname*`)` set the table width. Any two of the types of width specifications can be combined.

`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. When specifying the table width as a percentage, it cannot be greater than 100%. For example, `width(50%)` sets the table width to 50% of the default table width. The default is `width(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 percent 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.

`width()` may be specified multiple times in a single command to control the table width and individual column widths simultaneously.

`indent(#`[*unit*]`)` specifies the table indentation from the left margin of the current document.

`halign(`*hvalue*`)` sets the horizontal alignment of the table within the page. *hvalue* may be `left`, `right`, or `center`. The default is `halign(left)`.

Spacing

`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.

Remarks and examples

After finalizing your table of results, you can export it to a PDF file in two ways. One is to simply use `collect export`—this command will create a document with a table with items from a collection. The other method is to incorporate the table into a larger report created by `putpdf`. In this case, you create an active `.pdf` file, to which you can add a table from a collection along with formatted text, graphs, and other results created in Stata.

The second method allows you to take advantage of additional formatting features for the table you are exporting to the PDF file. Before you export your table, you use `collect style putpdf` to apply your desired styles to your collection. For example, you can specify the table indentation to the table. Then, as you are creating your report with `putpdf`, you can export the customized table to your document with `putpdf collect`.

If you do not like the appearance of the table, you can clear out the collection styles for `putpdf` by typing

```
. collect style putpdf
```

This will clear out the collection styles for the current collection. If you want to make this change with another collection that you have in memory, specify the collection name with the `name()` option.

Stored results

`collect style putpdf` stores the following in `s()`:

Macros
`s(collection)`   name of collection

Reference

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/.

Also see

[TABLES] **collect export** — Export table from a collection

[TABLES] **collect query** — Query collection style properties

[RPT] **putpdf collect** — Add a table from a collection to a PDF file

collect style row — Collection styles for row headers

Description

collect style row specifies row header style properties. collect style row determines how row headers are constructed, how factor variables are displayed, how duplicates are reported, and how labels wrap or truncate.

Quick start

Stack row header elements in a single column

```
collect style row stack
```

Same as above, and use a colon to separate factor variables from their levels

```
collect style row stack, binder(" : ")
```

Place row header elements in separate columns

```
collect style row split
```

Same as above, and use an x to delimit interaction terms

```
collect style row split, delimiter(" x ")
```

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Build and style table

Syntax

Split row header elements across columns

`collect style row split` [, *options split_options*]

Stack row header elements in a single column

`collect style row stack` [, *options stack_options*]

options	Description
`name(`*cname*`)`	specify row header styles for collection *cname*
`nodelimiter`	place factor-variable and interaction elements in separate cells without a delimiter
`delimiter(`*delim*`)`	use *delim* to delimit interaction terms composed in a single cell
`atdelimiter(`*atdelim*`)`	use *atdelim* to delimit interaction terms containing the `@` symbol
`bardelimiter(`*bardelim*`)`	use *bardelim* to delimit interaction terms containing the `|` symbol
`binder(`*binder*`)`	use *binder* to separate factor variables from their levels
`nobinder`	do not bind factor variables and their levels
[`no`]`spacer`	add a blank line between stacked row dimensions

`nobinder` is only allowed with `collect style row stack`.

split_options	Description
`dups(`*dups*`)`	specify how duplicate headers are displayed
`position(`*rowpos*`)`	specify the position of the row header to be filled first
[`no`]`span`	span row headers into empty row header columns

stack_options	Description
[`no`]`indent`	indent stacked headers
`length(`#`)`	specify maximum length for stacked headers
`wrapon(`*wrapon*`)`	specify how to break long headers
`wrap(`#`)`	specify number of lines to allow for long headers
`truncate(`*truncate*`)`	specify how to truncate headers that do not fit
[`no`]`abbreviate`	abbreviate long words that do not fit within the specified length

Options

Main

`name(`*cname*`)` specifies the collection to which column header style properties are to be applied. By default, properties are applied to the current collection.

`nodelimiter`, `delimiter()`, `atdelimiter()`, and `bardelimiter()` control how to compose factor-variable and interaction terms in headers.

`nodelimiter` specifies that factor-variable and interaction term elements (matrix stripe elements) be split into separate cells.

`delimiter(`*delim*`)` specifies that factor-variable and interaction term elements (matrix stripe elements) be composed in a single cell.

The variables in an interaction term are composed in a single cell using *delim* as the delimiter.

Factor-variable terms serve as their own dimension nested within the stripe dimensions `coleq`, `colname`, `roweq`, and `rowname`. Option `binder()` controls how levels of factor variables are composed within a single cell.

`atdelimiter(`*atdelim*`)` specifies that *atdelim* be used to delimit interaction terms containing the `@` symbol. This option is applicable when, for example, working with results from `contrast`, `mean`, `proportion`, `ratio`, and `total`.

`bardelimiter(`*bardelim*`)` specifies that *bardelim* be used to delimit interaction terms containing the `|` symbol. This option is applicable when, for example, working with results from `anova` and `manova`.

`binder(`*binder*`)` specifies how to compose levels of factor variables within a single cell.

The binder will be applied as long as the factor variable and its levels are not hidden. Note that the default style used by `collect`, which is `style-default.stjson`, will hide the dimension title from the headers. You can use `collect style header` to specify whether to display the label or name for a dimension and whether to display the label or value for the level of a dimension.

`nobinder` specifies that factor variables should not be bound to their levels. By default, when stacking row headers, factor variables are bound to their levels by an equal sign.

This option is only allowed with `collect style row stack`.

`nospacer` and `spacer` control whether a blank line is added between stacked row dimensions.

`nospacer`, the default, prevents the line from being added.

`spacer` adds the line.

Split options

`dups(`*dups*`)` controls how to handle duplicate header elements. *dups* is one of `repeat`, `first`, or `center`.

`dups(repeat)`, the default, specifies that `collect` repeat duplicate header elements.

`dups(first)` specifies that `collect` hide all duplicate header elements, except the first.

`dups(center)` specifies that `collect` horizontally center duplicate header elements, where the header element spans the duplicate header cell locations. When this style is not supported, such as when exporting to Markdown, `dups(first)` is used instead.

`position(`*rowpos*`)` specifies how split headers are filled in when one or more levels of a dimension occupy more than one cell. This option is used when factor variables are displayed in the row headers. *rowpos* may be `left` or `right`.

`position(left)` is the default and specifies that `collect` fill in row headers starting with the leftmost cell. This will result in some empty cells on the right for unbalanced row dimensions.

`position(right)` specifies that `collect` shift the row header cells to the right so that the cells in the last column are all filled in. This will result in some empty cells on the left for unbalanced row dimensions.

nospan and span control whether row headers span into empty row header columns. This option is effective only when position(left) is in effect.

span, the default, specifies that row headers should span into empty row header columns. This helps conserve horizontal space. Otherwise, each column of the row header will be forced to be wide enough to accommodate all the cells.

nospan specifies that row headers should not span into empty row header columns.

Stack options

noindent and indent control indenting of stacked headers.

indent, the default, turns on indenting.

noindent turns off indenting.

length(#) specifies the maximum display length for stacked headers.

Long headers, ones that contain more than # display characters, are broken into multiple rows. Values of # less than 5 are ignored.

If header elements are indented, each indent counts as 2 characters. If # is too small to fit the indented headers, # is increased to accommodate the most indented header. For example, if there is one level of indented headers, and length(5) was specified, then # is increased to 7.

By default, there is no limit to the header length.

wrapon(*wrapon*) specifies how to break long headers. *wrapon* may be word or length.

wrapon(word), the default, specifies that long headers break at word boundaries.

wrapon(length) specifies that headers break based on available space.

wrap(#) specifies how many lines to allow when long headers are broken into multiple lines. Headers requiring more than # lines are truncated with ellipses. Values of # less than 1 are ignored.

By default, there is no limit to the number of lines for wrapped headers.

truncate(*truncate*) specifies how to truncate headers that do not fit within the specified number of lines to wrap. *truncate* may be tail, middle, or head.

truncate(tail), the default, specifies that long headers are truncated at the end.

truncate(middle) specifies that long headers are truncated in the middle.

truncate(head) specifies that long headers are truncated at the beginning.

noabbreviate and abbreviate control whether long words are abbreviated when wrapon(word) is in effect.

noabbreviate, the default, specifies that words that do not fit in the specified length should not be abbreviated.

abbreviate specifies that long words be abbreviated if they do not fit in the specified length.

Remarks and examples

collect style row determines how row headers are constructed, how factor variables are displayed, how duplicates are reported, and how labels wrap or truncate. In the examples that follow, we explore how factor-variable and interaction terms are incorporated in row headers.

▷ Example 1: Working with factor variables

Below, we use data from the Second National Health and Nutrition Examination Survey (NHANES II) (McDowell et al. 1981). We begin by fitting a model for systolic blood pressure as a function of agegrp and sex. We collect the results, requesting that coefficients (_r_b) appear in subsequent tables, and use the quietly prefix to suppress the output. Then, we arrange the items in our collection with collect layout. We place the variable names on the rows and the statistics (result) on the columns:

```
. use https://www.stata-press.com/data/r19/nhanes2
. quietly: collect _r_b: regress bpsystol i.agegrp i.sex
. collect layout (colname) (result)
Collection: default
      Rows: colname
   Columns: result
   Table 1: 9 x 1

---------------------------
          |  Coefficient
----------+----------------
20-29     |            0
30-39     |     2.916153
40-49     |     9.603552
50-59     |     18.38803
60-69     |     24.18566
70+       |     30.93702
Male      |            0
Female    |    -4.015163
Intercept |     119.4303
---------------------------
```

collect's default style omits the dimension titles, and factor variables get treated as dimensions as well. This is why we see the labels for the levels of the factor variables but not the names of the factor variables.

Below, we specify that we want to see the title for agegrp. We also specify that we want to split the row headers across columns. Then, we get a preview of our table:

```
. collect style header agegrp, title(label)
. collect style row split
. collect preview
```

```
                          Coefficient

Age group 20-29                     0
Age group 30-39              2.916153
Age group 40-49              9.603552
Age group 50-59              18.38803
Age group 60-69              24.18566
Age group 70+                30.93702
Sex       Male                      0
Sex       Female            -4.015163
Intercept                    119.4303
```

By splitting, we have created two columns within our row header, one for variable names (or their labels) and one for the levels of the factor variables.

We do not need to see Age group and Sex repeated on every row, so we can add the dups(first) option to indicate that duplicates should be displayed only the first time they appear.

```
. collect style row split, dups(first)
. collect preview
```

```
                          Coefficient

Age group 20-29                     0
          30-39              2.916153
          40-49              9.603552
          50-59              18.38803
          60-69              24.18566
          70+                30.93702
Sex       Male                      0
          Female            -4.015163
Intercept                    119.4303
```

Now, let's stack all the elements of the row headers into a single column.

```
. collect style row stack
. collect preview
```

```
                         Coefficient

Age group=20-29                    0
Age group=30-39             2.916153
Age group=40-49             9.603552
Age group=50-59             18.38803
Age group=60-69             24.18566
Age group=70+               30.93702
Male                               0
Female                     -4.015163
Intercept                   119.4303
```

By default, when we stack row headers, the titles for the factor variables are bound to their levels by an equal sign, and each bound term is placed in a single cell in the row header. With this binding, we cannot see an effect of stacking the row headers for this simple table.

Continuing with the binders for now, we can specify the `binder()` option to bind the factor variables and their levels using other characters. Here we replace the equal sign with a colon.

```
. collect style row stack, binder(":")
. collect preview
```

	Coefficient
Age group:20-29	0
Age group:30-39	2.916153
Age group:40-49	9.603552
Age group:50-59	18.38803
Age group:60-69	24.18566
Age group:70+	30.93702
Male	0
Female	-4.015163
Intercept	119.4303

This may look better if we stack the levels of factor variables underneath their titles. We can obtain this layout by removing the binder with `nobinder`.

```
. collect style row stack, nobinder
. collect preview
```

	Coefficient
Age group	
20-29	0
30-39	2.916153
40-49	9.603552
50-59	18.38803
60-69	24.18566
70+	30.93702
Sex	
Male	0
Female	-4.015163
Intercept	119.4303

Here we demonstrated the stacked and split row header arrangements using factor variables, but these layouts can also be used to control the look of row headers with other dimensions and when you have multiple row dimensions.

◁

▷ Example 2: Working with interactions

When working with models with interactions, you may also want to specify the delimiter. For example, below we create a new collection called `interaction`, which then becomes the current collection. Then, we fit a model with an interaction between `race` and `sex`, requesting that only the coefficients appear in our tables. We specify the same layout as we did in the previous example:

```
. collect create interaction
(current collection is interaction)

. quietly: collect _r_b: regress bpsystol race##sex

. collect layout (colname) (result)

Collection: interaction
      Rows: colname
   Columns: result
   Table 1: 12 x 1
```

	Coefficient
White	0
Black	.8423655
Other	-2.177732
Male	0
Female	-4.32123
White # Male	0
White # Female	0
Black # Male	0
Black # Female	4.479353
Other # Male	0
Other # Female	.3729767
Intercept	132.8476

Note that by default a # is used to delimit interaction terms. Below, we specify that we want to use an x:

```
. collect style row split, delimiter(" x ")

. collect preview
```

	Coefficient
White	0
Black	.8423655
Other	-2.177732
Male	0
Female	-4.32123
White x Male	0
White x Female	0
Black x Male	0
Black x Female	4.479353
Other x Male	0
Other x Female	.3729767
Intercept	132.8476

◁

Stored results

`collect style row` stores the following in `s()`:

Macros
    `s(collection)`   name of collection

References

Huber, C. 2021. Customizable tables in Stata 17, part 3: The classic table 1. *The Stata Blog: Not Elsewhere Classified*. https://blog.stata.com/2021/06/24/customizable-tables-in-stata-17-part-3-the-classic-table-1/.

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

[TABLES] **collect query** — Query collection style properties

[TABLES] **collect style header** — Collection styles for hiding and showing header components

[TABLES] **collect style column** — Collection styles for column headers

collect style save — Save collection styles to disk

Description Quick start Menu Syntax
Options Remarks and examples Stored results Also see

Description

`collect style save` saves the current collection's style and layout to a file.

Quick start

Save current collection's style and layout to the file `mystyle.stjson`

```
collect style save mystyle
```

Same as above, but replace `mystyle.stjson` if it exists

```
collect style save mystyle, replace
```

Save style and layout from collection `c1` rather than from the current collection

```
collect style save mystyle, name(c1) replace
```

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Collect styles > Save styles

Syntax

`collect style save` *filename* [`, replace name(`*cname*`)`]

where *cname* is a collection name.

If *filename* is specified without an extension, `.stjson` is assumed. If *filename* contains embedded spaces, enclose it in double quotes.

Options

`replace` specifies that *filename* be replaced if it already exists.

`name(`*cname*`)` specifies the collection from which the style and layout are to be saved. By default, the style and layout from the current collection are saved.

Remarks and examples

`collect style save` saves a collection's style and layout to a file. When you find yourself typing the same list of `collect style` subcommands repeatedly as you build your tables, saving a collection style that includes all your commonly used style settings is very useful. For example, you might add borders, specify the numeric format for results, modify the table headers, and specify the position of the intercept for your current collection. If you plan to create similar tables in the future, you can simply save this style and layout, and apply it to other collections in the future with `collect style use`.

By default, collect uses the styles defined in style-default.stjson when creating tables. However, the default styles can be changed. After saving your preferred style by using the collect style save command, you can use set collect_style to use your style as the default.

Stored results

collect style save stores the following in s():

Macros

s(collection)	name of collection
s(filename)	name of the new file

Also see

[TABLES] **collect style use** — Use collection styles from disk

collect style showbase — Collection styles for displaying base levels

Description | Menu | Syntax | Option
Remarks and examples | Stored results | References | Also see

Description

`collect style showbase` controls the visibility of coefficients and related statistics for base levels of factor variables and interactions in estimation results. When results corresponding to factor variables are included in your collection, you may specify that base levels are never shown, that base levels are shown in the main effects but not interactions, or that base levels are shown in both the main effects and interactions in your tables.

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Build and style table

Syntax

`collect style showbase` { `off` | `factor` | `all` } [, `name(`*cname*`)`]

where *cname* is a collection name.

Option

`name(`*cname*`)` specifies a collection *cname* to which the style is applied. By default, the style is applied to the current collection.

Remarks and examples

`collect style showbase` controls whether coefficients and related statistics are shown for base levels of factor variables and interactions. By default, all base levels are shown in tables. When `factor` is selected, base levels for the main effects of factor variables are shown, but base levels for interaction terms are not shown. When `off` is selected, no base levels are shown in the table.

To be more specific, this setting applies to the following results: `_r_b`, `_r_se`, `_r_z`, `_r_z_abs`, `_r_p`, `_r_lb`, `_r_ub`, `_r_ci`, `_r_df`, `_r_cri`, `_r_crlb`, and `_r_crub`. These are simply the names that are assigned by default to the coefficients, standard errors, test statistics, upper and lower confidence bounds, degrees of freedom, and upper and lower critical interval bounds that are collected with either `collect get` or the `collect` prefix.

To demonstrate, we use data from the Second National Health and Nutrition Examination Survey (NHANES II) (McDowell et al. 1981). Below, we fit a model for systolic blood pressure as a function of age group, `sex`, and their interaction. We use the `collect` prefix to collect the coefficients (`_r_b`) and standard errors (`_r_se`), and we specify the `quietly` prefix to suppress the output.

```
. use https://www.stata-press.com/data/r19/nhanes2
. quietly: collect _r_b _r_se : regress bpsystol sex##agegrp
```

Then, we arrange the items in our collection with `collect layout`. We place the variable names on the rows and the statistics (`result`) on the columns:

```
. collect layout (colname) (result)
Collection: default
      Rows: colname
   Columns: result
   Table 1: 21 x 2
```

	Coefficient	Std. error
Male	0	0
Female	-12.60132	.8402299
20–29	0	0
30–39	.7956175	.9473117
40–49	5.117078	1.018176
50–59	12.20018	1.022541
60–69	16.85887	.8155092
70+	22.50889	1.130959
Male # 20–29	0	0
Male # 30–39	0	0
Male # 40–49	0	0
Male # 50–59	0	0
Male # 60–69	0	0
Male # 70+	0	0
Female # 20–29	0	0
Female # 30–39	4.140156	1.31031
Female # 40–49	8.644866	1.412067
Female # 50–59	11.83134	1.406641
Female # 60–69	14.093	1.130882
Female # 70+	15.86608	1.542296
Intercept	123.8862	.6052954

By default, you see the base level for `sex`, the base level for `agegrp`, and the base levels for the interaction as well. Suppose that we want to display only the base level for each factor variable but not for the interaction. We make this change and then get a preview of the table:

```
. collect style showbase factor
. collect preview
```

	Coefficient	Std. error
Male	0	0
Female	-12.60132	.8402299
20–29	0	0
30–39	.7956175	.9473117
40–49	5.117078	1.018176
50–59	12.20018	1.022541
60–69	16.85887	.8155092
70+	22.50889	1.130959
Female # 30–39	4.140156	1.31031
Female # 40–49	8.644866	1.412067
Female # 50–59	11.83134	1.406641
Female # 60–69	14.093	1.130882
Female # 70+	15.86608	1.542296
Intercept	123.8862	.6052954

Sometimes, we might not want to display any of the base levels. Below, we suppress them all and then preview our table once more:

```
. collect style showbase off
. collect preview
```

	Coefficient	Std. error
Female	-12.60132	.8402299
30-39	.7956175	.9473117
40-49	5.117078	1.018176
50-59	12.20018	1.022541
60-69	16.85887	.8155092
70+	22.50889	1.130959
Female # 30-39	4.140156	1.31031
Female # 40-49	8.644866	1.412067
Female # 50-59	11.83134	1.406641
Female # 60-69	14.093	1.130882
Female # 70+	15.86608	1.542296
Intercept	123.8862	.6052954

Stored results

`collect style showbase` stores the following in `s()`:

Macros
    `s(collection)`   name of collection

References

Huber, C. 2021. Customizable tables in Stata 17, part 5: Tables for one regression model. *The Stata Blog: Not Elsewhere Classified*. https://blog.stata.com/2021/08/26/customizable-tables-in-stata-17-part-5-tables-for-one-regression-model/.

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

[TABLES] **collect query** — Query collection style properties

[TABLES] **collect style showempty** — Collection styles for displaying empty cells

[TABLES] **collect style showomit** — Collection styles for displaying omitted coefficients

collect style showempty — Collection styles for displaying empty cells

Description | Menu | Syntax | Option
Remarks and examples | Stored results | Reference | Also see

Description

`collect style showempty` controls the visibility of coefficients and related statistics in empty cells of factor-variable interactions in estimation results. When results corresponding to empty cells are included in a collection, you can specify whether these should be shown or omitted from your tables.

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Build and style table

Syntax

`collect style showempty` { `on` | `off` } [, `name(`*cname*`)`]

where *cname* is a collection name.

Option

`name(`*cname*`)` specifies a collection *cname* to which the style is applied. By default, the style is applied to the current collection.

Remarks and examples

`collect style showempty` controls whether coefficients and related statistics are shown for empty cells of factor-variable interactions. By default, empty cells are shown in tables.

More specifically, this setting applies to the following results: `_r_b`, `_r_se`, `_r_z`, `_r_z_abs`, `_r_p`, `_r_lb`, `_r_ub`, `_r_ci`, `_r_df`, `_r_cri`, `_r_crlb`, and `_r_crub`. These are simply the names that are assigned by default to the coefficients, standard errors, test statistics, upper and lower confidence bounds, degrees of freedom, and upper and lower critical interval bounds that are collected with either `collect get` or the `collect` prefix.

To demonstrate, we use data from the Second National Health and Nutrition Examination Survey (NHANES II) (McDowell et al. 1981). We wish to fit a model for systolic blood pressure as a function of age group and `race`. First, we will create an empty cell by replacing `agegrp` with a missing value for individuals in the third level of `race` and sixth level of `agegrp`.

```
. use https://www.stata-press.com/data/r19/nhanes2

. replace agegrp = . if race==3 & agegrp==6
(11 real changes made, 11 to missing)
```

Then, we fit our model, collecting only the coefficients (`_r_b`), and we use the `quietly` prefix to suppress the output. To keep the table compact, we include only the interaction and not the main effects of each variable.

```
. quietly: collect _r_b: regress bpsystol agegrp#race
```

Now, we specify that we want to display the empty cells in our table. Then, we arrange the items in our collection with the variable names on the rows and the statistics (`result`) on the columns:

```
. collect style showempty on
. collect layout (colname) (result)
Collection: default
      Rows: colname
   Columns: result
   Table 1: 19 x 1
```

	Coefficient
20-29 # White	0
20-29 # Black	-.2656245
20-29 # Other	-5.154448
30-39 # White	2.448515
30-39 # Black	5.456101
30-39 # Other	-.2603797
40-49 # White	8.440513
40-49 # Black	18.07027
40-49 # Other	10.91819
50-59 # White	17.43116
50-59 # Black	25.61819
50-59 # Other	8.398711
60-69 # White	23.25529
60-69 # Black	29.49347
60-69 # Other	34.57295
70+ # White	30.24816
70+ # Black	33.6855
70+ # Other	0
Intercept	117.5104

Because there are no observations for individuals who are in their 70s and in the `Other` category of race, we see a coefficient of 0. If we change our mind and decide to hide the empty cells, we can turn this setting `off` and preview our updated table:

```
. collect style showempty off
. collect preview
```

	Coefficient
20-29 # White	0
20-29 # Black	-.2656245
20-29 # Other	-5.154448
30-39 # White	2.448515
30-39 # Black	5.456101
30-39 # Other	-.2603797
40-49 # White	8.440513
40-49 # Black	18.07027
40-49 # Other	10.91819
50-59 # White	17.43116
50-59 # Black	25.61819
50-59 # Other	8.398711
60-69 # White	23.25529
60-69 # Black	29.49347
60-69 # Other	34.57295
70+ # White	30.24816
70+ # Black	33.6855
Intercept	117.5104

Stored results

`collect style showempty` stores the following in `s()`:

Macros

`s(collection)` name of collection

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

[TABLES] **collect query** — Query collection style properties

[TABLES] **collect style showbase** — Collection styles for displaying base levels

[TABLES] **collect style showomit** — Collection styles for displaying omitted coefficients

collect style showomit — Collection styles for displaying omitted coefficients

Description   Menu   Syntax   Option
Remarks and examples   Stored results   Also see

Description

`collect style showomit` controls the visibility of coefficients and related statistics for omitted covariates in estimation results. When collecting results from a model in which covariates have been omitted, you can specify whether these omitted covariates should be shown or omitted from your tables.

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Build and style table

Syntax

`collect style showomit` { `on` | `off` } [, `name(`*cname*`)`]

where *cname* is a collection name.

Option

`name(`*cname*`)` specifies a collection *cname* to which the style is applied. By default, the style is applied to the current collection.

Remarks and examples

`collect style showomit` controls whether coefficients and related statistics are shown for omitted covariates. By default, omitted coefficients are displayed in tables.

To be more specific, this setting applies to the following results: `_r_b`, `_r_se`, `_r_z`, `_r_z_abs`, `_r_p`, `_r_lb`, `_r_ub`, `_r_ci`, `_r_df`, `_r_cri`, `_r_crlb`, and `_r_crub`. These are simply the names that are assigned by default to the coefficients, standard errors, test statistics, upper and lower confidence bounds, degrees of freedom, and upper and lower critical interval bounds that are collected with either `collect get` or the `collect` prefix.

When you explore different model specifications, it is useful to see the omitted covariates in the output from the command. However, when you create tables of estimation results that you will be sharing with others, you may prefer to suppress the display of omitted covariates. To do this, you can type

```
. collect style showomit off
```

Stored results

`collect style showomit` stores the following in `s()`:

Macros
    `s(collection)`   name of collection

Also see

[TABLES] **collect query** — Query collection style properties

[TABLES] **collect style showbase** — Collection styles for displaying base levels

[TABLES] **collect style showempty** — Collection styles for displaying empty cells

collect style table — Collection styles for table headers

Description Quick start Menu Syntax Options
Remarks and examples Stored results Reference Also see

Description

`collect style table` specifies table header style properties. When `collect layout` specifies that multiple tables be created, `collect style table` determines how the headers for each of the tables are displayed.

Quick start

Use an `x` to delimit interaction terms

```
collect style table, delimiter(" x ")
```

Use a colon to delimit table headers with multiple dimensions

```
collect style table, dimdelimiter(" : ")
```

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Build and style table

Syntax

`collect style table` [, *options*]

options	Description
`name(`*cname*`)`	specify table header styles for collection *cname*
`nodelimiter`	place factor-variable and interaction elements in separate cells without a delimiter
`delimiter(`*delim*`)`	use *delim* to delimit interaction terms composed in a single cell
`atdelimiter(`*atdelim*`)`	use *atdelim* to delimit interaction terms containing the `@` symbol
`bardelimiter(`*bardelim*`)`	use *bardelim* to delimit interaction terms containing the `|` symbol
`binder(`*binder*`)`	use *binder* to separate factor variables from their levels
`dimdelimiter(`*dimdelim*`)`	use *dimdelim* to delimit table headers
`dimbinder(`*dimbinder*`)`	use *dimbinder* to separate dimensions from their levels

Options

name(*cname*) specifies the collection to which table header style properties are to be applied. By default, properties are applied to the current collection.

nodelimiter, delimiter(), atdelimiter(), and bardelimiter() control how to compose factor-variable and interaction terms in headers.

nodelimiter, the default, specifies that factor-variable and interaction term elements (matrix stripe elements) be split into separate cells.

delimiter(*delim*) specifies that factor-variable and interaction term elements (matrix stripe elements) be composed in a single cell.

The variables in an interaction term are composed in a single cell using *delim* as the delimiter.

Factor-variable terms serve as their own dimension nested within the stripe dimensions coleq, colname, roweq, and rowname. Option binder() controls how levels of factor variables are composed within a single cell.

atdelimiter(*atdelim*) specifies that *atdelim* be used to delimit interaction terms containing the @ symbol. This option is applicable when, for example, working with results from contrast, mean, proportion, ratio, and total.

bardelimiter(*bardelim*) specifies that *bardelim* be used to delimit interaction terms containing the | symbol. This option is applicable when, for example, working with results from anova and manova.

binder(*binder*) specifies how to compose levels of factor variables within a single cell.

The binder will be applied as long as the factor variable and its levels are not hidden. Note that the default style used by collect, which is style-default.stjson, will hide the dimension from the headers. You can use collect style header to specify whether to display the label or name for a dimension and whether to display the label or value for the level of a dimension.

dimdelimiter(*dimdelim*) specifies how to delimit table headers composed from multiple dimensions. The default is to use a comma as the delimiter.

dimbinder(*dimbinder*) specifies how dimension-level pairs are composed within the table headers.

Remarks and examples

collect style table specifies how the headers of individual tables are to be composed when collect layout specifies that multiple tables are to be created. When creating multiple tables, you may have one or more dimensions defining the tables. With a single dimension, you may have the title of the dimension and the label for the level of the dimension. For these tables, you can specify the delimiter used to separate the dimension from its level. With two dimensions, you may have multiple titles and labels, in which case you may also want to specify the delimiter for the dimensions.

▷ Example 1 Delimiters for dimensions

We use data from the Second National Health and Nutrition Examination Survey (NHANES II) (McDowell et al. 1981), and we model the occurrence of a heart attack as a function of systolic blood pressure, age, and body mass index (bmi). We fit two different models and collect the coefficients (_r_b) and standard errors (_r_se) for each.

```
. use https://www.stata-press.com/data/r19/nhanes2
(Second National Health and Nutrition Examination Survey)
. quietly: collect _r_b _r_se: logit heartatk bpsystol bmi
. quietly: collect _r_b _r_se : logit heartatk bpsystol age
```

We would like to create tables that focus on the coefficient and standard error for `bpsystol`. We can include just those variables in which we are interested by specifying the levels of `colname` when arranging the items in our collection with `collect layout`. The dimension `cmdset` identifies the commands from which we have collected results. We place the levels of this dimension on the columns and leave the row specification empty. We also create separate tables for the results of the covariate `bpsystol`:

```
. collect layout () (cmdset) (colname[bpsystol]#result)
Collection: default
   Columns: cmdset
    Tables: colname[bpsystol]#result
   Table 1: 1 x 2
   Table 2: 1 x 2
Systolic blood pressure, Coefficient
------------------
       1         2
------------------
.0155575 -.0021038
------------------

Systolic blood pressure, Std. error
-----------------
       1        2
-----------------
.0018229 .0020584
-----------------
```

The first label we see corresponds to the level `bpsystol` of the dimension `colname`. The second corresponds to the level of the dimension `result`. By default, a comma is used as the delimiter for the dimensions. We would instead like to use a colon, with spaces on each side. We specify that below with the `dimdelimiter()` option. We also label the levels of `cmdset` to indicate coefficients are adjusted for another covariate, center the results, and add extra space between columns. Then, we preview our table:

```
. collect style table, dimdelimiter(" : ")
. collect label levels cmdset 1 "Model 1 (BMI adjusted)"
>     2 "Model 2 (age adjusted)"
. collect style cell, halign(center)
. collect style column, extraspace(1)
. collect preview
Systolic blood pressure : Coefficient
-----------------------------------------------
Model 1 (BMI adjusted)   Model 2 (age adjusted)
-----------------------------------------------
       .0155575                -.0021038
-----------------------------------------------

Systolic blood pressure : Std. error
-----------------------------------------------
Model 1 (BMI adjusted)   Model 2 (age adjusted)
-----------------------------------------------
       .0018229                 .0020584
-----------------------------------------------
```

While it is rare that we would want to report coefficients and standard errors in separate tables, we can see that `collect style table` is useful for controlling the look of table headers when `collect` creates multiple tables at once.

◁

Stored results

`collect style table` stores the following in `s()`:

Macros
`s(collection)` name of collection

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

[TABLES] **collect query** — Query collection style properties

[TABLES] **collect style header** — Collection styles for hiding and showing header components

Description Quick start Menu Syntax
Options Remarks and examples Stored results Also see

Description

`collect style tex` specifies styles to be used when exporting a table from a collection to a LaTeX file.

`collect style tex`, typed without any options, will clear the existing LaTeX appearance styles for the current collection.

Quick start

Specify that tables not be centered on the page when exported from the current collection to a LaTeX file

```
collect style tex, nocentering
```

Clear the current LaTeX appearance styles

```
collect style tex
```

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Collect styles > Styles for LaTeX

Syntax

Specify styles to be used when exporting a collection to a LaTeX file

`collect style tex` [, *options*]

Clear existing LaTeX appearance styles

`collect style tex` [, `name(`*cname*`)`]

options	Description
`name(`*cname*`)`	apply LaTeX styles to collection *cname*
[`no`]`begintable`	specify whether to use LaTeX table environment
[`no`]`centering`	specify whether to center table horizontally on the page

Options

`name(`*cname*`)` specifies that the LaTeX styles be applied to collection *cname*.

When `name(`*cname*`)` is specified without any other options, LaTeX styles are cleared from collection *cname*.

The default in both cases is to apply the style changes to the current collection.

`begintable` and `nobegintable` control whether to use the LaTeX table environment.

`begintable` specifies that `collect export` use the LaTeX table environment. In addition, table titles are specified using `\caption`.

`nobegintable` specifies that `collect export` not use the LaTeX table environment.

`centering` and `nocentering` control whether to center the table horizontally on the page.

`centering` specifies that `collect export` center the table horizontally on the page.

`nocentering` specifies that `collect export` not center the table horizontally on the page.

Remarks and examples

`collect style tex` allows you to specify styles for the table that you will export to a LaTeX file with `collect export`. If you do not like the change you have made, you can clear the LaTeX appearance styles by typing the following:

```
. collect style tex
```

This change will be applied to the current collection. To make this change for another collection, specify the collection name with the `name()` option.

Stored results

`collect style tex` stores the following in `s()`:

Macros
    `s(collection)`   name of collection

Also see

[TABLES] **collect query** — Query collection style properties

[TABLES] **collect export** — Export table from a collection

collect style title — Collection styles for table titles

Description

`collect style title` specifies appearance styles for table titles in the collection. This includes bolding, italics, font, text color, and shading. Certain appearance edits can be rendered only on certain export formats.

Quick start

Make the table titles bold

```
collect style title, font(, bold)
```

When exporting the table to a LaTeX file, use the LaTeX macro `textbf` to render the title

```
collect style title, latex(textbf)
```

Clear the table title styles

```
collect style title, clear
```

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Build and style table

Syntax

`collect style title` [, *options*]

options	Description
Main	
`name(`*cname*`)`	apply table title appearance styles to collection *cname*
`clear`	remove table title styles
Fonts	
`font(`[*fontfamily*] [, *font_opts*]`)`	set font style for title text
`smcl(`*smcl*`)`	specify formatting for SMCL files
`latex(`*latex*`)`	specify LaTeX macro
Shading	
`shading(`*sspec*`)`	set background color, foreground color, and fill pattern for titles

font_opts	Description
`size(`# [*unit*]`)`	specify font size
`color(`*color*`)`	specify font color
`variant(`*variant*`)`	specify font variant and capitalization
[`no`]`bold`	specify whether to format text as bold
[`no`]`italic`	specify whether to format text as italic
[`no`]`strikeout`	specify whether to strike out text
[`no`]`underline`	specify whether to underline text

fontfamily specifies a font family.

sspec is

[`background(`*bgcolor*`)` `foreground(`*fgcolor*`)` `pattern(`*fpattern*`)`]

bgcolor specifies the background color.

fgcolor specifies the foreground color.

fpattern specifies the fill pattern. A complete list of fill patterns is shown in *Shading patterns* of [TABLES] **Appendix**.

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

unit may be `in` (inch), `pt` (point), or `cm` (centimeter). An inch is equivalent to 72 points and 2.54 centimeters. The default is `pt`.

Options

Main

`name(`*cname*`)` specifies the collection to which title appearance styles are to be applied. By default, the title styles are applied to the current collection.

`clear` removes existing title style properties.

Fonts

`font(`[*fontfamily*] [`, size(`# [*unit*]`) color(`*color*`) variant(`*variant*`)` [`no`]`bold` [`no`]`italic` [`no`]`strikeout` [`no`]`underline`]`)` specifies the font style for the title text.

These font style properties are applicable when publishing items from a collection to Microsoft Word, Microsoft Excel, PDF, HTML, and LaTeX files.

fontfamily specifies a font family.

`size(`# [*unit*]`)` specifies the font size as a number optionally followed by units. If # is specified without the optional *unit*, points is assumed.

`color(`*color*`)` specifies the font color.

`variant(`*variant*`)` specifies the font variant and capitalization. *variant* may be `allcaps`, `smallcaps`, or `normal`. `variant(allcaps)` changes the text to all uppercase letters. `variant(smallcaps)` changes the text to use large capitals for uppercase letters and smaller capitals for lowercase letters. `variant(normal)` changes the font variant back to normal; capitalization is unchanged from the original text.

This style property is applicable only when publishing to HTML and LaTeX files.

`bold` and `nobold` specify the font weight. `bold` changes the font weight to bold; `nobold` changes the font weight back to normal.

`italic` and `noitalic` specify the font style. `italic` changes the font style to italic; `noitalic` changes the font style back to normal.

`strikeout` and `nostrikeout` specify whether to add a strikeout mark to the text. `strikeout` adds a strikeout mark to the text; `nostrikeout` changes the text back to normal.

Only one of `strikeout` or `underline` is allowed when publishing to HTML files.

`underline` and `nounderline` specify whether to underline the title. `underline` adds a single line under the title; `nounderline` removes the underline.

Only one of `strikeout` or `underline` is allowed when publishing to HTML files.

`smcl(`*smcl*`)` specifies how to render title text for SMCL output. The supported SMCL directives are `input`, `error`, `result`, and `text`.

This style property is applicable only when publishing to a SMCL file.

`latex(`*latex*`)` specifies the name of a LaTeX macro to render title text for LaTeX output. This style property is applicable only when publishing to a LaTeX file.

Example LaTeX macro names are `textbf`, `textsf`, `textrm`, and `texttt`. Custom LaTeX macros are also allowed. If *title* is the table title, then *latex* is translated to the following when exporting to LaTeX:

latex{*title*}

Shading

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

These shading style properties are applicable when publishing to Microsoft Excel, Microsoft Word, PDF, and HTML.

Remarks and examples

collect style title allows you to customize the appearance of the title added with collect title. For example, if you export your table to a Microsoft Excel, a Microsoft Word, a PDF, an HTML, or a LaTeX file, you can customize the title by changing the font, text color, and more. Additionally, you can use SMCL directives and LaTeX macros to specify how to render the table title.

To list the current appearance styles for the table title, you can use collect query title.

▷ Example 1: Customizing the font for the table title

Suppose that we want to create a table comparing regression results and we want to export that table to an Excel file. Below, we load data from the Second National Health and Nutrition Examination Survey (NHANES II) (McDowell et al. 1981). We fit two models for systolic blood pressure and specify the title with collect title:

```
. use https://www.stata-press.com/data/r19/nhanes2l
(Second National Health and Nutrition Examination Survey)
. quietly: collect _r_b: regress bpsystol i.agegrp
. quietly: collect _r_b: regress bpsystol i.agegrp i.sex
. collect title "Models for systolic blood pressure"
```

Then, we suppress the base levels, hide the labels for the levels of dimension result, and lay out our table:

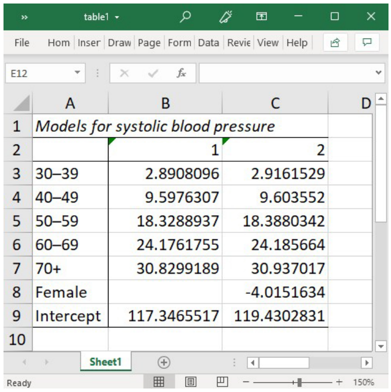
```
. collect style showbase off
. collect style header result, level(hide)
. collect layout (colname) (cmdset#result)
Collection: default
      Rows: colname
   Columns: cmdset#result
   Table 1: 7 x 2
Models for systolic blood pressure
```

	1	2
30-39	2.89081	2.916153
40-49	9.597631	9.603552
50-59	18.32889	18.38803
60-69	24.17618	24.18566
70+	30.82992	30.93702
Female		-4.015163
Intercept	117.3466	119.4303

Here we see that our title is added in plain text. We wish to make our title italic, so we use the `font(, italic)` option and then export our table to a file called `table1.xlsx`.

```
. collect style title, font(, italic)

. collect export table1.xlsx, replace
(collection default exported to file table1.xlsx)
```

When we open the file, we see the following:



	A	B	C	D
1	*Models for systolic blood pressure*			
2		1	2	
3	30–39	2.8908096	2.9161529	
4	40–49	9.5976307	9.603552	
5	50–59	18.3288937	18.3880342	
6	60–69	24.1761755	24.185664	
7	70+	30.8299189	30.937017	
8	Female		-4.0151634	
9	Intercept	117.3465517	119.4302831	
10				

Stored results

`collect style title` stores the following in `s()`:

Macros
`s(collection)`   name of collection

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

[TABLES] **collect query** — Query collection style properties

[TABLES] **collect title** — Add a custom table title in a collection

Description

`collect style use` reads style properties and layout information from a file and applies them to a collection.

Quick start

Apply any style properties in `mystyle.stjson` that are not specified in the current collection; also apply layout information from the file if no layout exists in the collection

```
collect style use mystyle
```

Same as above, but for any style properties that are specified in both `mystyle.stjson` and the current collection, override the current specification with the one in `mystyle.stjson`

```
collect style use mystyle, override
```

Replace the current style with the one specified in `mystyle.stjson`

```
collect style use mystyle, replace
```

Replace the current style and layout information with those specified in `mystyle.stjson`

```
collect style use mystyle, replace layout
```

Menu

Statistics > Summaries, tables, and tests > Tables and collections > Collect styles > Use styles

Syntax

`collect style use` *style* [, *options*]

style specifies the name of a file that defines layout information and style properties. If *style* is not a filename or a file path, then the following search logic is employed:

1. search ado-path for `style-`*style*`.stjson`; use this file if found.
2. search ado-path for *style*`.stjson`; use this file if found.

options	Description
`name(`*cname*`)`	apply style and layout information to collection *cname*
`layout`	replace current layout with the layout defined in *style*
`override`	give precedence to style information in *style* over the collection's current style
`replace`	replace the collection's style with the one defined in *style*
[`no`]`warn`	display or suppress notes about tags that are not recognized; default is to display

Options

`name(`*cname*`)` specifies a collection *cname* to which the style and layout information are applied. By default, the style and layout information are applied to the current collection.

`layout` replaces the collection's layout with the layout defined in *style*.

`override` specifies that style properties specified in *style* should take precedence over the styles in the collection. The default is to give precedence to the styles in the collection if those styles are found both in *style* and in the collection.

`replace` specifies that the collection's style properties be replaced with the style properties defined in *style*.

`warn` and `nowarn` control the display of notes when `collect` encounters a tag it does not recognize.

`warn`, the default, specifies that `collect` display notes when it encounters a tag it does not recognize.

`nowarn` specifies that `collect` not show the notes.

These options override the `collect_warn` setting; see [TABLES] **set collect_warn**.

Remarks and examples

`collect style use` allows you to apply the style and layout information from a file to another collection. You can choose to apply only the style information or both the style and layout information. By default, if a collection has a layout, `collect style use` will keep that layout. Otherwise, `collect style use` will use the layout defined in *style*. Also by default, for any style properties that are specified in both the collection and the file being loaded, the specifications in the collection will take precedence. However, you can choose to give precedence to the style properties in the file or to completely replace the current style with the one in the file.

To demonstrate, we use data from the Second National Health and Nutrition Examination Survey (NHANES II) (McDowell et al. 1981). Below, we fit a model for systolic blood pressure as a function of age. We use the `collect` prefix to collect the coefficients (`_r_b`), and we specify the `quietly` prefix to suppress the output.

```
. use https://www.stata-press.com/data/r19/nhanes2
. quietly: collect _r_b: regress bpsystol age
```

Then, we make some modifications to the style. First, we format the results to display only two digits after the decimal. Then, we specify that the constant (`_cons`) be placed at the end of the list of covariates. Next, we arrange the values in our collection with `collect layout`. We place the covariate names (`colname`) on the rows and the statistics (`result`) on the columns. We save these style properties and layout information in a file called `myreg.stjson`. The `replace` option allows us to overwrite that file if it exists.

```
. collect style cell, nformat(%5.2f)
. collect style _cons last
. collect layout (colname) (result)
Collection: default
      Rows: colname
   Columns: result
   Table 1: 2 x 1

-------------------------
            | Coefficient
------------+------------
Age (years) |        0.65
Intercept   |       99.86
-------------------------

. collect style save myreg, replace
(style from default saved to file myreg.stjson)
```

Next, we create a new collection called `logit`, which then becomes the current collection. In this collection, we collect coefficients from a logistic regression of `highbp`, which indicates whether someone has high blood pressure.

```
. collect create logit
(current collection is logit)
. quietly: collect _r_b: logit highbp age
```

Here we explore a different style. We list the constant first, and we place the variable names on the columns:

```
. collect style _cons first
. collect layout (result) (colname)
Collection: logit
      Rows: result
   Columns: colname
   Table 1: 1 x 2

-----------------------------------
            | Intercept Age (years)
------------+----------------------
Coefficient | -2.615888    .0472671
-----------------------------------
```

Looking at this table, we now decide that we prefer the style and layout from our other collection. We load that file with `collect style use`. We want to replace all our current style properties with those defined in `myreg.stjson`, so we specify the `replace` option. We also use the `layout` option to replace our current layout with the one from the file.

```
. collect style use myreg.stjson, replace layout
Collection: logit
      Rows: colname
   Columns: result
   Table 1: 2 x 1

. collect preview

-------------------------
             | Coefficient
-------------+-----------
Age (years)  |        0.05
Intercept    |       -2.62
-------------------------
```

Now, we see that our rows correspond to the covariates, the intercept is listed last, and our results are formatted with only two digits after the decimal.

Stored results

`collect style use` stores the following in `s()`:

Macros

`s(collection)`	name of collection
`s(filename)`	name of the file used

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

[TABLES] **collect style save** — Save collection styles to disk

Description   Also see

Description

This is the appendix for `collect style cell`, `collect style notes`, and `collect style title`.

Border patterns

bpattern	
nil	thickThinMediumGap
single	thinThickThinMediumGap
thick	thinThickLargeGap
double	thickThinLargeGap
dotted	thinThickThinLargeGap
dashed	wave
dotDash	doubleWave
dotDotDash	dashSmallGap
triple	dashDotStroked
thinThickSmallGap	threeDEmboss
thickThinSmallGap	threeDEngrave
thinThickThinSmallGap	outset
thinThickMediumGap	inset

Diagonal border patterns

dbpattern	
none	hair
thin	medium_dashed
medium	dash_dot
dashed	medium_dash_dot
dotted	dash_dot_dot
thick	medium_dash_dot_dot
double	slant_dash_dot

Colors

bgcolor, *fgcolor*, and *color*

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

Use `nil` or none to remove a previously specified color.

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

Also see

[TABLES] **collect style cell** — Collection styles for cells

[TABLES] **collect style notes** — Collection styles for table notes

[TABLES] **collect style title** — Collection styles for table titles

Description Remarks and examples Also see

Description

This entry is a self-contained introduction to tags, dimensions, and levels and how you use them in `collect layout` to specify and create tables. It introduces other commands that are helpful in laying out tables along the way. It uses simple examples on real data to demonstrate all concepts.

It explains what tags are and why they are organized into dimensions that contain levels. It explains the inner workings of `collect layout` so you can understand when things do not go as you expect. It demonstrates how to create one-way, two-way, multiway, and stacked tables and discusses what to do when things go wrong.

Admittedly, there is quite a bit of overlap with [TABLES] **Intro 2**. Unlike **Intro 2**, this entry is focused solely on laying out tables.

Remarks and examples

Remarks are presented under the following headings:

Basic concepts

How do you make collections work for you? The answer is you just use tags organized into the levels of dimensions to request tabular results. What? Let's give meaning to that sentence.

We start by collecting something. That something will be incredibly simple. The undocumented Stata command `echo` simply displays whatever number or string you type and returns that number or string in `r(value)`.

```
. echo 11
value = 11

. return list

scalars:
            r(value) =  11
```

To collect its results, we simply prefix our `echo` command with `collect:`, but let's do a little more. Let's collect the result and give it the tag `myres1`.

```
. collect, tag(myres1[]): echo 11
value = 11
```

Do not worry for now about the `[]` after `myres1`; just know that we have collected the value 11 and tagged it with `myres1`.

The collect system is built to create tables of results, perhaps lots of different results from different commands. The way we get things out of a collection is to lay out a table. We have only one value collected, so let's create the world's simplest table.

```
. collect layout (myres1)

Collection: default
      Rows: myres1
   Table 1: 1 x 1

──
11
──
```

collect layout is the command to specify the layout of a table. Its first argument is a parentheses-bound list of the tags that we want on the rows of the table, in this case (myres1). A tag is simply a way to name and find things. We tagged our value 11 as myres1. When we asked for myres1, collect layout gave our 11 back to us.

You may have noticed that we did not include the [] on myres. We could have; it would make no difference.

Let's add another value to our collection.

```
. collect, tag(myres2[]): echo 22
value = 22
```

And let's show "all" of this as a table.

```
. collect layout (myres1 myres2)

Collection: default
      Rows: myres1 myres2
   Table 1: 2 x 1

──
11
22
──
```

We could go on, but I think we are going to get tired of typing myres1 ....

Tags do not have to be a simple name; in fact, they rarely are. Tables tend to put a set of related things on the rows and another set of related things on the columns. The contents of the table are the intersection of those related things. Consider a cross-tabulation of region and sex.

	Sex	
	Male	Female
Region		
NE	1,018	1,078
MW	1,310	1,464
S	1,332	1,521
W	1,255	1,373

"NE", "MW", "S", and "W" are the related things on the rows. "Male" and "Female" are the related things on the columns. The counts in the cells of the table are the intersection when both the row "thing" and column "thing" are true. On this table, that is all obvious, but it is also at the heart of how tags are used in the collect system.

Tags in the `collect` system provide a structure that directly supports sets of related things. Tags are organized as dimensions that contain levels. In the table above, `region` is a dimension, and the levels are `NE`, `MW`, `S`, and `W`. Likewise, `sex` is another dimension whose levels are `Male` and `Female`.

If this all seems like an unnecessary abstraction, it is not. The table above was a simple cross-tabulation of two categorical variables. But that need not be the case. One of our dimensions might be sets of regressions with different covariates. Or it might be sets of results from different datasets. All categorical variables can be dimensions, but not all dimensions can be categorical variables.

Let's now use the level within dimension organization to create a more interesting table. First, we clear our current collection.

```
. collect clear
```

We collect the results of an `echo` but give it two tags.

```
. collect, tag(myrow[1] mycol[1]): echo 11
value = 11
```

We have tagged value 11 with `myrow[1]` and `mycol[1]`. We read tag `myrow[1]` as "dimension `myrow`, level 1" or "level 1 in dimension `myrow`".

Let's collect and tag more results from `echo` commands.

```
. collect, tag(myrow[2] mycol[1]): echo 21
value = 21
. collect, tag(myrow[1] mycol[2]): echo 12
value = 12
. collect, tag(myrow[2] mycol[2]): echo 22
value = 22
```

You might see where this is heading.

Now, we can create a table from our four collected values,

```
. collect layout (myrow[1] myrow[2]) (mycol[1] mycol[2])
Collection: Table
      Rows: myrow[1] myrow[2]
   Columns: mycol[1] mycol[2]
   Table 1: 3 x 2

─────────────┬────────────
             │    mycol
             │    1     2
─────────────┼────────────
myrow        │
  1          │   11    12
  2          │   21    22
─────────────┴────────────
```

The first parentheses-bound list still specifies the tags we want on the rows of our table. The second parentheses-bound list specifies the tags we want on the columns of our table.

You can specify multiple levels inside the `[]`; thus, a better way to type the layout command above is

```
. collect layout (myrow[1 2]) (mycol[1 2])
```

If you are following along, type it. You will get the same result.

Better still, you can refer to an entire dimension and all the tags defined by its levels by typing just the dimension name. The concise way to specify our table is

```
. collect layout (myrow) (mycol)
```

And now you see why `collect` organizes its tags as levels within dimensions.

Let's elaborate on that point just a bit. You can tell from this example that it may take more than one tag to uniquely identify a value. Each of our 4 values required 2 tags, for example, value 12 required `myrow[1]` and `mycol[2]`. Thus, there is a great advantage to representing tags as levels within dimensions. If it takes two tags to uniquely identify a value and you organize those tags as the rows and columns of a table, your values will naturally populate the cells of a table.

Moreover, this idea generalizes to higher-dimensional tables. If each of your values requires 3 tags and those tags can be arranged in 3 dimensions, you have the makings of a 3-dimensional (3D) table. One rarely presents 3D tables as their natural cube. It is hard to print. They are usually presented as tables with super rows or super columns. Regardless, dimensions give you a natural way to specify the structure of a table, whether that structure is a simple table with rows and columns or it is a table with columns, super columns, rows, super rows, and super-super rows.

If you came here just to learn about the terms "tag", "dimension", and "level", you can stop reading.

Basics in practice

Let's put this organization to use on a real collection.

Grab the venerable (but familiar) automobile dataset.

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

Clear our default collection.

```
. collect clear
```

And collect the results of a simple regression.

```
. collect: regress mpg displacement i.foreign

      Source |       SS           df       MS      Number of obs   =        74
-------------+----------------------------------   F(2, 71)        =     35.57
       Model |  1222.85283         2  611.426414   Prob > F        =    0.0000
    Residual |  1220.60663        71  17.1916427   R-squared       =    0.5005
-------------+----------------------------------   Adj R-squared   =    0.4864
       Total |  2443.45946        73  33.4720474   Root MSE        =    4.1463

------------------------------------------------------------------------------
         mpg | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
displacement |  -.0469161   .0066931    -7.01   0.000    -.0602618   -.0335704
             |
     foreign |
    Foreign  |  -.8006817   1.335711    -0.60   0.551    -3.464015    1.862651
       _cons |   30.79176   1.666592    18.48   0.000     27.46867    34.11485
------------------------------------------------------------------------------
```

Just so you know, every number saved in the `e()` results after `regress`, which includes every number displayed in the results above, has been pulled into the collection. You just have to tell `collect` how you would like them pulled out and displayed.

But, you wonder, we did not specify any tags. What can we possibly do? We can do a lot. `collect` creates tags for us behind the scenes. We get a list of the dimensions by typing

```
. collect dims

Collection dimensions
Collection: default
─────────────────────────────────────────
                   Dimension   No. levels
─────────────────────────────────────────
Layout, style, header, label
                      cmdset   1
                       coleq   1
                     colname   4
           colname_remainder   1
                     foreign   2
               program_class   1
                      result   33
                 result_type   3
                     rowname   1
Style only
                border_block   4
                   cell_type   4
─────────────────────────────────────────
```

Let's focus on two of those dimensions. First, `colname`,

```
. collect levelsof colname

Collection: default
 Dimension: colname
    Levels: displacement 0.foreign 1.foreign _cons
```

Those look promising. They are the coefficient names from our regression. And, yes, the levels are strings—`displacement`, `0.foreign`, `1.foreign`, and `_cons`. Dimension levels can be either integers or strings, and the strings can have spaces if you wish.

We apologize for the name `colname`; it is a bit arcane. It comes from the fact that Stata matrices have `colnames`, and this dimension was taken from the `colnames` on the `e(b)` matrix saved by `regress`. We will also find that many different commands save many different things that need to go into the `colname` dimension. There simply is no good name for all the levels that can appear in `colname`. If it makes you feel any better, everyone does eventually get used to typing `colname`.

If you really cannot abide `colname`, you can actually change it. Type

```
. collect remap colname = parameters
```

Now, you can type `parameters` instead of `colname`.

Second, the dimension, `result`, sounds truly promising.

```
. collect levelsof result

Collection: default
 Dimension: result
    Levels: F N _r_b _r_ci _r_df _r_lb _r_p _r_se _r_ub _r_z _r_z_abs beta
            cmd cmdline depvar df_m df_r estat_cmd ll ll_0 marginsok model
            mss predict properties r2 r2_a rank rmse rss sum_w title vce
```

That is a lot to figure out. We recognize some things: `r2` sounds as if it might be "R^2", and `rmse` might be "Root mean squared error". What about those underscore things—`_r_b`, `_r_se`, `_r_ci`. We might guess. Let's not. Let's use another command that gives us a bit more information, `collect label list`.

```
. collect label list result, all

  Collection: default
   Dimension: result
       Label: Result
Level labels:
           F  F statistic
           N  Number of observations
        _r_b  Coefficient
       _r_ci  __LEVEL__% CI
       _r_df  df
       _r_lb  __LEVEL__% lower bound
        _r_p  p-value
       _r_se  Std. error
       _r_ub  __LEVEL__% upper bound
        _r_z  t
    _r_z_abs  |t|
        beta  Standardized coefficient
         cmd  Command
     cmdline  Command line as typed
      depvar  Dependent variable
        df_m  Model DF
        df_r  Residual DF
   estat_cmd  Program used to implement estat
          ll  Log likelihood
        ll_0  Log likelihood, constant-only model
   marginsok  Predictions allowed by margins
       model  Model
         mss  Model sum of squares
     predict  Program used to implement predict
  properties  Command properties
          r2  R-squared
        r2_a  Adjusted R-squared
        rank  Rank of VCE
        rmse  RMSE
         rss  Residual sum of squares
       sum_w  Sum of the weights
       title  Title of output
         vce  SE method
```

We have listed all the levels of dimension `result` and the labels for each level. Now this dimension does look promising; it includes all the results from the regression. Apparently, level `_r_b` is the level that refers to the coefficients. `_r_se` refers to the standard errors of the coefficients. `_r_ci` is a little odd because apparently it contains a placeholder for the level of significance. Regardless, it looks like a confidence interval. Many of the levels are a one-to-one match with the names of the `e()` results—`df_m`, `df_r`, `ll`, `r2`, ... In fact, all the `e()` results are here, and they have the same names they had in `e()`.

We say all the `e()` results, but that is not quite true. `e(V)` is excluded unless you explicitly collect it. Why would we need the full VCE? Also, `e(b)` is not here. It is effectively here because you can use the level `_r_b` to access the coefficient values.

It seems as if we have enough information to pull some values out of the collection using their tags. Let's pull the value for R^2. From the listing above, we know the dimension (`result`) and level (`r2`) of its tag.

```
. collect layout (result[r2])

Collection: default
      Rows: result[r2]
   Table 1: 1 x 1

-----------------------
R-squared  |  .5004596
-----------------------
```

How about grabbing all the results by just using the whole `result` dimension.

```
. collect layout (result)

Collection: default
      Rows: result
   Table 1: 24 x 1

-------------------------------------------------------------------------
F statistic                            |                         35.56533
Number of observations                 |                               74
Standardized coefficient               |                        -.7447313
Command                                |                          regress
Command line as typed                  | regress mpg displacement i.foreign
Dependent variable                     |                              mpg
Model DF                               |                                2
Residual DF                            |                               71
Program used to implement estat        |                    regress_estat
Log likelihood                         |                        -208.7139
Log likelihood, constant-only model    |                        -234.3943
Predictions allowed by margins         |                       XB default
Model                                  |                              ols
Model sum of squares                   |                         1222.853
Program used to implement predict      |                        regres_p
Command properties                     |                              b V
R-squared                              |                         .5004596
Adjusted R-squared                     |                         .4863881
Rank of VCE                            |                                3
RMSE                                   |                         4.146281
Residual sum of squares                |                         1220.607
Sum of the weights                     |                               74
Title of output                        |                Linear regression
SE method                              |                              ols
-------------------------------------------------------------------------
```

Well, that is both more and less than we probably expected. Regarding the "more", we probably do not care about "Command" or "Command line" or several of the other string results (really macro results). Let's ask specifically for what we want and for the order we want.

```
. collect layout (result[N F df_r df_m r2 r2_a rmse ll])

Collection: default
      Rows: result[N F df_r df_m r2 r2_a rmse ll]
   Table 1: 8 x 1

-------------------------------------
Number of observations  |          74
F statistic             |    35.56533
Residual DF             |          71
Model DF                |           2
R-squared               |    .5004596
Adjusted R-squared      |    .4863881
RMSE                    |    4.146281
Log likelihood          |   -208.7139
-------------------------------------
```

More importantly, where are our coefficients? The answer is that no coefficient can be uniquely identified by just the tag `result[_r_b]`. There were three coefficients in our model, one for `displacement`, one for `1.foreign`, and one for `_cons`. Tag `result[_r_b]` refers to all of those, but `collect layout` needs us to tell it where each of those coefficients goes in the table. We have not done that. Just as we needed both a `row` and a `column` dimension to create our table in *Basic concepts*, we need another dimension to create a table with coefficients. Recall that the `colname` dimension enumerated the coefficient names; that is what we need.

```
. collect layout (colname) (result[_r_b])
Collection: default
      Rows: colname
   Columns: result[_r_b]
   Table 1: 4 x 1
```

	Coefficient
Displacement (cu. in.)	-.0469161
Domestic	0
Foreign	-.8006817
Intercept	30.79176

We put the `colname` dimension on our table's rows and the `result` dimension on our table's columns. We also limited the `result` dimension to the level `_r_b`.

Let's get a more complete regression table by adding some levels to the `result` dimension.

```
. collect layout (colname) (result[_r_b _r_se _r_p _r_ci])
Collection: default
      Rows: colname
   Columns: result[_r_b _r_se _r_p _r_ci]
   Table 1: 4 x 4
```

	Coefficient	Std. error	p-value	95% CI	
Displacement (cu. in.)	-.0469161	.0066931	0.000	-.0602618	-.0335704
Domestic	0	0			
Foreign	-.8006817	1.335711	0.551	-3.464015	1.862651
Intercept	30.79176	1.666592	0.000	27.46867	34.11485

How collect layout processes tag specifications

When we specify layouts, it is helpful to understand what `collect layout` does with the tags we specify for the rows and columns. When we type

```
. collect layout (result[N F r2])
```

a search is performed to see whether any values are tagged `result[N]`. If exactly one value with that tag is found, `collect layout` creates a row in the table for `result[N]` and places that value into the newly created row. If nothing is found with that tag, `collect layout` does nothing. If more than one thing with that tag is found, `collect layout` does nothing. Then, the process repeats for values tagged `result[F]` and finally `result[r2]`. That is it.

When we type the command

```
. collect layout (result)
```

the process is as we just described, but it is done for every level in the dimension `result`, not just for the levels `N`, `F`, and `r2`.

Let's call this process enumerating the levels of a dimension.

Enumerating a single dimension is all that is required for a one-way table, like the one we just specified. Two-way tables add just a bit to this process. When we type

```
. collect layout (colname) (result)                                          (1)
```

the command not only enumerates the levels in dimension `colname` and `result` but also interacts all the levels of `colname` and `result`. Let's specify the levels we want to make this a bit easier to explain.

```
. collect layout (colname[displacement _cons]) (result[_r_b _r_se r2])       (2)
```

`collect layout` begins with the tag `colname[displacement]`, which might form the first row. It looks sequentially for all pairings of `colname[displacement]` with the levels of `result`. It looks first for values that are tagged `colname[displacement]` and tagged `result[_r_b]`. If it finds exactly one value, it creates the row for `colname[displacement]` and the column for `result[_r_b]` and places the value it finds in that row/column position. It then looks for values tagged `colname[displacement]` and `result[_r_se]`. If it finds exactly one value with those tags, it places that value in the correct row and column. It then does the same thing for the tags `colname[displacement]` and `result[r2]`. That completes the process for the `displacement` row in the table.

`collect layout` then repeats that whole process with `colname[_cons]` to create the potential second row in the table.

Why do we say "potential" second row? Because it is possible that for some pairings of the levels of `colname` and `result`, `collect layout` will not find a unique value. Or that it will always find multiple values. If either happens for a whole row or column, then that row or column is not created.

The whole process is hardly different when we type

```
. collect layout (colname) (result)                                          (3)
```

In this case, `collect layout` enumerates over all the levels of `result` within all the levels of `colname`, rather than just the three levels of `result[_b_r _b_se r2]` within the two levels of `colname[displacement _cons]`, which we explicitly specified in (2).

An important thing to realize is that `collect layout` must find exactly one thing or it does nothing. Why can't it handle finding more than one thing? The row and column arguments to `collect layout` specify both what to look for and where to put it. Each level from the row specification is a possible row for the table. Each level from the column specification is a possible column for the table. Finding multiple values for a row and column combination means that we have not told `collect layout` where those values go. It means that we have not included enough dimensions in our specification.

We can also tell you that in (2) our use of `r2` in `result[_r_b _r_se r2]` had no effect on the table. It yields exactly the same table as typing `result[_r_b _r_se]`. Type it and see. Why? Because the value of R^2 is a model statistic, not a coefficient. It is not tagged with any specific variable. It is not tagged with `colname[displacement]` or with `colname[_cons]`. You cannot find model statistics when the `result` dimension is interacted with `colname`. More on that in *The process in practice*.

`collect layout` always interacts row and column specifications. That is really what makes a table a table. We can also explicitly specify interactions. That lets us create multiway tables rather than just two-way tables.

The process in practice

Our collection currently has a single regression. What if we wanted to compare that regression with another regression? Let's add `weight` to our regression and collect those results.

```
. collect: regress mpg displacement i.foreign weight

      Source |       SS           df       MS      Number of obs   =        74
-------------+----------------------------------   F(3, 70)        =     45.88
       Model |  1619.71935         3  539.906448   Prob > F        =    0.0000
    Residual |  823.740114        70  11.7677159   R-squared       =    0.6629
-------------+----------------------------------   Adj R-squared   =    0.6484
       Total |  2443.45946        73  33.4720474   Root MSE        =    3.4304

------------------------------------------------------------------------------
         mpg | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
displacement |   .0019286   .0100701     0.19   0.849    -.0181556    .0220129
             |
     foreign |
    Foreign  |  -1.600631   1.113648    -1.44   0.155    -3.821732    .6204699
      weight |  -.0067745   .0011665    -5.81   0.000    -.0091011   -.0044479
       _cons |   41.84795   2.350704    17.80   0.000     37.15962    46.53628
------------------------------------------------------------------------------
```

We might want to see how the additional covariate affects the coefficient on `displacement`.

```
. collect layout (colname) (result[_r_b _r_se _r_p _r_ci])

Collection: default
      Rows: colname
   Columns: result[_r_b _r_se _r_p _r_ci]
   Table 1: 1 x 4

------------------------------------------------------------------
               | Coefficient Std. error p-value          95% CI
---------------+--------------------------------------------------
Weight (lbs.)  |   -.0067745   .0011665   0.000 -.0091011 -.0044479
------------------------------------------------------------------
```

That is disappointing. We typed just what we typed to create a table from a single regression. We added another whole regression, and we get just one row?

Let's apply what we learned in *How collect layout processes tag specifications*. The first thing `collect layout` searched for was the first level of dimension `colname` interacted with the first specified level of dimension `result`. That would be the two tags `colname[displacement]` and `result[_r_b]`. That search finds two values: −0.047 from the first regression and 0.002 from the second regression. `collect layout` did not find a unique value, so it did nothing. That same thing happens when `collect layout` searches for `colname[displacement]` in combination with `result[_r_se]`, `result[_r_p]`, and `result[_r_ci]`. So there is nothing to report for the whole potential first row. The whole sequence happens again for the second level of `colname`—`colname[0.foreign]`. Two values are again found for each of the specified levels of `result`.

The only time `collect layout` finds a single value for each level of `result` is when it enumerates the `weight` level of dimension `colname`. That is the only coefficient that appears in only one of our two regressions. We clearly need to somehow add a dimension to our table, a dimension whose levels represent our regressions.

Let's again list all the dimensions in our collection and see whether there is anything promising.

```
. collect dims

Collection dimensions
Collection: default
-----------------------------------------
                      Dimension   No. levels
-----------------------------------------
Layout, style, header, label
                         cmdset   2
                          coleq   1
                        colname   5
              colname_remainder   1
                        foreign   2
                  program_class   1
                         result   33
                    result_type   3
                        rowname   1

Style only
                   border_block   4
                      cell_type   4
-----------------------------------------
```

`cmdset` looks promising. Let's learn a bit more about that dimension.

```
. collect label list cmdset, all

  Collection: default
   Dimension: cmdset
       Label: Command results index
Level labels:
           1
           2
```

We see `Command results index`, which does indeed look promising.

How do we add that dimension? We previously hinted that multiway tables could be specified by interacting additional dimensions with those already specified on the rows or columns. We perform that interaction using the same operator we use to create interactions in factor variables—`#`.

Let's try interacting dimension `cmdset` with dimension `colname`. We will interact with `colname` because it is on the row dimension and we do not have room for any more columns.

```
. collect layout (colname#cmdset) (result[_r_b _r_se _r_p _r_ci])
Collection: default
      Rows: colname#cmdset
   Columns: result[_r_b _r_se _r_p _r_ci]
   Table 1: 14 x 4
```

	Coefficient	Std. error	p-value	95% CI	
Displacement (cu. in.)					
1	-.0469161	.0066931	0.000	-.0602618	-.0335704
2	.0019286	.0100701	0.849	-.0181556	.0220129
Domestic					
1	0	0			
2	0	0			
Foreign					
1	-.8006817	1.335711	0.551	-3.464015	1.862651
2	-1.600631	1.113648	0.155	-3.821732	.6204699
Weight (lbs.)					
2	-.0067745	.0011665	0.000	-.0091011	-.0044479
Intercept					
1	30.79176	1.666592	0.000	27.46867	34.11485
2	41.84795	2.350704	0.000	37.15962	46.53628

That is not bad. We have all the coefficients, standard errors, p-values, and confidence intervals from both regressions. They are not exactly organized the way they are in most comparative regression tables.

Let's go for that organization. We will need to put the dimension `cmdset` onto the columns, and then interact the coefficient names (dimension `colname`) with the statistics (dimension `result`) on the rows. We have room on the rows, so let's just ask for all the levels of dimension `result`.

```
. collect layout (colname#result) (cmdset)

Collection: default
      Rows: colname#result
   Columns: cmdset
   Table 1: 48 x 2

-------------------------------------------------------------------------------
                            |                     1                       2
----------------------------+--------------------------------------------------
Displacement (cu. in.)      |
  Coefficient               |             -.0469161                .0019286
  95% CI                    | -.0602618 -.0335704 -.0181556        .0220129
  df                        |                    71                      70
  95% lower bound           |             -.0602618               -.0181556
  p-value                   |                 0.000                   0.849
  Std. error                |              .0066931                .0100701
  95% upper bound           |             -.0335704                .0220129
  t                         |                 -7.01                    0.19
  |t|                       |                  7.01                    0.19
  Standardized coefficient  |             -.7447313                .0306148
Domestic                    |
  Coefficient               |                     0                       0
  df                        |
  Std. error                |                     0                       0
  Standardized coefficient  |                     0                       0
Foreign                     |
  Coefficient               |             -.8006817               -1.600631
  95% CI                    | -3.464015  1.862651 -3.821732        .6204699
  df                        |                    71                      70
  95% lower bound           |             -3.464015               -3.821732
  p-value                   |                 0.551                   0.155
  Std. error                |              1.335711                1.113648
  95% upper bound           |              1.862651                .6204699
  t                         |                 -0.60                   -1.44
  |t|                       |                  0.60                    1.44
  Standardized coefficient  |             -.0636875               -.1273168
Weight (lbs.)               |
  Coefficient               |                                     -.0067745
  95% CI                    |                         -.0091011 -.0044479
  df                        |                                            70
  95% lower bound           |                                     -.0091011
  p-value                   |                                         0.000
  Std. error                |                                      .0011665
  95% upper bound           |                                     -.0044479
  t                         |                                         -5.81
  |t|                       |                                          5.81
  Standardized coefficient  |                                     -.9100491
Intercept                   |
  Coefficient               |              30.79176                41.84795
  95% CI                    |  27.46867  34.11485  37.15962        46.53628
  df                        |                    71                      70
  95% lower bound           |              27.46867                37.15962
  p-value                   |                 0.000                   0.000
  Std. error                |              1.666592                2.350704
  95% upper bound           |              34.11485                46.53628
  t                         |                 18.48                   17.80
  |t|                       |                 18.48                   17.80
-------------------------------------------------------------------------------
```

That is most of what we would want in a comparative regression table and a bit more. We probably do not want both the confidence interval and separately its upper and lower bound. Folks would disagree about which among the standard error, t statistic, p-values, and confidence interval should be included.

More importantly, where are the overall model F statistic, the R^2, and the other model results? We saw earlier that these are in the `result` dimension, and we asked for everything in the `result` dimension.

This again comes down to how `collect layout` constructs the table by enumerating the levels in the specified dimensions. We discussed earlier that the row specification is interacted with the column specification. We specifically requested an interaction of `colname` and `result` on the rows. So, because `collect layout` enumerates all combinations of `cmdset`, `colname`, and `result`, it is always trying to find a unique value for a specific level of each of these dimensions.

If we want overall model results, the problem with the fully interacted enumeration is that it always includes a level for `colname`. Model results cannot be tagged with a `colname`. They are not associated with any variable or other parameter. We need to ask for results that do not include a `colname`. Easy enough; we never said that the dimensions in row or column specifications had to be interacted. They can also be stacked, one after the other. We can add the `result` dimension to our row specification again, but this time not interacting it with `colname`.

```
. collect layout (colname#result result) (cmdset)
```

We have added a whole new set of enumerations to our table. After enumerating all possible combinations of the levels of `colname`, `result`, and `cmdset`, `collect layout` will then enumerate all possible combinations of just `result` and `cmdset`.

Before we run that, let's put back our request for a subset of the levels of `result` when interacted with `colname`. We will leave all the model results, just to see what is there.

```
. collect layout (colname#result[_r_b _r_se _r_z _r_p] result) (cmdset)
```

Okay, we, the authors, tried that, and the result will not fit in the width of the page you are reading. So let's ask for only one of our regressions first, just so we can see what is there.

```
. collect layout (colname#result[_r_b _r_se _r_z _r_p] result) (cmdset[1])

Collection: default
      Rows: colname#result[_r_b _r_se _r_z _r_p] result
   Columns: cmdset[1]
   Table 1: 42 x 1
```

	1
Displacement (cu. in.)	
Coefficient	-.0469161
Std. error	.0066931
t	-7.01
p-value	0.000
Domestic	
Coefficient	0
Std. error	0
Foreign	
Coefficient	-.8006817
Std. error	1.335711
t	-0.60
p-value	0.551
Intercept	
Coefficient	30.79176
Std. error	1.666592
t	18.48
p-value	0.000
F statistic	35.56533
Number of observations	74
Standardized coefficient	-.7447313
Command	regress
Command line as typed	regress mpg displacement i.foreign
Dependent variable	mpg
Model DF	2
Residual DF	71
Program used to implement estat	regress_estat
Log likelihood	-208.7139
Log likelihood, constant-only model	-234.3943
Predictions allowed by margins	XB default
Model	ols
Model sum of squares	1222.853
Program used to implement predict	regres_p
Command properties	b V
R-squared	.5004596
Adjusted R-squared	.4863881
Rank of VCE	3
RMSE	4.146281
Residual sum of squares	1220.607
Sum of the weights	74
Title of output	Linear regression
SE method	ols

Goodness, that even includes the command line as typed. For our comparison, let's request a subset of the model results by specifying specific levels of dimension `result`.

```
. collect layout (colname#result[_r_b _r_se _r_z _r_p] result[N F r2 ll])
> (cmdset)

Collection: default
      Rows: colname#result[_r_b _r_se _r_z _r_p] result[N F r2 ll]
   Columns: cmdset
   Table 1: 27 x 2
```

	1	2
Displacement (cu. in.)		
Coefficient	-.0469161	.0019286
Std. error	.0066931	.0100701
t	-7.01	0.19
p-value	0.000	0.849
Domestic		
Coefficient	0	0
Std. error	0	0
Foreign		
Coefficient	-.8006817	-1.600631
Std. error	1.335711	1.113648
t	-0.60	-1.44
p-value	0.551	0.155
Weight (lbs.)		
Coefficient		-.0067745
Std. error		.0011665
t		-5.81
p-value		0.000
Intercept		
Coefficient	30.79176	41.84795
Std. error	1.666592	2.350704
t	18.48	17.80
p-value	0.000	0.000
Number of observations	74	74
F statistic	35.56533	45.88031
R-squared	.5004596	.6628796
Log likelihood	-208.7139	-194.1637

There is a lot we could do to make this table prettier. You can learn about that by reading the examples in this manual. What we hope is that you are now more comfortable with how and why "you just use tags organized into the levels of dimensions to request tabular results."

Also see

[TABLES] **Intro 2** — A tour of concepts and commands

[TABLES] **collect layout** — Specify table layout for the current collection

Description

Predefined styles provide an easy way to customize the look of a table. You can access predefined styles by typing

```
. collect style use stylename
```

when you create a table with `collect` or by specifying the `style(`*stylename*`)` option when you create a table with the `dtable`, `etable`, `table`, `tabulate oneway`, `tabulate twoway`, `tabulate`, `summarize()`, `svy: tabulate oneway`, or `svy: tabulate twoway` command.

You can create your own style files, which you can use over and over as you build tables. Or you can use any of the numerous style files that are installed with Stata.

Remarks and examples

Remarks are presented under the following headings:

Creating a new style

If you find yourself specifying the same `collect style` commands to many of your tables, you can create a style once and then apply it to many tables you create in the future.

For example, suppose you use the `table` command to produce tables of regression results, such as the ones you would get by typing

```
. use https://www.stata-press.com/data/r19/nhanes2
. table, command(regress bpsystol age weight) ///
        command(regress bpsystol age weight i.region)
```

By default, `table` will use the style named `table`, which means `table` uses the styles defined in the file `style-table.stjson`, which is installed with Stata. However, suppose that you find that you almost always want to show the values rather than the default labels for the `command` dimension, hide the titles but show the labels for other dimensions, and show the base categories for only the main effects but not the interaction terms for factor variables in the models. After each `table` command similar to the one above, you could type

```
. collect style header, title(hide) level(label)
. collect style header command, level(value)
. collect style showbase factor
. collect preview
```

to make these modifications. Alternatively, you could create your own style by typing

```
. collect style clear
. collect style use style-table, replace
. collect style header, title(hide) level(label)
. collect style header command, level(value)
. collect style showbase factor
. collect style save mytablereg, replace
```

In the first line above, we clear the current collection's style properties. In the second line, we specify that we want to start with the styles that `table` uses by default. The third through fifth lines apply the style edits that we prefer. Finally, in the last line we save our style, giving it the name `mytablereg`.

Once we have created this style, we can now use it with subsequent `table` commands. For example, we can type

```
. table, command(regress bpsystol age weight) ///
        command(regress bpsystol age weight i.region) style(mytablereg)
```

More generally, we can create a new style by typing

```
. collect clear
. collect style use basestyle, replace
. style modifications
. collect style save mystyle, replace
```

The `collect style use` command is not necessary, but it is often most convenient to start with a style that you are familiar with such as `style-table`, which is the default for the `table` command, or `style-default`, which is the default for the `collect` command, and then make edits to that style.

After you have created your new style, you can apply it to a table created by `collect` by typing

```
. collect style use mystyle
```

or by adding the `style(`*mystyle*`)` option to your `dtable`, `etable`, `table`, or `tabulate` command.

Default styles

The following styles define the default style properties for `collect`, `dtable`, `etable`, `table`, and `tabulate`.

These styles are composed from other targeted styles and are not directly created by any `collect style` commands. Targeted styles are defined by `collect style` commands that focus on specific style properties, such as borders, headers, or formats.

default

This style is the default for tables created by `collect`. It is defined in `style-default.stjson` and is composed as follows:

```
{
  "StyleIncludes": [
    "default_borders",
    "default_cidelimiter",
    "default_halign",
    "default_headers",
    "default_margins",
    "default_nformats",
    "default_smcl",
    "default_tex"
  ]
}
```

This style can be accessed by typing

```
. collect style use default
```

or adding the `style(default)` option to the `table` command.

You can change the default style for `collect` with `set collect_style`; see [TABLES] **set collect_style**.

For an example of the `default` style, see the first example in [TABLES] **Example 5**.

dtable

This style is the default for tables created by `dtable`. It is defined in `style-dtable.stjson` and is composed as follows:

```
{
  "StyleIncludes": [
    "default_margins",
    "default_smcl",
    "default_tex",
    "dtable_borders",
    "dtable_font",
    "dtable_halign",
    "dtable_headers",
    "dtable_nformats"
  ]
}
```

This style can be accessed by typing

```
. collect style use dtable
```

or adding the `style(default)` option to the `dtable` command.

You can change the default style for `dtable` with `set dtable_style`; see [TABLES] **set dtable_style**.

etable

This style is the default for tables created by `etable`. It is defined in `style-etable.stjson` and is composed as follows:

```
{
  "StyleIncludes": [
    "default_cidelimiter",
    "default_margins",
    "default_smcl",
    "default_tex",
    "etable_borders",
    "etable_etable",
    "etable_font",
    "etable_halign",
    "etable_headers",
    "etable_nformats",
    "etable_showitem",
    "etable_stars"
  ]
}
```

This style can be accessed by typing

```
. collect style use etable
```

or adding the `style(etable)` option to the `etable` command.

You can change the default style for `etable` with `set etable_style`; see [TABLES] **set etable_style**.

For an example of the `etable` style, see [TABLES] **Example 6**.

lcstats

This style is the default for tables created by `lcstats`. It is defined in `style-lcstats.stjson` and is composed as follows:

```
{
  "StyleIncludes": [
    "default_margins",
    "default_tex",
    "default_borders",
    "lcstats_font",
    "lcstats_halign",
    "lcstats_headers",
    "lcstats_nformats",
    "lcstats_smcl",
    "lcstats_txt"
  ]
}
```

This style can be accessed by typing

```
. collect style use lcstats
```

or adding the `style(lcstats)` option to the `lcstats` command.

table

This style is the default for tables created by `table`. It is defined in `style-table.stjson` and is composed as follows:

```
{
  "StyleIncludes": [
    "default_borders",
    "default_halign",
    "default_margins",
    "default_smcl",
    "default_tex",
    "table_cidelimiter",
    "table_headers",
    "table_nformats"
  ]
}
```

This style can be accessed by typing

```
. collect style use table
```

or adding the `style(table)` option to the `table` command.

You can change the default style for `table` with `set table_style`; see [TABLES] **set table_style**.

For an example of the `table` style, see the first example in [TABLES] **Example 2**.

tabulate

This style is the default for tables created by `tabulate`, `tab1`, `tab2`, `tabi`, and `svy: tabulate`. It is defined in `style-tabulate.stjson` and is composed as follows:

```
{
  "StyleIncludes": [
    "default_halign",
    "default_margins",
    "default_smcl",
    "default_tex",
    "tabulate_borders",
    "tabulate_headers",
    "tabulate_nformats"
  ]
}
```

This style can be accessed by typing

```
. collect style use tabulate
```

or adding the `style(tabulate)` option to the `tabulate` command.

You can change the default style for `tabulate` with `set tabulate_style`; see [TABLES] **set tabulate_style**.

For examples of this style, see *Publish your tables* in [R] **tabulate oneway**, [R] **tabulate twoway**, [R] **tabulate, summarize()**, [SVY] **svy: tabulate oneway**, and [SVY] **svy: tabulate twoway**.

Other styles

The following styles are installed with Stata. They are example styles intended to help you customize your tables.

anova

This style can be used with results from anova or oneway and has the building blocks for constructing ANOVA tables. It is defined in style-anova.stjson and is composed as follows:

```
{
  "StyleIncludes": [
    "anova_borders",
    "anova_halign",
    "anova_headers",
    "anova_layout"
    "anova_nformats",
    "default_margins",
    "default_smcl",
    "default_tex",
  ]
}
```

This style can be accessed by typing

```
. collect style use anova, replace
```

or, if you want to keep the collection's current style properties that do not conflict with this style, by typing

```
. collect style use anova, layout override
```

For an example of the anova style, see the first example in [TABLES] **Example 8**.

coef-table

This style is useful for building tables with model coefficients. It is defined in style-coef-table.stjson and is composed as follows:

```
{
  "StyleIncludes": [
    "coef-table_halign",
    "coef-table_headers",
    "default_borders",
    "default_cidelimiter",
    "default_margins",
    "default_nformats",
    "default_smcl",
    "default_tex"
  ]
}
```

This style can be accessed by typing

```
. collect style use coef-table
```

or adding the style(coef-table) option to the table command.

table-1

This style builds on the `table` style and has the following modifications:

1. The names of statistics, the levels of the `result` dimension, are hidden.
2. The row headers are stacked into a single column, and vertical space is added between dimensions.
3. The row headers are right-aligned.

This style is defined using the following commands:

```
. collect style use style-table, replace
. collect style header result, level(hide)
. collect style row stack, nodelimiter spacer
. collect style cell cell_type[row-header], halign(right)
```

This style is saved in `style-table-1.stjson` and can be accessed by typing

```
. collect style use table-1
```

or adding the `style(table-1)` option to the `table` command.

For an example of the `table-1` style, see *Classic Table 1* in [R] **table summary**.

table-reg1

This style builds on the `table` style and has the following modification:

1. The level labels for the `command` dimension, the full commands typed in the `command()` option, are hidden.

This style is defined using the following commands:

```
. collect style use style-table, replace
. collect style header command, level(value)
```

This style is saved in `style-table-reg1.stjson` and can be accessed by typing

```
. collect style use table-reg1
```

or adding the `style(table-reg1)` option to the `table` command.

For an example of the `table-reg1` style, see *Regression results with factor variables* in [R] **table regression**.

table-reg1-fv1

This style builds on the `table` style and has the following modifications:

1. The dimension titles are hidden for all dimensions, and the level labels are shown for all dimensions other than `command`.
2. The level labels for the `command` dimension, the full commands typed in the `command()` option, are hidden.
3. The base category is shown for the main effects of factor variables but not for interactions.

This style is defined using the following commands:

```
. collect style use style-table, replace
. collect style header, title(hide) level(label)
. collect style header command, level(value)
. collect style showbase factor
```

This style is saved in `style-table-reg1-fv1.stjson` and can be accessed by typing

```
. collect style use table-reg1-fv1
```

or adding the `style(table-reg1-fv1)` option to the `table` command.

For an example of the `table-reg1-fv1` style, see *Regression results with factor variables* in [R] **table regression**.

table-reg2

This style builds on the `table` style and has the following modifications:

1. The level labels for the `command` dimension, the full commands typed in the `command()` option, are hidden.
2. The names of statistics, the levels of the `result` dimension, are hidden.
3. The row headers are right-aligned.

This style is defined using the following commands:

```
. collect style use style-table, replace
. collect style header command, level(value)
. collect style header result, level(hide)
. collect style cell cell_type[row-header], halign(right)
```

This style is saved in `style-table-reg2.stjson` and can be accessed by typing

```
. collect style use table-reg2
```

or adding the `style(table-reg2)` option to the `table` command.

table-reg2-fv1

This style builds on the `table` style and has the following modifications:

1. The dimension titles are hidden for all dimensions, and the level labels are shown for all dimensions other than `command`.
2. The level labels for the `command` dimension, the full commands typed in the `command()` option, are hidden.
3. The names of statistics, the levels of the `result` dimension, are hidden.
4. The row headers are right-aligned.
5. The base category is shown for the main effects of factor variables but not for interactions.

This style is defined using the following commands:

```
. collect style use style-table, replace
. collect style header, title(hide) level(label)
. collect style header command, level(value)
. collect style header result, level(hide)
. collect style cell cell_type[row-header], halign(right)
. collect style showbase factor
```

This style is saved in `style-table-reg2-fv1.stjson` and can be accessed by typing

```
. collect style use table-reg2-fv1
```

or adding the `style(table-reg2-fv1)` option to the `table` command.

For an example of the `table-reg2-fv1` style, see *Regression results with factor variables* in [R] **table regression**.

table-reg3

This style builds on the `table` style and has the following modifications:

1. The level labels for the `command` dimension, the full commands typed in the `command()` option, are hidden.
2. The names of statistics, the levels of the `result` dimension, are hidden.
3. The values in the body of the table are horizontally centered within the cells.
4. The row headers are right-aligned.
5. The row headers are stacked into a single column, and vertical space is added between dimensions.

This style is defined using the following commands:

```
. collect style use style-table, replace
. collect style header command, level(value)
. collect style header result, level(hide)
. collect style cell cell_type[item], halign(center)
. collect style cell cell_type[row-header], halign(right)
. collect style row stack, spacer
```

This style is saved in `style-table-reg3.stjson` and can be accessed by typing

```
. collect style use table-reg3
```

or adding the `style(table-reg3)` option to the `table` command.

For an example of the `table-reg3` style, see *Tables with results from a single command* in [R] **table regression**.

table-reg3-fv1

This style builds on the `table` style and has the following modifications:

1. The dimension titles are hidden for all dimensions, and the level labels are shown for all dimensions other than `command` and `result`.
2. The level labels for the `command` dimension, the full commands typed in the `command()` option, are hidden.
3. The names of statistics, the levels of the `result` dimension, are hidden.
4. The values in the body of the table are horizontally centered within the cells.
5. The row headers are right-aligned.
6. The row headers are stacked into a single column, and vertical space is added between dimensions.
7. The base category is shown for the main effects of factor variables but not for interactions.

This style is defined using the following commands:

```
. collect style use style-table, replace
. collect style header, title(hide) level(label)
. collect style header command, level(value)
. collect style header result, level(hide)
. collect style cell cell_type[item], halign(center)
. collect style cell cell_type[row-header], halign(right)
. collect style row stack, spacer
. collect style showbase factor
```

This style is saved in `style-table-reg3-fv1.stjson` and can be accessed by typing

```
. collect style use table-reg3-fv1
```

or adding the `style(table-reg3-fv1)` option to the `table` command.

table-right

This style builds on the `table` style and has the following modification:

1. The row headers are right-aligned.

This style is defined using the following commands:

```
. collect style use style-table, replace
. collect style cell cell_type[row-header], halign(right)
```

This style is saved in `style-table-right.stjson` and can be accessed by typing

```
. collect style use table-right
```

or adding the `style(table-right)` option to the `table` command.

For an example of the `table-right` style, see *Customizing results* in [R] **table oneway**.

table-tab2

This style builds on the `table` style and has the following modifications:

1. The row headers are right-aligned.
2. The names of statistics, the levels of the `result` dimension, are hidden.
3. The row headers are stacked into a single column, and vertical space is added between dimensions.

This style is defined using the following commands:

```
. collect style use style-table, replace
. collect style cell cell_type[row-header], halign(right)
. collect style header result, level(hide)
. collect style row stack, spacer
```

This style is saved in `style-table-tab2.stjson` and can be accessed by typing

```
. collect style use table-tab2
```

or adding the `style(table-tab2)` option to the `table` command.

For an example of the `table-tab2` style, see *Customizing results* in [R] **table twoway**.

Targeted styles

The following styles each focus on specific style properties and are referenced by other styles.

anova_borders

This style defines cell border properties targeted to look like most ANOVA tables in Stata output. It is part of the definition for the `anova` style and is defined using the following commands:

```
. collect style clear
. collect style cell border_block[corner], border(bottom right)
. collect style cell border_block[row-header], border(right)
. collect style cell border_block[column-header], border(bottom)
. collect style cell term[Total], border(top)
```

This style is saved in `style-anova_borders.stjson` and can be accessed by typing

```
. collect style use anova_borders
```

as one step in building your own style.

anova_halign

This style defines horizontal alignment properties targeted to look like most ANOVA tables in Stata output. It is part of the definition for the anova style and is defined using the following commands:

```
. collect style clear
. collect style cell cell_type[row-header], halign(right)
```

This style is saved in style-anova_halign.stjson and can be accessed by typing

```
. collect style use anova_halign
```

as one step in building your own style.

anova_headers

This style defines header properties targeted to look like most ANOVA tables in Stata output. It is part of the definition for the anova style and is defined using the following commands:

```
. collect style clear
. collect style header, basestyle title(hide) level(label)
. collect style header term, level(value) fvlevels(hide)
. collect style column, extraspace(1) dups(center)
. collect style row stack, length(24) nobinder
```

This style is saved in style-anova_headers.stjson and can be accessed by typing

```
. collect style use anova_headers
```

as one step in building your own style.

anova_layout

This style defines the standard ANOVA layout. It is part of the definition for the anova style and is defined using the following commands:

```
. collect style clear
. collect layout (term) (result[SS df MS F p])
```

This style is saved in style-anova_layout.stjson and can be accessed by typing

```
. collect style use anova_layout
```

as one step in building your own style.

anova_nformats

This style defines numeric format properties. It is part of the definition for the anova style and is defined using the following commands:

```
. collect style clear
. collect style cell, basestyle nformat(%18.2f)
. collect style cell result[df], nformat(%18.0fc)
. collect style cell result[p], nformat(%6.4f) minimum(0.0001)
```

This style is saved in style-anova_nformats.stjson and can be accessed by typing

```
. collect style use anova_nformats
```

as one step in building your own style.

coef-table_halign

This style defines horizontal alignment properties targeted to look like Stata's coefficient and estimation tables. It is part of the definition for the `coef-table` style and is defined using the following commands:

```
. collect style clear
. collect style cell cell_type[corner], halign(center)
. collect style cell cell_type[column-header], halign(right)
. collect style cell cell_type[column-header]#result[_r_ci _r_cri], halign(center)
. collect style cell cell_type[item], halign(right)
. collect style cell cell_type[row-header]#coleq, halign(left)
. collect style cell cell_type[row-header]#colname, halign(right)
```

This style is saved in `style-coef-table_halign.stjson` and can be accessed by typing

```
. collect style use coef-table_halign
```

as one step in building your own style.

coef-table_headers

This style defines header properties targeted to look like Stata's coefficient and estimation tables. It is part of the definition for the `coef-table` style and is defined using the following commands:

```
. collect style clear
. collect style header, basestyle title(hide) level(label)
. collect style header stars, title(hide) level(hide)
. collect style row stack, noindent nobinder
```

This style is saved in `style-coef-table_headers.stjson` and can be accessed by typing

```
. collect style use coef-table_headers
```

as one step in building your own style.

default_borders

This style defines cell border properties targeted to look like most tables in Stata output. It is part of the definition for the `default`, `lcstats`, `table`, and `coef-table` styles and is defined using the following commands:

```
. collect style clear
. collect style cell border_block[corner], border(top right)
. collect style cell border_block[column-header], border(top)
. collect style cell border_block[row-header], border(top bottom right)
. collect style cell border_block[item], border(top bottom)
```

This style is saved in `style-default_borders.stjson` and can be accessed by typing

```
. collect style use default_borders
```

as one step in building your own style.

default_cidelimiter

This style defines the delimiters for confidence intervals and credible intervals. It is part of the definition for the default, coef-table, and etable styles and is defined using the following commands:

```
. collect style clear
. collect style cell, basestyle cidelimiter(" ") cridelimiter(" ")
```

This style is saved in style-default_cidelimiter.stjson and can be accessed by typing

```
. collect style use default_cidelimiter
```

as one step in building your own style.

default_halign

This style defines horizontal alignment properties. It is part of the definition for the default, table, and tabulate styles and is defined using the following commands:

```
. collect style clear
. collect style cell cell_type[corner], halign(center)
. collect style cell cell_type[column-header], halign(right)
. collect style cell cell_type[column-header]#result[_r_ci _r_cri], halign(center)
. collect style cell cell_type[item], halign(right)
. collect style cell cell_type[row-header], halign(left)
```

This style is saved in style-default_halign.stjson and can be accessed by typing

```
. collect style use default_halign
```

as one step in building your own style.

default_headers

This style defines header properties. It is part of the definition for the default style and is defined using the following commands:

```
. collect style clear
. collect style header, basestyle title(hide) level(label)
. collect style header stars, title(hide) level(hide)
. collect style row stack
```

This style is saved in style-default_headers.stjson and can be accessed by typing

```
. collect style use default_headers
```

as one step in building your own style.

default_margins

This style defines cell margin properties. It is part of the definition for the default, dtable, etable, lcstats, table, tabulate, and coef-table styles and is defined using the following commands:

```
. collect style clear
. collect style cell cell_type[column-header item], margin(left, width(10pt))
```

This style is saved in style-default_margins.stjson and can be accessed by typing

```
. collect style use default_margins
```

as one step in building your own style.

default_nformats

This style defines numeric format properties. It is part of the definition for the default and coef-table styles and is defined using the following commands:

```
. collect style clear
. collect style cell, basestyle nformat(%9.0g)
. collect style cell result[_r_b _r_se _r_lb _r_ub _r_ci _r_crlb _r_crub _r_cri] ///
        , nformat(%9.0g)
. collect style cell result[_r_z _r_z_abs], nformat(%8.2f)
. collect style cell result[_r_p], nformat(%5.3f)
```

This style is saved in style-default_nformats.stjson and can be accessed by typing

```
. collect style use default_nformats
```

as one step in building your own style.

default_smcl

This style defines SMCL properties targeted to look like most tables in Stata output. It is part of the definition for the default, dtable, etable, lcstats, table, tabulate, and coef-table styles and is defined using the following commands:

```
. collect style clear
. collect style cell cell_type[item], smcl(result)
```

This style is saved in style-default_smcl.stjson and can be accessed by typing

```
. collect style use default_smcl
```

as one step in building your own style.

default_tex

This style defines LaTeX properties. It is part of the definition for the default, dtable, etable, lcstats, table, tabulate, and coef-table styles and is defined using the following commands:

```
. collect style clear
. collect style tex, begintable centering
```

This style is saved in style-default_tex.stjson and can be accessed by typing

```
. collect style use default_tex
```

as one step in building your own style.

dtable_borders

This style defines cell border properties. It is part of the definition for the `dtable` style and is defined using the following commands:

```
. collect style clear
. collect style cell border_block[corner], border(top)
. collect style cell border_block[column-header], border(top)
. collect style cell border_block[row-header], border(top bottom)
. collect style cell border_block[item], border(top bottom)
```

This style is saved in `style-dtable_borders.stjson` and can be accessed by typing

```
. collect style use dtable_borders
```

as one step in building your own style.

dtable_composites

This style defines some convenient composite results for use with the `dtable` command. It is not part of the definition for the `dtable` style but is described in *Composite results* of [R] **dtable**.

This style is defined using the following commands:

```
. collect style clear
. collect composite define idi = p10 p90, trim
. collect composite define iqi = q1 q3, trim
. collect composite define rangei = min max, trim
. collect style cell result[idi iqi rangei], sformat("[%s]")
```

This style is saved in `style-dtable_composites.stjson` and can be accessed by typing

```
. collect style use dtable_composites
```

as one step in building your own style.

dtable_font

This style is a placeholder intended to define cell font properties. It is part of the definition for the `dtable` style, but its definition is currently empty.

dtable_halign

This style defines horizontal alignment properties. It is part of the definition for the `dtable` style and is defined using the following commands:

```
. collect style clear
. collect style cell cell_type[column-header], halign(center)
. collect style cell cell_type[row-header], halign(left)
. collect style cell stars[label], halign(left)
```

This style is saved in `style-dtable_halign.stjson` and can be accessed by typing

```
. collect style use dtable_halign
```

as one step in building your own style.

dtable_headers

This style defines header properties. It is part of the definition for the `dtable` style and is defined using the following commands:

```
. collect style clear
. collect style header, basestyle title(hide) level(label)
. collect style row stack, nobinder
. collect style column, dups(center)
```

This style is saved in `style-dtable_headers.stjson` and can be accessed by typing

```
. collect style use dtable_headers
```

as one step in building your own style.

dtable_nformats

This style defines numeric format properties. It is part of the definition for the `dtable` style and is defined using the following commands:

```
. collect style clear
. collect style cell, basestyle nformat(%21.3fc)
. collect style cell result[N count sumw frequency fvfrequency fvrawfrequency] ///
          , nformat(%21.0fc)
. collect style cell result[percent rawpercent fvpercent fvrawpercent] ///
          , nformat(%9.1fc) sformat("(%s%%)")
. collect style cell result[sd], sformat("(%s)")
. collect style cell result[ regress poisson lnormal
                             kwallis kwallis_ties ///
                             pearson fisher lrchi2 ///
                             svylr svywald svyllwald ///
          ], nformat(%6.3f) minimum(.001)
```

This style is saved in `style-dtable_nformats.stjson` and can be accessed by typing

```
. collect style use dtable_nformats
```

as one step in building your own style.

etable_borders

This style defines cell border properties. It is part of the definition for the `etable` style and is defined using the following commands:

```
. collect style clear
. collect style cell border_block[corner], border(top)
. collect style cell border_block[column-header], border(top)
. collect style cell border_block[row-header], border(top bottom)
. collect style cell border_block[item], border(top bottom)
```

This style is saved in `style-etable_borders.stjson` and can be accessed by typing

```
. collect style use etable_borders
```

as one step in building your own style.

etable_etable

This style defines properties unique to the etable command; see [R] **etable**. These properties control the default behavior for the column(), cstat(), eqrecode(), equations(), fvlabel, keep(), mstat(), showeq, showstars, showstarsnote, and varlabel options. It is part of the definition for the etable style.

This style is saved in style-etable_etable.stjson and can be accessed by typing

```
. collect style use etable_etable
```

as one step in building your own style.

etable_font

This style is a placeholder intended to define cell font properties. It is part of the definition for the etable style, but its definition is currently empty.

etable_halign

This style defines horizontal alignment properties. It is part of the definition for the etable style and is defined using the following commands:

```
. collect style clear
. collect style cell cell_type[column-header], halign(center)
. collect style cell cell_type[row-header], halign(left)
. collect style cell stars[label], halign(left)
```

This style is saved in style-etable_halign.stjson and can be accessed by typing

```
. collect style use etable_halign
```

as one step in building your own style.

etable_headers

This style defines header properties. It is part of the definition for the etable style and is defined using the following commands:

```
. collect style clear
. collect style header, basestyle title(hide) level(value)
. collect style header cmdset, level(label)
. collect style header stars, level(hide)
. collect style header result, level(label)
. collect style header result[ _r_b _r_se ///
                              _r_z _r_z_abs ///
                              _r_p ///
                              _r_lb _r_ub _r_ci ///
                              _r_crlb _r_crub  _r_cri ///
         ] , level(hide)
. collect style row stack, nobinder
. collect style column, dups(center)
```

This style is saved in style-etable_headers.stjson and can be accessed by typing

```
. collect style use etable_headers
```

as one step in building your own style.

etable_nformats

This style defines numeric format properties. It is part of the definition for the `etable` style and is defined using the following commands:

```
. collect style clear
. collect style cell, basestyle nformat(%9.2f)
. collect style cell result[N df_m df_r], nformat(%21.0f)
. collect style cell result[ _r_b _r_se ///
                             _r_lb _r_ub _r_ci ///
                             _r_crlb _r_crub  _r_cri ///
          ] , nformat(%9.3f)
. collect style cell result[_r_p] , nformat(%9.2f)
. collect style cell result[_r_se]#stars[value], sformat("(%s)")
. collect style cell result[_r_ci _r_cri]#stars[value], sformat("[%s]")
```

This style is saved in `style-etable_nformats.stjson` and can be accessed by typing

```
. collect style use etable_nformats
```

as one step in building your own style.

etable_showitem

This style defines `showbase`, `showomit`, and `showempty` properties. It is part of the definition for the `etable` style and is defined using the following commands:

```
. collect style clear
. collect style showbase off
. collect style showempty off
. collect style showomit off
```

This style is saved in `style-etable_showitem.stjson` and can be accessed by typing

```
. collect style use etable_showitem
```

as one step in building your own style.

etable_stars

This style defines stars properties. It is part of the definition for the `etable` style and is defined using the following commands:

```
. collect style clear
. collect stars _r_p .05 "*" .01 "**", attach(_r_b) dimension
```

This style is saved in `style-etable_stars.stjson` and can be accessed by typing

```
. collect style use etable_stars
```

as one step in building your own style.

lcstats_font

This style is a placeholder intended to define cell font properties. It is part of the definition for the `lcstats` style, but its definition is currently empty.

lcstats_halign

This style defines horizontal alignment properties. It is part of the definition for the `lcstats` style and is defined using the following commands:

```
. collect style clear
. collect style cell cell_type[column-header], halign(right)
. collect style cell cell_type[row-header], halign(right)
```

This style is saved in `style-lcstats_halign.stjson` and can be accessed by typing

```
. collect style use lcstats_halign
```

as one step in building your own style.

lcstats_headers

This style defines header properties. It is part of the definition for the `lcstats` style and is defined using the following commands:

```
. collect style clear
. collect style header, basestyle title(hide) level(label)
. collect style row stack, nobinder
. collect style column, dups(center) extraspace(1)
```

This style is saved in `style-lcstats_headers.stjson` and can be accessed by typing

```
. collect style use lcstats_headers
```

as one step in building your own style.

lcstats_nformats

This style defines numeric format properties. It is part of the definition for the `lcstats` style and is defined using the following commands:

```
. collect style clear
. collect style cell, basestyle nformat(%9.0gc)
. collect style cell result[N k_classes rank df], nformat(%21.0fc)
. collect style cell result[entropy], nformat(%6.4f)
. collect style cell result[mean sd], nformat(%21.2fc)
. collect style cell result[aic aicc bic caic], nformat(%21.2fc)
. collect style cell result[ll lmr vlmr], nformat(%21.2fc)
. collect style cell result[p_lmr p_vlmr], nformat(%6.3f) minimum(.001)
```

This style is saved in `style-lcstats_nformats.stjson` and can be accessed by typing

```
. collect style use lcstats_nformats
```

as one step in building your own style.

lcstats_smcl

This style defines SMCL properties. It is part of the definition for the `lcstats` style and is defined using the following commands:

```
. collect style clear
. collect style cell cell_type[item], smcl(result)
. collect style smcl, wrapnotes
```

This style is saved in `style-lcstats_smcl.stjson` and can be accessed by typing

```
. collect style use lcstats_smcl
```

as one step in building your own style.

lcstats_txt

This style defines plain text properties. It is part of the definition for the `lcstats` style and is defined using the following commands:

```
. collect style clear
. collect style txt, wrapnotes
```

This style is saved in `style-lcstats_txt.stjson` and can be accessed by typing

```
. collect style use lcstats_txt
```

as one step in building your own style.

table_cidelimiter

This style defines the delimiters for confidence intervals and credible intervals. It is part of the definition for the `table` style and is defined using the following commands:

```
. collect style clear
. collect style cell, basestyle cidelimiter("   ") cridelimiter("   ")
```

This style is saved in `style-table_cidelimiter.stjson` and can be accessed by typing

```
. collect style use table_cidelimiter
```

as one step in building your own style.

table_headers

This style defines header properties. It is part of the definition for the `table` style and is defined using the following commands:

```
. collect style clear
. collect style header, basestyle title(label) level(label)
. collect style column, dups(center) extraspace(1)
. collect style header across, title(hide)
. collect style header coleq, title(hide)
. collect style header colname, title(hide)
. collect style header command, title(hide)
. collect style header result, title(hide)
. collect style header roweq, title(hide)
. collect style header rowname, title(hide)
. collect style header stars, title(hide) level(hide)
. collect style header statcmd, title(hide)
. collect style header var, title(hide)
. collect style row stack
```

This style is saved in `style-table_headers.stjson` and can be accessed by typing

```
. collect style use table_headers
```

as one step in building your own style.

table_nformats

This style defines numeric format properties similar to `default_nformats` but adds numeric formats for targeted statistics of the `table` command. It is part of the definition for the `table` style and is defined using the following commands:

```
. collect style clear
. collect style cell, basestyle nformat(%9.0g)
. collect style cell result[_r_b _r_se _r_lb _r_ub _r_ci _r_crlb _r_crub _r_cri] ///
          , nformat(%9.0g)
. collect style cell result[_r_z _r_z_abs], nformat(%8.2f)
. collect style cell result[_r_p], nformat(%5.3f)
. collect style cell result[count frequency fvrawfrequency], nformat(%18.0fc)
. collect style cell result[fvpercent fvrawpercent percent rawpercent] ///
          , nformat(%9.2f)
. collect style cell result[fvproportion fvrawproportion proportion rawproportion] ///
          , nformat(%9.4g)
. collect style cell result[rawtotal sumw total fvfrequency], nformat(%18.6gc)
```

This style is saved in `style-table_nformats.stjson` and can be accessed by typing

```
. collect style use table_nformats
```

as one step in building your own style.

tabulate_borders

This style defines cell border properties. It is part of the definition for the `tabulate` style and is defined using the following commands:

```
. collect style clear
. collect style cell border_block[corner], border(top)
. collect style cell border_block[column-header], border(top)
. collect style cell border_block[row-header], border(top bottom)
. collect style cell border_block[item], border(top bottom)
. collect style cell border_block[row-header]#item_type[overall] ///
          , border(top bottom, pattern(none))
. collect style cell border_block[item]#item_type[overall] ///
          , border(top bottom, pattern(none))
```

This style is saved in `style-tabulate_borders.stjson` and can be accessed by typing

```
. collect style use tabulate_borders
```

as one step in building your own style.

tabulate_headers

This style defines header properties. It is part of the definition for the `tabulate` style and is defined using the following commands:

```
. collect style clear
. collect style header, basestyle title(label) level(label)
. collect style header result, title(hide)
. collect style header _overall item_type, title(hide) level(hide)
. collect style column, dups(center) extraspace(1)
. collect style row stack
```

This style is saved in `style-tabulate_headers.stjson` and can be accessed by typing

```
. collect style use tabulate_headers
```

as one step in building your own style.

tabulate_nformats

This style defines numeric format properties. It is part of the definition for the `tabulate` style and is defined using the following commands:

```
. collect style clear
. collect style cell, basestyle nformat(%18.0g)
. collect style cell result[frequency count], nformat(%18.0fc)
. collect style cell result[mean sd sumw expected cchi2 clrchi2], nformat(%18.1fc)
. collect style cell result[ CramersV ///
                           ase_gam ase_taub ///
                           chi2 chi2_lr ///
                           gamma taub ///
                           F_Pear F_Penl ///
                           cun_Pear cun_Penl ///
                           cun_LR cun_LRnl ///
                           F_LR F_LRnl ///
                           F_Wald F_LLW ///
                           Fun_Wald Fun_LLW ///
                           cun_Wald cun_LLW ///
                           mgdeff cvgdeff ///
         ], nformat(%18.4fc)
. collect style cell result[ p p_exact p1_exact p_lr ///
                           p_Pear p_Penl ///
                           p_LR p_LRnl ///
                           p_Wald p_LLW ///
                           pun_Wald pun_LLW ///
         ], nformat(%18.4fc) minimum(.0001)
. collect style cell result[proportion row_proportion column_proportion] ///
         , nformat(%18.4fc)
. collect style cell result[ row column cell ///
                           percent cumulative ///
    row_percent column_percent ///
         ] , nformat(%9.2f)
. collect style cell result[deff deft] , nformat(%9.3f)
. collect style cell result[ cv ///
                           df1_Pear df2_Pear ///
                           df1_Penl df2_Penl ///
                           df1_LR df2_LR ///
                           df1_LRnl df2_LRnl ///
         ], nformat(%18.2f)
. collect style header cmdset, title(hide) level(hide)
. collect style header var, title(hide) level(label)
```

This style is saved in `style-tabulate_nformats.stjson` and can be accessed by typing

```
. collect style use tabulate_nformats
```

as one step in building your own style.

Modifying the default style

If you routinely change your style to one of the styles installed with Stata or to one you have created, you can consider changing the style used by default. For information on changing the default style used by `collect`, see [TABLES] **set collect_style**. For information on changing the default style used by `etable`, see [TABLES] **set etable_style**. For information on changing the default style used by `dtable`, see [TABLES] **set dtable_style**. For information on changing the default style used by `table`, see [TABLES] **set table_style**. For information on changing the default style used by `tabulate` and `svy: tabulate`, see [TABLES] **set tabulate_style**.

Also see

[TABLES] **collect style use** — Use collection styles from disk

[TABLES] **set collect_style** — Style settings for collections

[TABLES] **set dtable_style** — Default style settings for dtable

[TABLES] **set etable_style** — Default style settings for etable

[TABLES] **set table_style** — Default style settings for table

[TABLES] **set tabulate_style** — Default style settings for tabulate

set collect_double — Storage type settings for collections

Description

set collect_double controls the storage type for numeric values that are saved in collections when using collect save. The default setting is on, which saves numeric values using double precision. When set collect_double is off, numeric values are converted to single precision before they are saved.

Syntax

set collect_double { on | off } [, permanently]

Option

permanently specifies that, in addition to making the change right now, the setting be remembered and become the default setting when you invoke Stata.

Remarks and examples

set collect_double controls the storage type for numeric values that are saved in collections when using collect save. If you wish to save your collection with values in single precision, you can type

```
. set collect_double off
```

The default setting is on, which saves numeric values using double precision. To apply the setting above permanently, meaning that it will be remembered when you invoke Stata, you can type

```
. set collect_double off, permanently
```

To see the current setting, you can type

```
. display c(collect_double)
```

Also see

[TABLES] **collect save** — Save a collection to disk

set collect_label — Label settings for collections

Description   Syntax   Option   Remarks and examples   Also see

Description

`set collect_label` controls the default labels used in tables created by `collect`. The default setting is `default`, which means that `collect` uses the labels defined in the file `label-default.stjson`. This file can be found in the ado-path.

Syntax

Use the system default labels in tables

`set collect_label default` [, `permanently`]

Specify a label set to be used as default labels in tables

`set collect_label` *label* [, `permanently`]

Option

`permanently` specifies that, in addition to making the change right now, the setting be remembered and become the default setting when you invoke Stata.

Remarks and examples

Remarks are presented under the following headings:

Overview
Labels for e-class results
Labels for r-class results
Labels for other results

Overview

`set collect_label` controls the default labels used in tables created by `collect`. The default setting is `default`, which means that `collect` uses the labels defined in the file `label-default.stjson`. This file can be found in the ado-path.

However, if you have a set of labels that you plan to use with many of the tables that you will be creating, you can save those labels to a file with `collect label save`. For example, you can save your labels under the filename `mylabels.stjson` by typing the following:

```
. collect label save mylabels.stjson
```

Then, to use these labels by default when creating tables with `collect`, you would type

```
. set collect_label mylabels
```

set collect_label will then search for label-mylabels.stjson in the ado-path. If label-mylabels.stjson is not found, it will search the ado-path for mylabels.stjson.

To see the current setting, type

```
. display c(collect_label)
```

In the following sections, we outline the logic that collect uses to determine the labels to be used when there is not a label for the result that was collected.

Labels for e-class results

When collecting an e-class result, which we will call e(*res*), collect will use the label from the collection that corresponds to that result. If that label does not exist, then a label is determined using the following steps:

1. If macro e(*res*__CL) exists, the label is pulled from this macro.
2. Search the system labels for a command-specific label attached to result *res*. If results are collected using the collect prefix, the prefixed command name is used. If results are collected using the collect get command, the command name is taken from macro e(cmd).
3. Search the system labels for a global (command-agnostic) label attached to result *res*.

Labels for r-class results

When collecting an r-class result, which we will call r(*res*), collect will use the label from the collection that corresponds to that result. If that label does not exist, then a label is determined using the following steps:

1. If macro r(*res*__CL) exists, the label is pulled from this macro.
2. Search the system labels for a command-specific label attached to result *res*. If results are collected using the collect prefix, the prefixed command name is used. If results are collected using the collect get command, the command name is taken from macro r(cmd).
3. Search the system labels for a global (command-agnostic) label attached to result *res*.

Labels for other results

When collecting results from other commands, collect will use the label from the collection if one exists. If there is no label for this result, then a label is determined by searching the system labels for a global (command-agnostic) label attached to that result.

Also see

[TABLES] **collect label** — Manage custom labels in a collection

set collect_style — Style settings for collections

Description

set collect_style controls the default styles used in tables created by collect. The default setting is default, which means that collect uses the styles defined in the file style-default.stjson. This file can be found in the ado-path.

Syntax

Use the system default styles in tables

set collect_style default [, permanently]

Specify a style set to be used as the default in tables

set collect_style *style* [, permanently]

Option

permanently specifies that, in addition to making the change right now, the setting be remembered and become the default setting when you invoke Stata.

Remarks and examples

set collect_style controls the default style used in tables created by collect. The default setting is default, which means that collect uses the styles defined in the file style-default.stjson. This file can be found in the ado-path.

However, if you have a style that you plan to use with many of the tables that you will be creating, you can save that style to a file with collect style save. For example, you can save your style as mystyle.stjson by typing the following:

```
. collect style save mystyle.stjson
```

Then, to use that style by default with tables created by collect, you would type

```
. set collect_style mystyle
```

set collect_style will then search for style-mystyle.stjson in the ado-path. If style-mystyle.stjson is not found, it will search the ado-path for mystyle.stjson.

To see the current setting, type

```
. display c(collect_style)
```

Also see

[TABLES] **collect style save** — Save collection styles to disk

Description

set collect_warn controls whether collect shows notes warning about unrecognized tags. The default setting is on, which means that collect subcommands will produce a note when they encounter a tag (dimension[level]) that is not present in a given collection. When set collect_warn is off, no such notes are produced.

Syntax

set collect_warn { on | off } [, permanently]

Option

permanently specifies that, in addition to making the change right now, the setting be remembered and become the default setting when you invoke Stata.

Remarks and examples

set collect_warn controls whether collect shows notes warning about unrecognized tags. By default, collect layout, collect style use, collect use, collect combine, and collect style cell produce a note when they encounter a tag that is not present in the collection. If you prefer not to see such notes, you can type

```
. set collect_warn off
```

This can be useful if, for instance, you use a style with collect style use before you have collected all your results. You may get warnings related to tags that will be created later.

If you wish to turn off all future warnings, you can type

```
. set collect_warn off, permanently
```

Whether you set the warnings to be off or to be on, you can control the warnings when you run one of the collect subcommands by specifying the warn or nowarn option.

To see the current setting, you can type

```
. display c(collect_warn)
```

set dtable_style — Default style settings for dtable

Description

set dtable_style controls the default styles used in tables created by dtable. The default setting is dtable, which means that dtable uses the styles defined in the file style-dtable.stjson. This file can be found in the ado-path.

Syntax

Use the system default styles in tables

set dtable_style dtable [, permanently]

Specify a style set to be used as the default in tables

set dtable_style *style* [, permanently]

Option

permanently specifies that, in addition to making the change right now, the setting be remembered and become the default setting when you invoke Stata.

Remarks and examples

set dtable_style controls the default style used in tables created by dtable. The default setting is dtable, which means dtable uses the styles defined in the file style-dtable.stjson. This file can be found in the ado-path.

However, if you have a style that you plan to use with many of the tables that you will be creating, you can save that style to a file with collect style save. For example, you can save your style as tabstyle.stjson by typing the following:

```
. collect style save tabstyle.stjson
```

Then, to use that style by default with tables created by dtable, you would type

```
. set dtable_style tabstyle
```

set dtable_style will then search for style-tabstyle.stjson in the ado-path. If style-tabstyle.stjson is not found, it will search the ado-path for tabstyle.stjson.

To see the current setting, type

```
. display c(dtable_style)
```

Also see

[R] **dtable** — Create a table of descriptive statistics

[TABLES] **collect style save** — Save collection styles to disk

set etable_style — Default style settings for etable

Description

`set etable_style` controls the default styles used in tables created by `etable`. The default setting is `etable`, which means that `etable` uses the styles defined in the file `style-etable.stjson`. This file can be found in the ado-path.

Syntax

Use the system default styles in tables

`set etable_style etable` [, permanently]

Specify a style set to be used as the default in tables

`set etable_style` *style* [, permanently]

Option

`permanently` specifies that, in addition to making the change right now, the setting be remembered and become the default setting when you invoke Stata.

Remarks and examples

`set etable_style` controls the default style used in tables created by `etable`. The default setting is `etable`, which means `etable` uses the styles defined in the file `style-etable.stjson`. This file can be found in the ado-path.

However, if you have a style that you plan to use with many of the tables that you will be creating, you can save that style to a file with `collect style save`. For example, you can save your style as `tabstyle.stjson` by typing the following:

```
. collect style save tabstyle.stjson
```

Then, to use that style by default with tables created by `etable`, you would type

```
. set etable_style tabstyle
```

`set etable_style` will then search for `style-tabstyle.stjson` in the ado-path. If `style-tabstyle.stjson` is not found, it will search the ado-path for `tabstyle.stjson`.

To see the current setting, type

```
. display c(etable_style)
```

Also see

[R] **etable** — Create a table of estimation results

[TABLES] **collect style save** — Save collection styles to disk

Description

set table_style controls the default styles used in tables created by table. The default setting is table, which means that table uses the styles defined in the file style-table.stjson. This file can be found in the ado-path.

Syntax

Use the system default styles in tables

set table_style table [, permanently]

Specify a style set to be used as the default in tables

set table_style *style* [, permanently]

Option

permanently specifies that, in addition to making the change right now, the setting be remembered and become the default setting when you invoke Stata.

Remarks and examples

set table_style controls the default style used in tables created by table. The default setting is table, which means table uses the styles defined in the file style-table.stjson. This file can be found in the ado-path.

However, if you have a style that you plan to use with many of the tables that you will be creating, you can save that style to a file with collect style save. For example, you can save your style as tabstyle.stjson by typing the following:

```
. collect style save tabstyle.stjson
```

Then, to use that style by default with tables created by table, you would type

```
. set table_style tabstyle
```

set table_style will then search for style-tabstyle.stjson in the ado-path. If style-tabstyle.stjson is not found, it will search the ado-path for tabstyle.stjson.

To see the current setting, type

```
. display c(table_style)
```

Also see

[R] **table** — Table of frequencies, summaries, and command results

[TABLES] **collect style save** — Save collection styles to disk

set tabulate_style — Default style settings for tabulate

Description Syntax Option Remarks and examples Also see

Description

set tabulate_style controls the default styles used in tables created by tabulate, tab1, tab2, tabi, and svy: tabulate. The default setting is tabulate, meaning that the styles defined in the style-tabulate.stjson file are used. This file can be found in the ado-path.

set tabulate_comps controls the default styles containing the composite results defined for two-way measures of association reported by tabulate, tab2, and tabi. The default setting is tabulate_composites, meaning that the composite results defined in the style-tabulate_composites.stjson file are used. This file can be found in the ado-path.

set svy_tab_comps controls the default styles containing the composite results defined for two-way tests of independence reported by svy: tabulate. The default setting is svy_tabulate_composites, meaning that the composite results defined in the style-svy_tabulate_composites.stjson file are used. This file can be found in the ado-path.

Syntax

Use the system default styles in tabulation tables

set tabulate_style tabulate [, permanently]

set tabulate_comps tabulate_composites [, permanently]

set svy_tab_comps svy_tabulate_composites [, permanently]

Specify a style set to be used as the default in tabulation tables

set tabulate_style *style* [, permanently]

Specify a style defining the default composite results for measures of association to be used in two-way tabulation tables

set tabulate_comps *style* [, permanently]

Specify a style defining the default composite results for tests of independence to be used in two-way tabulation tables for survey data

set svy_tab_comps *style* [, permanently]

Option

permanently specifies that, in addition to making the change right now, the setting be remembered and become the default setting when you invoke Stata.

Remarks and examples

`set tabulate_style` controls the default style used in tables created by tabulate for one-way tabulations, `tab1`, tabulate for two-way tabulations, `tab2`, `tabi`, tabulate, summarize(), svy: tabulate for one-way tabulations, and svy: tabulate for two-way tabulations. The default setting is `tabulate`, which means these commands use the styles defined in the file `style-tabulate.stjson`. This file can be found in the ado-path.

However, if you have a style that you plan to use with many tabulation tables that you will be creating, you can save that style to a file with collect style save. For example, you can save your style as `tabstyle.stjson` by typing the following:

```
. collect style save tabstyle.stjson
```

Then, to use that style by default with tables created by `tabulate`, you would type

```
. set tabulate_style tabstyle
```

`set tabulate_style` will then search for `style-tabstyle.stjson` in the ado-path. If `style-tabstyle.stjson` is not found, it will search the ado-path for `tabstyle.stjson`.

To see the current setting, type

```
. display c(tabulate_style)
```

`set tabulate_comps` controls the styles that define composite results for two-way tables produced by `tabulate`. See [R] **tabulate twoway** for details on these composite results and their default definitions. To see the current setting, type

```
. display c(tabulate_comps)
```

`set svy_tab_comps` controls the styles that define composite results for two-way tables produced by `svy: tabulate`. See [SVY] **svy: tabulate twoway** for details on these composite results and their default definitions. To see the current setting, type

```
. display c(svy_tab_comps)
```

Also see

[R] **tabulate oneway** — One-way table of frequencies

[R] **tabulate twoway** — Two-way table of frequencies

[R] **tabulate, summarize()** — One- and two-way tables of summary statistics

[SVY] **svy: tabulate oneway** — One-way tables for survey data

[SVY] **svy: tabulate twoway** — Two-way tables for survey data

[TABLES] **collect style save** — Save collection styles to disk

Description Remarks and examples Reference Also see

Description

In this example, we demonstrate how to use `table` to compute means and standard deviations, run the `pwcorr` command to obtain a correlation matrix, and create a table with all of these statistics. We also demonstrate how you can use `collect` to customize this table.

Remarks and examples

Remarks are presented under the following headings:

Table of correlations

Below, we use data from the Second National Health and Nutrition Examination Survey (NHANES II) (McDowell et al. 1981). We want to create a table to explore the correlation between `age`, `weight`, and systolic blood pressure (`bpsystol`).

Let's begin by looking at the correlations with the `pwcorr` command, which allows us to report the significance levels along with the correlation coefficients.

```
. use https://www.stata-press.com/data/r19/nhanes2l
(Second National Health and Nutrition Examination Survey)

. pwcorr age weight bpsystol, sig

             |      age   weight bpsystol
-------------+---------------------------
         age |   1.0000 
             |
             |
      weight |   0.0388   1.0000 
             |   0.0001
             |
    bpsystol |   0.4811   0.2861   1.0000 
             |   0.0000   0.0000
             |
```

pwcorr stores the correlations and significance levels in the matrices r(C) and r(sig), respectively. Below, we create a matrix called vech, which contains the lower triangle elements of the matrix r(C). Then, we list both matrices:

```
. matrix define vech = vech(r(C))
. matrix list r(C)
symmetric r(C)[3,3]
                age     weight   bpsystol
     age          1
  weight  .03881324          1
bpsystol  .48110277  .28607421          1
. matrix list vech
vech[6,1]
                          c1
          age:age          1
       age:weight  .03881324
     age:bpsystol  .48110277
    weight:weight          1
  weight:bpsystol  .28607421
bpsystol:bpsystol          1
```

We can see that vech() creates a column vector by stacking elements from the first column of r(C), then the second column, and finally the third column. The variable before the colon is the row equation, and the variable after the colon is the row name.

To create our table, we can use collect get to collect the column vectors of both the correlations matrix and the significance levels matrix. We then use collect layout to lay out our table. We place the results for each row name on the rows, by interacting rowname and result, and the row equations (roweq) on the columns:

```
. collect get corr=vech(r(C)) sig=vech(r(sig))
. collect layout (rowname#result) (roweq)
Collection: default
      Rows: rowname#result
   Columns: roweq
   Table 1: 9 x 3
```

	Age (years)	Weight (kg)	Systolic blood pressure
Age (years)			
corr	1		
sig	.		
Weight (kg)			
corr	.0388132	1	
sig	.0000782	.	
Systolic blood pressure			
corr	.4811028	.2860742	1
sig	0	3.7e-194	.

This is a good starting point, but instead of displaying the significance levels, we would like to display stars representing those levels. We use collect stars to display three stars for values of sig less than 0.01, two stars for values less than 0.05, and one star for values less than 0.1. The attach option tells Stata to attach these stars to the correlations (corr), and shownote adds the note explaining the significance levels that the stars represent.

Additionally, rather than displaying a value of 1 for the correlation of a variable with itself, we want to display a dash. We use `collect style cell` to specify that values of `corr` beyond 0.99 should be labeled with a dash. We also format the correlations to four decimal places and center them horizontally. Whereas before we included all results, now we update our table layout to include only the correlations (`result[corr]`):

```
. collect stars sig 0.01 "***" 0.05 "**" 0.1 "*", attach(corr) shownote
. collect style cell result[corr], maximum(0.99, label("-")) nformat(%6.4f)
> halign(center)
. collect layout (rowname#result[corr]) (roweq)
Collection: default
      Rows: rowname#result[corr]
   Columns: roweq
   Table 1: 6 x 3
```

	Age (years)	Weight (kg)	Systolic blood pressure
Age (years)			
corr	-		
Weight (kg)			
corr	0.0388***	-	
Systolic blood pressure			
corr	0.4811***	0.2861***	-

```
*** p<.01, ** p<.05, * p<.1
```

This table looks much better, but we can customize it further. Because we are displaying only correlations, we can hide the labels for the results with `collect style header`. Additionally, we will remove the border on the right side of the corner and row-header sections of the table. The dimension `border_block` divides the table into four sections: `corner`, `column-header`, `row-header`, and `item`. This dimension allows us to modify borders for a whole section of the table. Finally, we add a descriptive title with `collect title` and preview our table with `collect preview`:

```
. collect style header result, level(hide)
. collect style cell border_block[corner row-header], border(right, pattern(nil))
. collect title "Table of correlations"
. collect preview
Table of correlations
```

	Age (years)	Weight (kg)	Systolic blood pressure
Age (years)	-		
Weight (kg)	0.0388***	-	
Systolic blood pressure	0.4811***	0.2861***	-

```
*** p<.01, ** p<.05, * p<.1
```

We can now export our table in our preferred style—Word, PDF, HTML, LaTeX, Excel, or Markdown—using `collect export`.

Table of correlations, means, and standard deviations

Building on our last table, suppose we also want to report the means and standard deviations for each variable. We previously created a column vector with the correlations and significance levels, but for this table, we will create a row vector with those values. The reason is that now we are going to collect other statistics for our table, and the `collect` system uses the dimensions `coleq` (column equation) and `colname` (column name) to identify those results. To align our results, we will need to reference our correlations using column equations and names as well.

To see how this works, let's create another collection for our new table and run our `pwcorr` command again. We will create a row vector of the correlations by transposing (`'`) the column vector of correlations (`vech(r(C))`):

```
. collect create corr2
(current collection is corr2)

. pwcorr age weight bpsystol, sig

             |      age   weight bpsystol
-------------+---------------------------
         age |   1.0000 
             |
             |
      weight |   0.0388   1.0000 
             |   0.0001
             |
    bpsystol |   0.4811   0.2861   1.0000 
             |   0.0000   0.0000
             |

. matrix define vechrow = vech(r(C))'

. matrix list r(C)

symmetric r(C)[3,3]
                age     weight   bpsystol
     age          1
  weight  .03881324          1
bpsystol  .48110277  .28607421          1

. matrix list vechrow

vechrow[1,6]
          age:        age:        age:     weight:     weight:   bpsystol:
          age      weight    bpsystol      weight    bpsystol    bpsystol
c1          1   .03881324   .48110277           1   .28607421           1
```

Now, we can identify the correlations by referring to the column equation and column name. We are now ready to create our table.

Previously, we used `collect get` to collect our results, but with the `table` command, we can compute summary statistics and incorporate the results from another Stata command in the same table. This versatility will allow us to compute all our statistics with a single `table` command. With the `statistic()` option, we request the means and standard deviations of the three variables mentioned. With the `command()` option, we execute the `pwcorr` command. We also format the results to display only two digits to the right of the decimal. The arguments before the comma specify how we want to arrange our results. We place the summary statistics (`result`) and column equations on the rows and the column names of the matrix (`colname`) on the columns. We also modified the variable label for `bpsystol` to prevent the table from wrapping.

```
. label variable bpsystol "BP"
. table (result coleq) (colname),
> statistic(mean age weight bpsystol)
> statistic(sd age weight bpsystol)
> command(corr=(vech(r(C))') sig=(vech(r(sig))'):
> pwcorr age weight bpsystol, sig) nformat(%5.2f mean sd)
```

	Age (years)	Weight (kg)	BP
Mean			
Mean	47.58	71.90	130.88
Standard deviation			
Standard deviation	17.21	15.36	23.33
corr			
Age (years)			
pwcorr age weight bpsystol, sig	1	.0388132	.4811028
Weight (kg)			
pwcorr age weight bpsystol, sig		1	.2860742
BP			
pwcorr age weight bpsystol, sig			1
sig			
Age (years)			
pwcorr age weight bpsystol, sig	.	.0000782	0
Weight (kg)			
pwcorr age weight bpsystol, sig		.	3.7e-194
BP			
pwcorr age weight bpsystol, sig			.

We will use the maximum() option with collect style cell to display a dash for correlations between a variable and itself. And we will display stars for levels of significance, as we did with our previous table. Then, we can lay out our table with the variables (colname) on the rows and the results on the columns. The means and standard deviations can be identified by levels of colname and result. However, the correlations are identified by the value of the column name, column equation, and result. Therefore, we interact coleq with the level corr of result in our table layout.

```
. collect style cell result[corr],
>  maximum(0.99, label("  -  ")) nformat(%6.4f) halign(center)
. collect stars sig 0.01 "***" 0.05 "**" 0.1 "*", attach(corr) shownote
. collect layout (colname) (result[mean sd] coleq#result[corr])
Collection: Table
      Rows: colname
   Columns: result[mean sd] coleq#result[corr]
   Table 1: 3 x 5
```

	Mean	Standard deviation	Age (years) corr	Weight (kg) corr	BP corr
Age (years)	47.58	17.21	-		
Weight (kg)	71.90	15.36	0.0388***	-	
BP	130.88	23.33	0.4811***	0.2861***	-

```
*** p<.01, ** p<.05, * p<.1
```

We now have the layout we want, but we will make a few modifications to polish the table further. First, we will hide the label "corr" with collect style header. Then, we will shorten the label for the standard deviations with collect label levels. Rather than repeating the variable labels on the rows and columns, we want to create an index for the variables and use those numbers in the column headers. We can do this by simply modifying the labels for the levels of `colname` and `coleq`. Then, we preview our table:

```
. collect style header result[corr], level(hide)
. collect label levels result sd "SD", modify
. collect label levels colname age "1. Age" weight "2. Weight"
> bpsystol "3. Systolic BP", modify
. collect label levels coleq age "1" weight "2" bpsystol "3", modify
. collect preview
```

	Mean	SD	1	2	3
1. Age	47.58	17.21	-		
2. Weight	71.90	15.36	0.0388***	-	
3. Systolic BP	130.88	23.33	0.4811***	0.2861***	-

```
*** p<.01, ** p<.05, * p<.1
```

Finally, we will center the column headers for the column equations, remove the borders on the right side of the corner and row headers, and provide a title for the table. Both `border_block` and `cell_type` divide the table into four sections; the former is used to modify borders, and the latter is used to modify all other appearance styles for the cells. After we make these changes, we preview our table for the last time.

```
. collect style cell cell_type[column-header]#coleq, halign(center)
. collect style cell border_block[corner row-header], border(right, pattern(nil))
. collect title "Descriptive statistics and correlations"
. collect preview
Descriptive statistics and correlations
```

	Mean	SD	1	2	3
1. Age	47.58	17.21	-		
2. Weight	71.90	15.36	0.0388***	-	
3. Systolic BP	130.88	23.33	0.4811***	0.2861***	-

```
*** p<.01, ** p<.05, * p<.1
```

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

[R] **table** — Table of frequencies, summaries, and command results

[TABLES] **collect style column** — Collection styles for column headers

[TABLES] **collect style header** — Collection styles for hiding and showing header components

Example 2 — Table of medians and rank-sum test results

Description Remarks and examples Reference Also see

Description

In this example, we demonstrate how to use `table` to compute medians and store them in a collection. We also use `collect` to store the results of rank-sum tests in the collection and then create a customized table combining the results.

Remarks and examples

Remarks are presented under the following headings:

Computing and collecting statistics

Below, we use data from the Second National Health and Nutrition Examination Survey (NHANES II) (McDowell et al. 1981). We wish to compute the median `age`, `weight`, systolic blood pressure (`bpsystol`), cholesterol, and iron for individuals who have diabetes and those who do not. We use the `table` command to compute these statistics. The first set of parentheses places the variables on the rows of the table, and the second set places the levels of `diabetes` on the columns. By default, `table` will display the table and store the results in a collection called `Table`. Also by default, `table` will report the statistics for each group, in our case diabetics and nondiabetics, and for the full dataset. We use `nototals` to suppress those medians for the full dataset.

```
. use https://www.stata-press.com/data/r19/nhanes2l
(Second National Health and Nutrition Examination Survey)
. table (var) (diabetes),
> statistic(median age weight bpsystol tcresult iron) nototals
```

	Diabetes status	
	Not diabetic	Diabetic
Age (years)	48	64
Weight (kg)	70.19	74.84
Systolic blood pressure	128	142
Serum cholesterol (mg/dL)	212	223
Serum iron (mcg/dL)	96	88

We would also like to perform a rank-sum test for each of those variables to test whether the distributions are the same across the categories of `diabetes`. If we wanted to perform the test only for `age`, we could type

```
. ranksum age, by(diabetes)
```

Because we want to perform the test for multiple variables, we write a loop to issue the `ranksum` command for each variable. We use the `collect` prefix to collect the two-sided p-value (`r(p)`). The `tag()` option tags the results with the dimension `var`, which will allow us to align these results with the medians we computed above.

```
. foreach x in age weight bpsystol tcresult iron {
  2.         quietly: collect r(p), tag(var[‘x’]): ranksum ‘x’, by(diabetes)
  3. }
```

We want to create a table with the medians we computed with `table` and the p-values we collected with the `collect` prefix. `collect` stored the results in the current collection, so we have the results all in one place. Now, we can use `collect layout` to arrange the items from the collection into a table. Again, we place the variables on the rows and the levels of `diabetes` and the statistics from `ranksum` on the columns.

```
. collect layout (var) (diabetes result)

Collection: Table
      Rows: var
   Columns: diabetes result
   Table 1: 5 x 3
```

(*output omitted*)

We omit the table preview here because of the table's width.

Customizing the table

The table above is wide because of the long label for the p-values. We can see the labels by using the `collect label list` command with the `result` dimension.

```
. collect label list result

  Collection: Table
   Dimension: result
       Label: Result
Level labels:
           N  Sample size
         N_1  Sample size of first group
         N_2  Sample size of second group
       Var_a  Adjusted variance
      group1  Value of variable for first group
      median  Median
           p  Two-sided p-value from normal approximation
         p_l  Lower one-sided p-value from normal approximation
         p_u  Upper one-sided p-value from normal approximation
     sum_exp  Expected sum of ranks for first group
     sum_obs  Observed sum of ranks for first group
           z  Z statistic
```

The p-values correspond to the level `p` of the dimension `result`. Below, we modify this label with `collect label levels`. Then, we preview our table:

```
. collect label levels result p "p-value", modify

. collect preview
```

	Diabetes status		p-value
	Not diabetic	Diabetic	
Age (years)	48	64	9.33e-69
Weight (kg)	70.19	74.84	1.12e-10
Systolic blood pressure	128	142	3.61e-43
Serum cholesterol (mg/dL)	212	223	.0000178
Serum iron (mcg/dL)	96	88	2.17e-08

Because labels for the levels of diabetes are descriptive enough, we can hide the title for the dimension. We format the p-values to have three decimal places. We also remove the vertical border. Then, we preview our table once more:

```
. collect style header diabetes, title(hide)
. collect style cell result[p], nformat(%5.3f)
. collect style cell border_block, border(right, pattern(nil))
. collect preview
```

	Not diabetic	Diabetic	p-value
Age (years)	48	64	0.000
Weight (kg)	70.19	74.84	0.000
Systolic blood pressure	128	142	0.000
Serum cholesterol (mg/dL)	212	223	0.000
Serum iron (mcg/dL)	96	88	0.000

In fact, we prefer not to report our p-values in this form. Instead, we can display them as being less than 0.001. With collect style cell, we can specify a minimum value, and any p-values smaller than the minimum will be displayed as simply less than that minimum:

```
. collect style cell result[p], minimum(0.001)
. collect preview
```

	Not diabetic	Diabetic	p-value
Age (years)	48	64	<0.001
Weight (kg)	70.19	74.84	<0.001
Systolic blood pressure	128	142	<0.001
Serum cholesterol (mg/dL)	212	223	<0.001
Serum iron (mcg/dL)	96	88	<0.001

See [TABLES] **collect style header** and [TABLES] **collect style cell** for more information on the commands we used here to customize 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

[R] **table** — Table of frequencies, summaries, and command results

[TABLES] **collect get** — Collect results from a Stata command

Description Remarks and examples Reference Also see

Description

In this example, we demonstrate how to use `table` to compute summary statistics for levels of a categorical variable and store them in a collection. We also demonstrate how to use the `collect` suite of commands to create a customized table with these results.

Remarks and examples

Remarks are presented under the following headings:

Computing statistics with the table command

Below, we use data from the Second National Health and Nutrition Examination Survey (NHANES II) (McDowell et al. 1981). We want to create a table to compare summary statistics for males and females.

With the `table` command, we can compute several types of summary statistics. Below, we use the `statistic()` option to compute the `mean` and standard deviation (`sd`) of `age`, body mass index (`bmi`), and systolic blood pressure (`bpsystol`) for each category of `sex`. We place our variables (`var`) on the rows and the levels of `sex` on the columns. Additionally, we format the means and standard deviations to display only two digits to the right of the decimal.

```
. use https://www.stata-press.com/data/r19/nhanes2l
(Second National Health and Nutrition Examination Survey)

. table (var) (sex),
>     statistic(mean age bmi bpsystol)
>     statistic(sd age bmi bpsystol)
>     nformat(%6.2f)
```

	Sex		
	Male	Female	Total
Age (years)			
Mean	47.42	47.72	47.58
Standard deviation	17.17	17.26	17.21
Body mass index (BMI)			
Mean	25.51	25.56	25.54
Standard deviation	4.02	5.60	4.91
Systolic blood pressure			
Mean	132.89	129.07	130.88
Standard deviation	20.99	25.13	23.33

The table above reports summary statistics for continuous variables. We might also want to incorporate statistics for categorical variables. For instance, let's report frequencies and percentages for the levels of `diabetes` and `hlthstat`. The statistic `fvfrequency` provides the frequency for each level of a categorical variable, and `fvpercent` reports the percentage of observations in each category. We still want to format our means and standard deviations but not our other statistics. With `nformat()`, we can specify the statistics to which we want to apply the format.

```
. table (var) (sex),
>     statistic(fvfrequency diabetes) statistic(fvpercent diabetes)
>     statistic(mean age bmi) statistic(sd age bmi)
>     statistic(fvfrequency hlthstat) statistic(fvpercent hlthstat)
>     statistic(mean bpsystol) statistic(sd bpsystol)
>     nformat(%6.2f mean sd)
```

	Sex		
	Male	Female	Total
Diabetes status=Not diabetic			
Factor-variable frequency	4,698	5,152	9,850
Factor-variable percent	95.58	94.81	95.18
Diabetes status=Diabetic			
Factor-variable frequency	217	282	499
Factor-variable percent	4.42	5.19	4.82
Age (years)			
Mean	47.42	47.72	47.58
Standard deviation	17.17	17.26	17.21
Body mass index (BMI)			
Mean	25.51	25.56	25.54
Standard deviation	4.02	5.60	4.91
Health status=Excellent			
Factor-variable frequency	1,252	1,155	2,407
Factor-variable percent	25.50	21.29	23.29
Health status=Very good			
Factor-variable frequency	1,213	1,378	2,591
Factor-variable percent	24.71	25.40	25.07
Health status=Good			
Factor-variable frequency	1,340	1,598	2,938
Factor-variable percent	27.30	29.45	28.43
Health status=Fair			
Factor-variable frequency	722	948	1,670
Factor-variable percent	14.71	17.47	16.16
Health status=Poor			
Factor-variable frequency	382	347	729
Factor-variable percent	7.78	6.40	7.05
Systolic blood pressure			
Mean	132.89	129.07	130.88
Standard deviation	20.99	25.13	23.33

We now have a table with summary statistics for males and females in our data. However, we likely want to polish the table so that the labels are not distracting.

Customizing the table

By default, `table` will display the table and store the results in a collection called `Table`. We can now use the `collect` suite of commands to work with this collection and modify the look of the table.

To get started, note that the statistics are stored as levels of the dimension `result`. We can see the levels of this dimension by using `collect levelsof`. We will use the names of the dimension and its levels in the `collect` subcommands that we use to modify our table.

```
. collect levelsof result

Collection: Table
 Dimension: result
    Levels: fvfrequency fvpercent mean sd
```

First, let's remove the labels for the statistics in the row headers. We can use `collect style header` to hide the level labels for the dimension `result`. Then, we preview our table with `collect preview`.

```
. collect style header result, level(hide)

. collect preview
```

	Sex		
	Male	Female	Total
Diabetes status=Not diabetic	4,698	5,152	9,850
	95.58	94.81	95.18
Diabetes status=Diabetic	217	282	499
	4.42	5.19	4.82
Age (years)	47.42	47.72	47.58
	17.17	17.26	17.21
Body mass index (BMI)	25.51	25.56	25.54
	4.02	5.60	4.91
Health status=Excellent	1,252	1,155	2,407
	25.50	21.29	23.29
Health status=Very good	1,213	1,378	2,591
	24.71	25.40	25.07
Health status=Good	1,340	1,598	2,938
	27.30	29.45	28.43
Health status=Fair	722	948	1,670
	14.71	17.47	16.16
Health status=Poor	382	347	729
	7.78	6.40	7.05
Systolic blood pressure	132.89	129.07	130.88
	20.99	25.13	23.33

The variable labels and value labels for our categorical variables are bound by an equal sign. Instead of repeating the variable labels, we can use `collect style row stack` to list each one only once and stack these headers in a single column. We also specify the `spacer` option to insert a blank line between row dimensions. Finally, we can remove the border on the right side of the row headers by setting the border pattern to `nil`. We then preview our table once more.

```
. collect style row stack, nobinder spacer
. collect style cell border_block, border(right, pattern(nil))
. collect preview
```

	Sex Male	Female	Total
Diabetes status			
Not diabetic	4,698	5,152	9,850
	95.58	94.81	95.18
Diabetic	217	282	499
	4.42	5.19	4.82
Age (years)	47.42	47.72	47.58
	17.17	17.26	17.21
Body mass index (BMI)	25.51	25.56	25.54
	4.02	5.60	4.91
Health status			
Excellent	1,252	1,155	2,407
	25.50	21.29	23.29
Very good	1,213	1,378	2,591
	24.71	25.40	25.07
Good	1,340	1,598	2,938
	27.30	29.45	28.43
Fair	722	948	1,670
	14.71	17.47	16.16
Poor	382	347	729
	7.78	6.40	7.05
Systolic blood pressure	132.89	129.07	130.88
	20.99	25.13	23.33

This layout is one nice way to compare the summary statistics. We could continue to modify its style to finalize our table.

However, we may also want to consider another layout—one in which the means of continuous variables and frequencies of categorical variables are in one column and the standard deviations of continuous variables and percentages for categorical variables are in another column.

Currently, the frequencies for the categorical variables are tagged with the level `fvfrequency` of the `result` dimension, and the percentages are tagged with level `fvpercent` of the `result` dimension. To align the frequencies with the means and the percentages with the standard deviations, we recode them to the levels `mean` and `sd` of the same dimension. Then, we lay out our table with the variables on the rows and the results for males and females on the columns. Note that by typing `sex[1 2]`, we specify that only levels `1` and `2` of the `sex` dimension be included. This allows us to omit the statistics that `table` computed for all observations in the data and that would be included if we simply include the dimension `sex`.

```
. collect recode result fvfrequency=mean  fvpercent=sd
(42 items recoded in collection Table)
. collect layout (var) (sex[1 2]#result)
Collection: Table
      Rows: var
   Columns: sex[1 2]#result
   Table 1: 15 x 4
```

	Sex			
	Male		Female	
Diabetes status				
Not diabetic	4698.00	95.58	5152.00	94.81
Diabetic	217.00	4.42	282.00	5.19
Age (years)	47.42	17.17	47.72	17.26
Body mass index (BMI)	25.51	4.02	25.56	5.60
Health status				
Excellent	1252.00	25.50	1155.00	21.29
Very good	1213.00	24.71	1378.00	25.40
Good	1340.00	27.30	1598.00	29.45
Fair	722.00	14.71	948.00	17.47
Poor	382.00	7.78	347.00	6.40
Systolic blood pressure	132.89	20.99	129.07	25.13

Now, we can finish customizing this table by adding percent signs to the percentages, enclosing our standard deviations in parentheses, and fixing the numeric formatting. (Now that the frequencies are part of the level `mean`, they have the numeric format that we applied earlier to that level.)

We can use `collect style cell` to modify all cells in the table or specific cells.

First, we add a percent sign to our percentages. Because we recoded the percentages to the `sd` level of `result`, we will need to refer to them with the tag `result[sd]`. However, this is not enough. If we refer to only `result[sd]`, we will refer to both standard deviations and percentages. To apply a change only to our categorical variables, we type `result[sd]#var[i.diabetes i.hlthstat]`. By interacting these two tags, we reference only values that are tagged with the `sd` level of `result` as well as the levels of either `i.diabetes` or `i.hlthstat` of `var`.

The option `sformat()` changes the string format, and `%s` refers to the numeric value. The text will be placed around our numeric values in the table as we place it around `%s` in this option. Adding a percent sign requires a special character, `%%`.

Similarly, we can type `result[sd]#var[age bmi bpsystol]` to refer to the standard deviations of our continuous variables. We enclose these values in parentheses.

```
. collect style cell result[sd]#var[i.diabetes i.hlthstat], sformat("%s%%")
. collect style cell result[sd]#var[age bmi bpsystol], sformat("(%s)")
```

Last, we do not want to display any digits to the right of the decimal for the frequencies. So we use `collect style cell` with the `nformat()` option for the frequencies (tagged with `mean` of the `result` dimension and of the levels `hlthstat` and `diabetes` of the `var` dimension).

```
. collect style cell result[mean]#var[i.diabetes i.hlthstat], nformat(%4.0f)
. collect preview
```

	Sex			
	Male		Female	
Diabetes status				
Not diabetic	4698	95.58%	5152	94.81%
Diabetic	217	4.42%	282	5.19%
Age (years)	47.42	(17.17)	47.72	(17.26)
Body mass index (BMI)	25.51	(4.02)	25.56	(5.60)
Health status				
Excellent	1252	25.50%	1155	21.29%
Very good	1213	24.71%	1378	25.40%
Good	1340	27.30%	1598	29.45%
Fair	722	14.71%	948	17.47%
Poor	382	7.78%	347	6.40%
Systolic blood pressure	132.89	(20.99)	129.07	(25.13)

Our final table is much neater and easier to read.

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

[TABLES] **collect recode** — Recode dimension levels in a collection

[R] **table** — Table of frequencies, summaries, and command results

Description Remarks and examples Reference Also see

Description

In this example, we demonstrate how to use `collect` to store the results of mean-comparison tests (t tests) for levels of a categorical variable in a collection and how to create a customized table with these results.

Remarks and examples

Remarks are presented under the following headings:

Collecting statistics

Below, we use data from the Second National Health and Nutrition Examination Survey (NHANES II) (McDowell et al. 1981). We wish to test whether the mean systolic blood pressure (`bpsystol`) is the same across males and females in each category of race. To perform the test for each level of `race`, we use the `by` prefix. We first create a new collection named `ex4` and then use the `collect` prefix to collect the results from each `ttest` command and store them in the new collection. All results that `ttest` returns in `r()` will be collected, but only the ones we have specified will be automatically included in our table.

```
. use https://www.stata-press.com/data/r19/nhanes2l
(Second National Health and Nutrition Examination Survey)
. collect create ex4
(current collection is ex4)
. quietly: collect r(N_1) r(mu_1) r(N_2) r(mu_2) r(p):
> by race, sort: ttest bpsystol, by(sex)
```

These results are stored in the current collection. We can then use `collect layout` to arrange the items from the collection into a table. We place the levels of `race` on the rows and the results (`result`) on the columns.

```
. collect layout (race) (result)
Collection: ex4
      Rows: race
   Columns: result
   Table 1: 3 x 5
```

(*output omitted*)

The labels for these statistics are automatically included in the table, which makes it very wide. Therefore, we omit the table preview from the output. In the following section, we will format the table to make it ready for publication.

Customizing the table

To finalize our table from the previous section, we will want to label which statistics are for males and females, shorten the labels for the statistics, and display the results with two digits to the right of the decimal.

First, let's work on the labels. The statistics are part of the dimension `result`. We list the labels for the levels of this dimension:

```
. collect label list result

  Collection: ex4
   Dimension: result
       Label: Result
Level labels:
         N_1  Sample size n1
         N_2  Sample size n2
        df_t  Degrees of freedom
       level  Confidence level
        mu_1  x1 mean for population 1
        mu_2  x2 mean for population 2
           p  Two-sided p-value
         p_l  Lower one-sided p-value
         p_u  Upper one-sided p-value
          sd  Combined std. dev.
        sd_1  Standard deviation for first variable
        sd_2  Standard deviation for second variable
          se  Std. error
           t  t statistic
```

We would like to remap the statistics for males to their own dimension and similarly for females. This will allow us to categorize the results under the labels `Males` and `Females`. The levels `N_1` and `mu_1` correspond to males, and the levels `N_2` and `mu_2` correspond to females. We also remap the p-values to their own dimension called `Difference`.

```
. collect remap result[N_1 mu_1] = Males
(6 items remapped in collection ex4)

. collect remap result[N_2 mu_2] = Females
(6 items remapped in collection ex4)

. collect remap result[p] = Difference
(3 items remapped in collection ex4)
```

Then, we use `collect style header` to specify that we want to display the title for the specified dimensions. These titles are suppressed by default. Then, we arrange our items once more with the new dimension names. Again, we place the levels of `race` on the rows, but now we place the dimensions `Males`, `Females`, and `Difference` on the columns.

```
. collect style header Males Females Difference, title(name)
. collect layout (race) (Males Females Difference)
Collection: ex4
      Rows: race
   Columns: Males Females Difference
   Table 1: 3 x 5
```

	Males N_1	Males mu_1	Females N_2	Females mu_2	Difference p
White	4312	132.8476	4753	128.5264	1.78e-19
Black	500	133.69	586	133.8481	.9217363
Other	103	130.6699	97	126.7216	.3098674

Our table looks much better. Next, we will add labels to the statistics. The statistics are levels of the new dimensions that we remapped them to. To modify labels for levels of a dimension, we use collect label levels.

```
. collect label levels Males N_1 "N" mu_1 "Mean BP"
. collect label levels Females N_2 "N" mu_2 "Mean BP"
. collect label levels Difference p "p-value"
```

Previously, we saw the column headers Males and Females being repeated. We would like to display these only once and center them horizontally. We can use collect style column to make this change. We also set the columns to have the same width. Then, we center-align all the cells in the table. With collect style cell, we can modify all cells in the table or specific cells. For example, we wish to format the means and p-values to display two digits to the right of the decimal. Therefore, we specify the levels of the dimensions we want to apply this format to. Then, we get a preview of our table.

```
. collect style column, dups(center) width(equal)
. collect style cell, halign(center)
. collect style cell Males[mu_1] Females[mu_2] Difference[p], nformat(%5.2f)
. collect preview
```

	Males		Females		Difference
	N	Mean BP	N	Mean BP	p-value
White	4312	132.85	4753	128.53	0.00
Black	500	133.69	586	133.85	0.92
Other	103	130.67	97	126.72	0.31

Finally, we will modify the borders in the table by using collect style cell. First, we remove the vertical border. Because we do not want any vertical borders, we do not list any levels of the dimension border_block when we specify the border(right, pattern(nil)) option. Our next collect style cell command requires a bit more explanation. With it, we add a horizontal border below Males to indicate that the first N and Mean BP are for males. To target this very specific border, we specify cell_type[column-header]#Males. Here cell_type refers to cells in different parts of the table. We want to make a change only in the column header. We also want to make this change only for the Males dimension. By specifying the # between the tags, we direct the change only at the dimension Male within the column headers. We can also target the border under Females by specifying cell_type[column-header]#Females. To this command, we add the border(bottom, pattern(single)) option to place a single border on the bottom of these cells.

```
. collect style cell border_block, border(right, pattern(nil))

. collect style cell cell_type[column-header]#Males
> cell_type[column-header]#Females, border(bottom, pattern(single))

. collect preview
```

	Males		Females		Difference
	N	Mean BP	N	Mean BP	p-value
White	4312	132.85	4753	128.53	0.00
Black	500	133.69	586	133.85	0.92
Other	103	130.67	97	126.72	0.31

After finalizing our table of results, we can export it to another format with `collect export`.

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

[TABLES] **collect remap** — Remap tags in a collection

[TABLES] **collect style column** — Collection styles for column headers

[TABLES] **collect style header** — Collection styles for hiding and showing header components

Description   Remarks and examples   Reference   Also see

Description

In this example, we demonstrate how to collect results from a regression and create a table of coefficients and confidence intervals. We also show how to customize the resulting table.

Remarks and examples

Remarks are presented under the following headings:

Collecting regression results and creating a table

Below, we use data from the Second National Health and Nutrition Examination Survey (NHANES II) (McDowell et al. 1981). We fit a model for systolic blood pressure (`bpsystol`) as a function of `age`, `weight`, and the region of the country the individual resides in. We first create a new collection named `ex5` and then use the `collect` prefix to collect the coefficients (`_r_b`) and confidence intervals (`_r_ci`) into the this collection.

```
. use https://www.stata-press.com/data/r19/nhanes2l
(Second National Health and Nutrition Examination Survey)

. collect create ex5
(current collection is ex5)

. collect _r_b _r_ci: regress bpsystol age weight i.region

      Source |       SS           df       MS      Number of obs   =    10,351
-------------+----------------------------------   F(5, 10345)     =    900.55
       Model |  1708779.02         5  341755.804   Prob > F        =    0.0000
    Residual |     3925891    10,345  379.496472   R-squared       =    0.3033
-------------+----------------------------------   Adj R-squared   =    0.3029
       Total |  5634670.03    10,350  544.412563   Root MSE        =    19.481

------------------------------------------------------------------------------
    bpsystol | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
         age |   .6383029   .0111397    57.30   0.000     .6164668    .6601389
      weight |   .4069294   .0124796    32.61   0.000      .382467    .4313917
             |
      region |
         MW  |  -.2397311   .5640029    -0.43   0.671    -1.345286    .8658237
          S  |  -.6187414   .5604584    -1.10   0.270    -1.717348    .4798654
          W  |  -.8617777    .570496    -1.51   0.131     -1.98006    .2565047
             |
       _cons |   71.70779   1.107732    64.73   0.000     69.53642    73.87916
------------------------------------------------------------------------------
```

In fact, all visible `e()` results are collected from our regression and stored as levels of the dimension `result`. But by specifying those two results, we have set them to be automatically included in a table when we include the dimension `result`.

Now, we can use `collect layout` to arrange the results into a table. We place the covariate names (`colname`) on the rows and the statistics (`result`) on the columns:

```
. collect layout (colname) (result)

Collection: ex5
      Rows: colname
   Columns: result
   Table 1: 7 x 2
```

	Coefficient	95% CI	
Age (years)	.6383029	.6164668	.6601389
Weight (kg)	.4069294	.382467	.4313917
NE	0		
MW	-.2397311	-1.345286	.8658237
S	-.6187414	-1.717348	.4798654
W	-.8617777	-1.98006	.2565047
Intercept	71.70779	69.53642	73.87916

Notice that the statistics are labeled and that the variable labels and value labels are used in the table as well.

Customizing the table

With just a few modifications, we can make the table above look better.

By default, the base levels of factor variables are included in the table. Below, we remove the base levels. Then, we get a preview of our table.

```
. collect style showbase off

. collect preview
```

	Coefficient	95% CI	
Age (years)	.6383029	.6164668	.6601389
Weight (kg)	.4069294	.382467	.4313917
MW	-.2397311	-1.345286	.8658237
S	-.6187414	-1.717348	.4798654
W	-.8617777	-1.98006	.2565047
Intercept	71.70779	69.53642	73.87916

We would also like to format the statistics to two decimal places. We can do this with `collect style cell`. By not specifying a dimension, we have applied this formatting to all cells in the table with numeric content. The table would look neater if we enclose the confidence intervals in brackets and use a comma as the delimiter. This formatting applies only to the confidence intervals (`_r_ci`), so we specify the dimension and level with `collect style cell`. Then, we remove the vertical border and preview our table once more.

```
. collect style cell, nformat(%5.2f)
. collect style cell result[_r_ci], sformat("[%s]") cidelimiter(", ")
. collect style cell border_block, border(right, pattern(nil))
. collect preview
```

```
-----------------------------------------------
                 Coefficient      95% CI
-----------------------------------------------
Age (years)             0.64  [0.62,  0.66]
Weight (kg)             0.41  [0.38,  0.43]
MW                     -0.24 [-1.35,  0.87]
S                      -0.62 [-1.72,  0.48]
W                      -0.86 [-1.98,  0.26]
Intercept              71.71 [69.54, 73.88]
-----------------------------------------------
```

The last thing that would make this table complete would be to display stars for significance. Below, we define the levels of the p-values, stored in _r_p, for which stars should be shown. We will display three stars for p-values less than 0.01, two stars for values less than 0.05, and one star for values less than 0.1. Additionally, we have added three spaces to all other results, so that our results will be aligned when viewing them in plain text or in the Stata Markup and Control Language format. We can display the stars in a separate column, but below we attach them to the coefficients (_r_b). Then, we simply add a note explaining the significance represented by the stars and preview our table for the last time.

```
. collect stars _r_p 0.01 "***" 0.05 "** " 0.1 "*  " 1 "   ", attach(_r_b)
. collect notes : "*** p<.01, ** p<.05, * p<.1"
. collect preview
```

```
-----------------------------------------------
                 Coefficient      95% CI
-----------------------------------------------
Age (years)          0.64***  [0.62,  0.66]
Weight (kg)          0.41***  [0.38,  0.43]
MW                  -0.24    [-1.35,  0.87]
S                   -0.62    [-1.72,  0.48]
W                   -0.86    [-1.98,  0.26]
Intercept           71.71*** [69.54, 73.88]
-----------------------------------------------
*** p<.01, ** p<.05, * p<.1
```

Note that we could have easily added the note using the `shownote` option with `collect stars`. However, this note would have included `p<1`, so we instead added our custom note with `collect notes`.

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

[TABLES] **collect style cell** — Collection styles for cells

[TABLES] **collect style showbase** — Collection styles for displaying base levels

Description Remarks and examples Reference Also see

Description

In this example, we demonstrate how to collect results from multiple regressions and create a table of coefficients, standard errors, and statistics computed after fitting the model. We also show how to customize the resulting table.

Remarks and examples

Remarks are presented under the following headings:

Collecting regression results and creating a table
Customizing the table

Collecting regression results and creating a table

Below, we use data from the Second National Health and Nutrition Examination Survey (NHANES II) (McDowell et al. 1981). We would like to create a table comparing the results from two models. If we were including only the estimation results, we could easily create this table with `etable`; this command is used to create and export tables of estimation results in a single step. However, we want to include results from another command, `testparm`, in our table. So, we use the `collect` suite of commands.

We begin by creating a new collection named `ex6`. Then, we fit a model for systolic blood pressure (`bpsystol`) as a function of `weight`, `sex`, and whether an individual has `diabetes`. We use the `collect` prefix to collect the coefficients (`_r_b`) and standard errors (`_r_se`) into the `ex6` collection. We also attach the tag `model[(1)]` to these results. We can later use this tag to refer to these results when we build and customize our table.

```
. use https://www.stata-press.com/data/r19/nhanes2l
(Second National Health and Nutrition Examination Survey)

. collect create ex6
(current collection is ex6)

. collect _r_b _r_se, tag(model[(1)]): regress bpsystol weight i.diabetes i.sex

      Source |       SS           df       MS      Number of obs   =    10,349
-------------+----------------------------------   F(3, 10345)     =    382.25
       Model |  562138.283         3  187379.428   Prob > F        =    0.0000
    Residual |  5071141.76    10,345    490.2022   R-squared       =    0.0998
-------------+----------------------------------   Adj R-squared   =    0.0995
       Total |  5633280.05    10,348   544.38346   Root MSE        =    22.141

------------------------------------------------------------------------------
    bpsystol | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
      weight |   .4340474   .0153533    28.27   0.000     .4039519    .4641428
             |
    diabetes |
   Diabetic  |   14.34115   1.019611    14.07   0.000     12.34252    16.33979
             |
         sex |
     Female  |   1.107633   .4710559     2.35   0.019     .1842724    2.030994
       _cons |   98.40567   1.235476    79.65   0.000      95.9839    100.8274
------------------------------------------------------------------------------
```

In fact, all results that regress stores in e() are collected when we run the command above. By specifying _r_b and _r_se following collect, we have requested that these results be automatically included in a table when we include the dimension result.

Let's also use the testparm command to test whether the coefficient on diabetes is different from zero. We collect the *p*-value that testparm returns in the scalar r(p). We also tag this result with model[(1)], which will allow us to easily align the result from this command with the regression results when we construct our table.

```
. collect p_d=r(p), tag(model[(1)]): testparm i.diabetes

 ( 1)  1.diabetes = 0

       F(  1, 10345) =  197.83
            Prob > F =    0.0000
```

Now, we add the interaction between diabetes and sex to the model. We use the collect prefix again to collect the results from this model. We add quietly prefix to suppress the output. Now that diabetes is interacted with sex, we perform a joint test for the hypothesis that all coefficients associated with diabetes, including those in the interaction, are equal to zero. This time we attach the tag model[(2)] to the results from both the regress and the testparm commands.

```
. quietly: collect _r_b _r_se, tag(model[(2)]): regress bpsystol weight
> diabetes##sex

. collect p_d=r(p), tag(model[(2)]): testparm i.diabetes i.diabetes#i.sex

 ( 1)  1.diabetes = 0
 ( 2)  1.diabetes#2.sex = 0

       F(  2, 10344) =  100.11
            Prob > F =    0.0000
```

All the results are now stored in the current collection. We are ready to arrange the values into a table. These values are organized in the collection by tags, which are made up of dimensions and levels within those dimensions. We need to know the dimension names to lay out and customize our table. Below, we list the dimensions:

```
. collect dims

Collection dimensions
Collection: ex6
-----------------------------------------
                Dimension   No. levels
-----------------------------------------
Layout, style, header, label
                   cmdset   4
                    coleq   1
                  colname   10
        colname_remainder   1
                 diabetes   2
                    model   2
            program_class   3
                   result   37
              result_type   3
                  rowname   1
                      sex   2

Style only
             border_block   4
                cell_type   4
-----------------------------------------
```

The output indicates which dimensions can be used with the `collect` subcommands. For example, the first section lists dimensions that can be specified in `collect layout`, which is used to arrange the values in the collection into a table. We place the covariate names (`colname`) and the statistics (`result`) on the rows. We place `model`, the dimension we created with the `tag()` option of `collect` above, on the columns.

```
. collect layout (colname#result) (model)

Collection: ex6
      Rows: colname#result
   Columns: model
   Table 1: 30 x 2
```

	(1)	(2)
Weight (kg)		
Coefficient	.4340474	.4335342
Std. error	.0153533	.0153559
Not diabetic		
Coefficient	0	0
Std. error	0	0
Diabetic		
Coefficient	14.34115	12.57211
Std. error	1.019611	1.538361
Male		
Coefficient	0	0
Std. error	0	0
Female		
Coefficient	1.107633	.9520999
Std. error	.4710559	.4817911
Not diabetic # Male		
Coefficient		0
Std. error		0
Not diabetic # Female		
Coefficient		0
Std. error		0
Diabetic # Male		
Coefficient		0
Std. error		0
Diabetic # Female		
Coefficient		3.146466
Std. error		2.048958
Intercept		
Coefficient	98.40567	98.5238
Std. error	1.235476	1.237787

In the following section, we format the results, remove the base levels, and make some other edits to make this table ready for publication.

Customizing the table

In the table above, the base levels of factor variables were included in the table. Below, we remove the base levels with `collect style showbase`. We also format the statistics to two decimal places. We can do this with `collect style cell`. This command allows us to format all cells in the table at once or to format specific cells. Because we want to apply the numeric formatting to all cells, we do not specify a dimension. We also remove the border on the right side of the row headers by setting the border pattern for this location (`right`) to `nil`. Next, we want to enclose the standard errors in parentheses. The standard errors are stored in the level `_r_se` of the dimension `result`. To apply this format only to this result, we specify the dimension (`result`) and its level; we use brackets (`[]`) to refer to levels of a dimension. Then, we get a preview of our table.

```
. collect style showbase off
. collect style cell, nformat(%5.2f)
. collect style cell border_block, border(right, pattern(nil))
. collect style cell result[_r_se], sformat("(%s)")
. collect preview
```

	(1)	(2)
Weight (kg)		
Coefficient	0.43	0.43
Std. error	(0.02)	(0.02)
Diabetic		
Coefficient	14.34	12.57
Std. error	(1.02)	(1.54)
Female		
Coefficient	1.11	0.95
Std. error	(0.47)	(0.48)
Diabetic # Female		
Coefficient		3.15
Std. error		(2.05)
Intercept		
Coefficient	98.41	98.52
Std. error	(1.24)	(1.24)

Next, we would like to center the results and column headers horizontally. We will need to refer to the levels of the dimension `cell_type`. Below, we list the levels of this dimension. This dimension divides the table into four sections. The sections we want to modify are the values (`items`) in the body of the table and the `column-headers`. We specify the dimension `cell_type` and these levels with `collect style cell` to modify their horizontal alignment.

```
. collect levelsof cell_type
Collection: ex6
 Dimension: cell_type
    Levels: column-header corner item row-header
. collect style cell cell_type[item column-header], halign(center)
```

We also remove the labels `Coefficient` and `Std. error`. These labels are attached to the levels of the dimension `result`. We use `collect style header` to hide the level labels for this dimension. Then, to add an additional space between columns, we use `collect style column` with the `extraspace()` option.

We can arrange our row headers in two ways. One way is to place each item in a separate cell; the other way is to stack the elements in a single column. We choose the latter with `collect style row stack`. Also, notice that by default `collect` uses a `#` as a delimiter for interaction terms. We would instead like to use an `x`, with a space on each side.

```
. collect style header result, level(hide)
. collect style column, extraspace(1)
. collect style row stack, spacer delimiter(" x ")
. collect preview
```

	(1)	(2)
Weight (kg)	0.43	0.43
	(0.02)	(0.02)
Diabetic	14.34	12.57
	(1.02)	(1.54)
Female	1.11	0.95
	(0.47)	(0.48)
Diabetic x Female		3.15
		(2.05)
Intercept	98.41	98.52
	(1.24)	(1.24)

Recall that all `e()` results were collected from our models. In addition to reporting the p-value from `testparm`, we also want to report the R^2 value, which is stored under the level `r2` of the dimension `result`. So, in addition to the results for each covariate (`colname#result`), we also specify the levels `r2` and `p_d` of `result` in the first set of parentheses, which will be used to define the rows of the table. As before, we use `model` for the column identifier.

```
. collect layout (colname#result result[r2 p_d]) (model)
Collection: ex6
      Rows: colname#result result[r2 p_d]
   Columns: model
   Table 1: 18 x 2
```

	(1)	(2)
Weight (kg)	0.43	0.43
	(0.02)	(0.02)
Diabetic	14.34	12.57
	(1.02)	(1.54)
Female	1.11	0.95
	(0.47)	(0.48)
Diabetic x Female		3.15
		(2.05)
Intercept	98.41	98.52
	(1.24)	(1.24)
	0.10	0.10
	0.00	0.00

We hid the labels for the levels of the dimension `result`, but now that we have added the p-values and values of R^2, we want to display their levels. We specify these two levels of the dimension `result` with `collect style header`. Then, we modify the labels for these levels. Additionally, we want to format our p-values to three decimal places and display them as `<0.001` if they are less than 0.001. We can do this by specifying `minimum(0.001)` with `collect style cell`. After making that change, we preview our table once more:

```
. collect style header result[r2 p_d], level(label)
. collect label levels result p_d "Diabetes p-value" r2 "R-squared", modify
. collect style cell result[p_d], nformat(%5.3f) minimum(0.001)
. collect preview
```

	(1)	(2)
Weight (kg)	0.43	0.43
	(0.02)	(0.02)
Diabetic	14.34	12.57
	(1.02)	(1.54)
Female	1.11	0.95
	(0.47)	(0.48)
Diabetic x Female		3.15
		(2.05)
Intercept	98.41	98.52
	(1.24)	(1.24)
R-squared	0.10	0.10
Diabetes p-value	<0.001	<0.001

Last, we want to add stars to the coefficients to indicate which are significant at a 1%, 5%, and 10% level. The significance is determined by the p-values (_r_p), but the actual stars are attached to the coefficients (_r_b). The shownote option adds the note at the bottom of the table, explaining the significance represented by the stars.

```
. collect stars _r_p 0.01 "***"  0.05 "** " 0.1 "*  ", attach(_r_b) shownote
. collect preview
```

	(1)	(2)
Weight (kg)	0.43***	0.43***
	(0.02)	(0.02)
Diabetic	14.34***	12.57***
	(1.02)	(1.54)
Female	1.11**	0.95**
	(0.47)	(0.48)
Diabetic x Female		3.15
		(2.05)
Intercept	98.41***	98.52***
	(1.24)	(1.24)
R-squared	0.10	0.10
Diabetes p-value	<0.001	<0.001

```
*** p<.01, ** p<.05, * p<.1
```

Now, we can export our table to our preferred format—Word, PDF, HTML, LaTeX, Excel, or Markdown—using `collect export`.

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

[TABLES] **collect style cell** — Collection styles for cells

[TABLES] **collect style showbase** — Collection styles for displaying base levels

Description Remarks and examples Reference Also see

Description

In this example, we demonstrate how to use `collect` to create a table of regression results when working with complex survey data.

Remarks and examples

Remarks are presented under the following headings:

Introduction

Throughout this manual, we have used data from the Second National Health and Nutrition Examination Survey (NHANES II) (McDowell et al. 1981), ignoring the fact that it was derived from a complex survey design. Below, we report the current survey settings:

```
. use https://www.stata-press.com/data/r19/nhanes2l
(Second National Health and Nutrition Examination Survey)
. svyset
Sampling weights: finalwgt
             VCE: linearized
     Single unit: missing
        Strata 1: strata
 Sampling unit 1: psu
           FPC 1: <zero>
```

Because the survey design has already been declared for these data, we can perform our analyses accounting for the complex survey design by prefixing our estimation commands with the `svy` prefix. We will use this prefix when fitting a regression model below.

Table of regression results with complex survey data

We would like to fit the model we fit in [TABLES] **Example 5** but using the `svy` prefix to account for the complex survey design of our data. Also, we would like to include information about the survey design in our table, specifically the number of strata and primary sampling units. Below, we create a new collection named `ex7`. Then, we fit the model and use the `collect` prefix to collect all the results from `e()`. We also use the `quietly` prefix to suppress the output:

```
. collect create ex7
(current collection is ex7)
. quietly: collect: svy: regress bpsystol bmi i.agegrp i.sex i.race
```

Then, we arrange the items in our collection with collect layout. We specify that the coefficients (_r_b) for each covariate (colname) be placed on the rows, along with the number of strata (N_strata) and the primary sampling units (N_psu). All collected results are stored in the dimension result, and we use brackets ([]) to refer to the levels of a dimension. On the columns, we place the results for each command (cmdset). In this case, we collected results from a single command, but including this dimension will allow us to add a label at the top.

```
. collect layout (colname#result[_r_b] result[N_strata N_psu]) (cmdset)
Collection: ex7
      Rows: colname#result[_r_b] result[N_strata N_psu]
   Columns: cmdset
   Table 1: 28 x 1

──────────────────────────────────────
                        |            1
────────────────────────+─────────────
Body mass index (BMI)   |
  Coefficient           |     1.314165
20-29                   |
  Coefficient           |            0
30-39                   |
  Coefficient           |     .9554468
40-49                   |
  Coefficient           |      6.40242
50-59                   |
  Coefficient           |     14.87049
60-69                   |
  Coefficient           |     21.61949
70+                     |
  Coefficient           |     27.90405
Male                    |
  Coefficient           |            0
Female                  |
  Coefficient           |    -5.616304
White                   |
  Coefficient           |            0
Black                   |
  Coefficient           |     1.397293
Other                   |
  Coefficient           |    -.3119783
Intercept               |
  Coefficient           |     88.59293
Number of strata        |           31
Number of sampled PSU   |           62
──────────────────────────────────────
```

We can polish our table with a few modifications. First, we add a label to the level of cmdset, which will indicate the outcome variable. Then, because we are reporting only coefficients for each variable, we can hide the label Coefficient. This label is attached to the coefficients (_r_b), which are a level of the dimension result. Below, we hide the label for this level of the dimension with collect style header.

```
. collect label levels cmdset 1 "Model for systolic BP"
. collect style header result[_r_b], level(hide)
```

If you are wondering why we do not just hide the labels for all levels of result, recall that our survey design statistics are also part of this dimension.

Next, we omit the base levels for the factor variables with collect style showbase. We would like to include the labels for the factor variables and the labels for their levels in a single column. So we stack the row headers with collect style row, adding a blank line between the stacked row dimensions. By default, when we stack the row headers, the factor variables will be bound to their levels by an equal sign. We specify the nobinder option so that the levels will be stacked under the factor-variable labels, instead of bound by an equal sign.

```
. collect style showbase off
. collect style row stack, nobinder spacer
```

Finally, we format the regression results to display only two digits to the right of the decimal. We do not need to apply this format to the survey design information, so we specify colname#result, which includes only the regression results. Additionally, we remove the border that is displayed on the right by default by setting the border pattern to nil. Then, we preview our finalized table:

```
. collect style cell colname#result, nformat(%5.2f)
. collect style cell border_block, border(right, pattern(nil))
. collect preview
```

	Model for systolic BP
Body mass index (BMI)	1.31
Age group	
30-39	0.96
40-49	6.40
50-59	14.87
60-69	21.62
70+	27.90
Sex	
Female	-5.62
Race	
Black	1.40
Other	-0.31
Intercept	88.59
Number of strata	31
Number of sampled PSU	62

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

[TABLES] **collect style header** — Collection styles for hiding and showing header components

[TABLES] **collect style row** — Collection styles for row headers

Description Remarks and examples Reference Also see

Description

In this example, we demonstrate how to use `collect` to store the results of `anova` and how to use a predefined style to produce an ANOVA table with these results.

Remarks and examples

Below, we use data from the Second National Health and Nutrition Examination Survey (NHANES II) (McDowell et al. 1981). We wish to test whether the mean systolic blood pressure (`bpsystol`) is the same across males and females and across each category of race. To perform these tests, we use the `anova` command. We first create a new collection named `ex8` and then use the `collect` prefix to collect the results from `anova` and store them in the new collection.

```
. use https://www.stata-press.com/data/r19/nhanes2l
(Second National Health and Nutrition Examination Survey)
. collect create ex8
(current collection is ex8)
. quietly: collect : anova bpsystol sex race
```

All results that `anova` returns in `e()` and `r()` will be collected. These results include the usual linear regression estimation results, sums of squares of the model terms, their degrees of freedom, F statistics, and p-values.

Stata ships with a predefined style named `anova` suitable for producing ANOVA tables after `anova` and `oneway`. Here we replace the default style with the `anova` style and call `collect preview` to report our ANOVA table.

```
. collect style use anova, replace
Collection: ex8
      Rows: term
   Columns: result[SS df MS F p]
    Tables: cmdset
   Table 1: 5 x 5
. collect preview
```

	SS	df	MS	F statistic	Prob>F
Model	48978.71	3	16326.24	30.24	<0.0001
sex	38166.29	1	38166.29	70.70	<0.0001
race	11316.64	2	5658.32	10.48	<0.0001
Residual	5585691.32	10,347	539.84		
Total	5634670.03	10,350	544.41		

Use `collect style` commands to make further style changes. Use the `collect label levels` command to make any necessary label changes. Once we are happy with our ANOVA table's style and labels, we can export it to another format with `collect export`.

❑ Technical note

If you collect results from multiple calls to anova and use the above predefined style, the default layout will produce a separate table for each collected call to anova. The cmdset dimension indexes the collected estimation results, and the above predefined style specifies it in the layout to produce the separate tables.

Suppose we collected estimation results from a second ANOVA model.

```
. quietly: collect: anova bpsystol sex##race
```

To see how the estimation command results and cmdset levels are assigned, type

```
. collect layout (cmdset) (result[cmdline])
```

To restore the layout to show the above ANOVA tables, type

```
. collect style use anova, layout
```

If we want to publish the second model fit in our report, we can select it using collect style autolevels with the cmdset dimension without having to change collect layout by typing

```
. collect style autolevels cmdset 2, clear
. collect preview
```

❑

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

[TABLES] **collect style use** — Use collection styles from disk

[TABLES] **Predefined styles** — Predefined collection styles

Glossary

automatic levels. Automatic levels are levels of the `result` dimension (types of statistics) that are selected to be automatically included in the table if specific levels are not requested at the time the table is laid out. Automatic levels can be selected at the time results are collected by using `collect get` or the `collect` prefix. Alternatively, automatic results can be selected at any time by using `collect style autolevels`.

automatic results. See *automatic levels*.

collection. A collection contains results from one or more Stata commands. The results in a collection can be used to create a table. Within the collection, the values returned by the Stata commands are organized by tags, dimensions, and levels, which are used to determine how the values are arranged in a table.

current collection. Stata can have many collections in memory at a time. The current collection is the active collection—the collection to which `collect` subcommands are applied. By default, any new results collected with the `collect` prefix are placed in the current collection. Any style changes and label changes are applied to this collection. A new table built using `collect layout`, exported using `collect export`, or saved using `collect save` is based on this collection.

dimensions. See *tags, dimensions, and levels*.

item. An item is a value in a collection. See *value*.

layout. The layout is the arrangement of a table. The layout is determined by rows, columns, and separate tables. When creating a table from a collection, you specify the layout by identifying dimensions to be placed on the rows, columns, and potentially separate tables.

levels. See *tags, dimensions, and levels*.

tags. See *tags, dimensions, and levels*.

tags, dimensions, and levels. Tags are assigned to all values in a collection when you either `collect:` or `collect get` results. Custom tags can be added when collecting results. You can retrieve any value from a collection by specifying its tags on a `collect layout` command. More typically, you specify lists of tags to create a table.

Here are some examples of tags:

`result[r2]`	specifies the R^2 result
`foreign[1]`	specifies `foreign` $= 1$
`colname[mpg]`	specifies the covariate mpg

Tags comprise two parts, a dimension and level. Here are the parts of `result[r2]`:

`result[r2]`	tag
`result`	dimension—the dimension of tag `result[r2]`
`r2`	level—a level of dimension `result`

Dimensions can contain multiple tags; each tag will have its own level. Consider the following:

`result[N]`	another tag in dimension `result`
`N`	another level of dimension `result`

Dimensions must have valid names; see **[U] 11.3 Naming conventions**.

Levels can be integers or strings, and the strings may contain spaces. If a level contains spaces, it must be quoted, for example, “my level”.

Some `collect` command arguments and options require a single tag:

```
result[r2]
```

Most `collect` command arguments and options accept tag lists, for example,

```
result[r2] result[N]
```

or, equivalently,

```
result[r2 N]
```

You can also just type a dimension name,

```
result
```

Wherever tag lists are allowed, a dimension name alone specifies a list of all the tags in the dimension. If `result` has levels `r2`, `N`, `ll`, `rmse`, then `result` is interpreted as

```
result[r2] result[N] result[ll] result[rmse]
```

value. In a collection, a value is a number that can be used to fill a cell in a table. The values are obtained from the stored results of Stata commands that are included in the collection. Values are organized by tags, dimensions, and levels.

Subject and author index

See the combined subject index and the combined author index in the *Stata Index*.