

The *Modify Columns* Family of Functions

The `cols_*()` functions allow for modifications that act on entire columns. This includes alignment of the data in columns (`cols_align()`), hiding columns from view (`cols_hide()`), re-labeling the column labels (`cols_label()`), merging two columns together (`cols_merge*()`), moving columns around (`cols_move*()`), and using a column name delimiter to create labels in the column spanner (`cols_split_delim()`).

`cols_align()`: Set the alignment of columns

The individual alignments of columns (which includes the column labels and all of their data cells) can be modified. We have the option to align text to the `left`, the `center`, and the `right`. In a less explicit manner, we can allow **gt** to automatically choose the alignment of each column based on the data type (with the `auto` option).

EXAMPLE

Use `countrypops` to create a **gt** table. Align the `population` column data to the left.

```
countrypops %>%
  dplyr::select(-contains("code")) %>%
  dplyr::filter(country_name == "Mongolia") %>%
  tail(5) %>%
  gt() %>%
  cols_align(
    align = "left",
    columns = population
  )
```

country_name	year	population
Mongolia	2013	2869107
Mongolia	2014	2923896
Mongolia	2015	2976877
Mongolia	2016	3027398
Mongolia	2017	3075647

`cols_width()`: Set the widths of columns

Manual specifications of column widths can be performed using the `cols_width()` function. We choose which columns get specific widths (in pixels, usually by use of the `px()` helper function). Width assignments are supplied in `...` through two-sided formulas, where the left-hand side defines the target columns and the right-hand side is a single width value in pixels.

EXAMPLES

Use `exibble` to create a **gt** table. With named arguments in `...`, we can specify the exact widths for table columns (using `everything()` or `TRUE` will capture all remaining columns).

```

exibble %>%
  dplyr::select(num, char, date, datetime, row) %>%
  gt() %>%
  cols_width(
    num ~ px(150),
    ends_with("r") ~ px(100),
    starts_with("date") ~ px(200),
    everything() ~ px(60)
  )

```

num	char	date	datetime	row
1.111e-01	apricot	2015-01-15	2018-01-01 02:22	row_
2.222e+00	banana	2015-02-15	2018-02-02 14:33	row_
3.333e+01	coconut	2015-03-15	2018-03-03 03:44	row_
4.444e+02	durian	2015-04-15	2018-04-04 15:55	row_
5.550e+03	NA	2015-05-15	2018-05-05 04:00	row_
NA	fig	2015-06-15	2018-06-06 16:11	row_
7.770e+05	grapefruit	NA	2018-07-07 05:22	row_
8.880e+06	honeydew	2015-08-15	NA	row_

cols_label(): Relabel one or more columns

Column labels can be modified from their default values (the names of the columns from the input table data). When you create a **gt** table object using `gt()`, column names effectively become the column labels. While this serves as a good first approximation, column names aren't often appealing as column labels in a **gt** output table. The `cols_label()` function provides the flexibility to relabel one or more columns and we even have the option to use the `md()` or `html()` helper functions for rendering column labels from Markdown or using HTML.

EXAMPLES

Use `country pops` to create a **gt** table; label all the table's columns to present better.

```

country pops %>%
  dplyr::select(-contains("code")) %>%
  dplyr::filter(country_name == "Mongolia") %>%
  tail(5) %>%
  gt() %>%
  cols_label(
    country_name = "Name",
    year = "Year",
    population = "Population"
  )

```

Name	Year	Population
Mongolia	2013	2869107
Mongolia	2014	2923896
Mongolia	2015	2976877
Mongolia	2016	3027398
Mongolia	2017	3075647

Use `countrypops` to create a **gt** table; label columns as before but make them bold with markdown formatting.

```
countrypops %>%
  dplyr::select(-contains("code")) %>%
  dplyr::filter(country_name == "Mongolia") %>%
  tail(5) %>%
  gt() %>%
  cols_label(
    country_name = md("***Name***"),
    year = md("***Year***"),
    population = md("***Population***")
  )
```

Name	Year	Population
Mongolia	2013	2869107
Mongolia	2014	2923896
Mongolia	2015	2976877
Mongolia	2016	3027398
Mongolia	2017	3075647

`cols_move_to_start()`: Move one or more columns to the start

We can easily move set of columns to the beginning of the column series and we only need to specify which columns. It's possible to do this upstream of **gt**'s API, however, it is easier with this function and it presents less possibility for error. The ordering of the columns that are moved to the start is preserved (same with the ordering of all other columns in the table).

EXAMPLES

Use `countrypops` to create a **gt** table; With the remaining columns, move the `year` column to the start.

```
countrypops %>%
  dplyr::select(-contains("code")) %>%
  dplyr::filter(country_name == "Mongolia") %>%
  tail(5) %>%
  gt() %>%
  cols_move_to_start(columns = year)
```

year	country_name	population
2013	Mongolia	2869107
2014	Mongolia	2923896
2015	Mongolia	2976877
2016	Mongolia	3027398
2017	Mongolia	3075647

Use `countrypops` to create a **gt** table; With the remaining columns, move `year` and `population` to the start.

```
countrypops %>%
  dplyr::select(-contains("code")) %>%
  dplyr::filter(country_name == "Mongolia") %>%
  tail(5) %>%
  gt() %>%
  cols_move_to_start(columns = c(year, population))
```

year	population	country_name
2013	2869107	Mongolia
2014	2923896	Mongolia
2015	2976877	Mongolia
2016	3027398	Mongolia
2017	3075647	Mongolia

`cols_move_to_end()`: Move one or more columns to the end

It's possible to move a set of columns to the end of the column series, we only need to specify which columns are to be moved. While this can be done upstream of **gt**'s API, this function makes to process much easier and it's less error prone. The ordering of the columns that are moved to the end is preserved (same with the ordering of all other columns in the table).

EXAMPLES

Use `countrypops` to create a **gt** table. With the remaining columns, move the `year` column to the end.

```
countrypops %>%
  dplyr::select(-contains("code")) %>%
  dplyr::filter(country_name == "Mongolia") %>%
  tail(5) %>%
  gt() %>%
  cols_move_to_end(columns = year)
```

country_name	population	year
Mongolia	2869107	2013
Mongolia	2923896	2014
Mongolia	2976877	2015
Mongolia	3027398	2016
Mongolia	3075647	2017

Use `countrypops` to create a **gt** table. With the remaining columns, move `year` and `country_name` to the end.

```
countrypops %>%
  dplyr::select(-contains("code")) %>%
  dplyr::filter(country_name == "Mongolia") %>%
  tail(5) %>%
  gt() %>%
  cols_move_to_end(columns = c(year, country_name))
```

population	year	country_name
2869107	2013	Mongolia
2923896	2014	Mongolia
2976877	2015	Mongolia
3027398	2016	Mongolia
3075647	2017	Mongolia

cols_move(): Move one or more columns

On those occasions where you need to move columns this way or that way, we can make use of the `cols_move()` function. While it's true that the movement of columns can be done upstream of **gt**'s API, it is much easier and less error prone to use the function provided here. The movement procedure here takes one or more specified columns (in the `columns` argument) and places them to the right of a different column (the `after` argument). The ordering of the columns to be moved is preserved, as is the ordering of all other columns in the table.

EXAMPLE

Use `countrypops` to create a **gt** table; With the remaining columns, position `population` after `country_name`.

```
countrypops %>%
  dplyr::select(-contains("code")) %>%
  dplyr::filter(country_name == "Mongolia") %>%
  tail(5) %>%
  gt() %>%
  cols_move(
    columns = population,
    after = country_name
  )
```

country_name	population	year
Mongolia	2869107	2013
Mongolia	2923896	2014
Mongolia	2976877	2015
Mongolia	3027398	2016
Mongolia	3075647	2017

cols_hide(): Hide one or more columns

The `cols_hide()` function allows us to hide one or more columns from appearing in the final output table. While it's possible and often desirable to omit columns from the input table data before introduction to the `gt()` function, there can be cases where the data in certain columns is useful (as a column reference during formatting of other columns) but the final display of those columns is not necessary.

EXAMPLES

Use `countrypops` to create a **gt** table; Hide the columns `country_code_2` and `country_code_3`.

```
countrypops %>%
  dplyr::filter(country_name == "Mongolia") %>%
  tail(5) %>%
  gt() %>%
  cols_hide(columns = c(country_code_2, country_code_3))
```

country_name	year	population
Mongolia	2013	2869107
Mongolia	2014	2923896
Mongolia	2015	2976877
Mongolia	2016	3027398

country_name	year	population
Mongolia	2017	3075647

Use `countrypops` to create a gt table and use the `population` column to provide the conditional placement of footnotes, then hide that column and one other.

```
countrypops %>%
  dplyr::filter(country_name == "Mongolia") %>%
  tail(5) %>%
  gt() %>%
  cols_hide(columns = c(country_code_3, population)) %>%
  tab_footnote(
    footnote = "Population above 3,000,000.",
    locations = cells_body(
      columns = year,
      rows = population > 3000000
    )
  )
```

country_name	country_code_2	year
Mongolia	MN	2013
Mongolia	MN	2014
Mongolia	MN	2015
Mongolia	MN	2016 ¹
Mongolia	MN	2017 ¹
¹ Population above 3,000,000.		

cols_unhide(): Unhide one or more columns

The `cols_unhide()` function allows us to take one or more hidden columns (usually made so via the `cols_hide()` function) and make them visible in the final output table. This may be important in cases where the user obtains a `gt_tbl` object with hidden columns and there is motivation to reveal one or more of those.

EXAMPLES

Use `countrypops` to create a gt table. Hide the columns `country_code_2` and `country_code_3`.

```
gt_tbl <-
  countrypops %>%
  dplyr::filter(country_name == "Mongolia") %>%
  tail(5) %>%
  gt() %>%
  cols_hide(columns = c(country_code_2, country_code_3))
```

If the `gt_tbl` object is provided without the code or source data to regenerate it, and, the user wants to reveal otherwise hidden columns then the `cols_unhide()` function becomes useful.

```
gt_tbl %>%
  cols_unhide(columns = country_code_2)
```

country_name	country_code_2	year	population
Mongolia	MN	2013	2869107
Mongolia	MN	2014	2923896
Mongolia	MN	2015	2976877
Mongolia	MN	2016	3027398
Mongolia	MN	2017	3075647

cols_merge_uncert(): Merge two columns to a value & uncertainty column

The `cols_merge_uncert()` function is a specialized variant of the `cols_merge()` function. It operates by taking a base value column (`col_val`) and an uncertainty column (`col_uncert`) and merges them into a single column. What results is a column with values and associated uncertainties (e.g., 12.0 ± 0.1), and, the column specified in `col_uncert` is dropped from the output table.

EXAMPLE

Use `exibble` to create a **gt** table, keeping only the `currency` and `num` columns; merge columns into one with a base value and uncertainty (after formatting the `num` column).

```
exibble %>%
  dplyr::select(currency, num) %>%
  dplyr::slice(1:7) %>%
  gt() %>%
  fmt_number(
    columns = num,
    decimals = 3,
    use_seps = FALSE
  ) %>%
  cols_merge_uncert(
    col_val = currency,
    col_uncert = num
  ) %>%
  cols_label(currency = "value + uncert.")
```

value + uncert.
49.950 ± 0.111
17.950 ± 2.222

value + uncert.
1.390 ± 33.330
65100.000 ± 444.400
1325.810 ± 5550.000
13.255
NA

cols_merge_range(): Merge two columns to a value range column

The `cols_merge_range()` function is a specialized variant of the `cols_merge()` function. It operates by taking a two columns that constitute a range of values (`col_begin` and `col_end`) and merges them into a single column. What results is a column containing both values separated by a long dash (e.g., `12.0 – 20.0`). The column specified in `col_end` is dropped from the output table

EXAMPLE

Use `gtcars` to create a **gt** table, keeping only the `model` , `mpg_c` , and `mpg_h` columns; merge the `mpg` columns together as a single range column (which is labeled as *MPG* , in italics).

```
gtcars %>%
  dplyr::select(model, starts_with("mpg")) %>%
  dplyr::slice(1:8) %>%
  gt() %>%
  cols_merge_range(
    col_begin = mpg_c,
    col_end = mpg_h
  ) %>%
  cols_label(
    mpg_c = md("*MPG*")
  )
```

model	<i>MPG</i>
GT	11–18
458 Speciale	13–17
458 Spider	13–17
458 Italia	13–17
488 GTB	15–22
California	16–23
GTC4Lusso	12–17

model	<i>MPG</i>
FF	11–16

cols_merge_n_pct(): Merge two columns to combine counts and percentages

The `cols_merge_n_pct()` function is a specialized variant of the `cols_merge()` function. It operates by taking two columns that constitute both a count (`col_n`) and a fraction of the total population (`col_pct`) and merges them into a single column. What results is a column containing both counts and their associated percentages (e.g., 12 (23.2%)). The column specified in `col_pct` is dropped from the output table.

EXAMPLE

Use `pizzaplace` to create a gt table that displays the counts and percentages of the top 3 pizzas sold by pizza category in 2015; the `cols_merge_n_pct()` function is used to merge the `n` and `frac` columns (and the `frac` column is formatted using `fmt_percent()`).

```
pizzaplace %>%
  dplyr::group_by(name, type, price) %>%
  dplyr::summarize(
    n = dplyr::n(),
    frac = n/nrow(.),
    .groups = "drop"
  ) %>%
  dplyr::arrange(type, dplyr::desc(n)) %>%
  dplyr::group_by(type) %>%
  dplyr::slice_head(n = 3) %>%
  gt(
    rowname_col = "name",
    groupname_col = "type"
  ) %>%
  fmt_currency(columns = price) %>%
  fmt_percent(columns = frac) %>%
  cols_merge_n_pct(
    col_n = n,
    col_pct = frac
  ) %>%
  cols_label(
    n = md("*N* (%)"),
    price = "Price"
  ) %>%
  tab_style(
    style = cell_text(font = "monospace"),
    locations = cells_stub()
  ) %>%
  tab_stubhead(md("Cat. and \nPizza Code")) %>%
  tab_header(title = "Top 3 Pizzas Sold by Category in 2015") %>%
  tab_options(table.width = px(512))
```

Top 3 Pizzas Sold by Category in 2015

Top 3 Pizzas Sold by Category in 2015

Cat. and Pizza Code	Price	<i>N</i> (%)
chicken		
thai_ckn	\$20.75	1410 (2.84%)
southw_ckn	\$20.75	1016 (2.05%)
bbq_ckn	\$20.75	992 (2.00%)
classic		
big_meat	\$12.00	1914 (3.86%)
classic_dlx	\$16.00	1181 (2.38%)
hawaiian	\$10.50	1020 (2.06%)
supreme		
spicy_ital	\$20.75	1109 (2.24%)
ital_supr	\$16.50	941 (1.90%)
sicilian	\$12.25	751 (1.51%)
veggie		
five_cheese	\$18.50	1409 (2.84%)
four_cheese	\$17.95	1316 (2.65%)
mexicana	\$20.25	867 (1.75%)

cols_merge(): Merge data from two or more columns to a single column

This function takes input from two or more columns and allows the contents to be merged them into a single column, using a pattern that specifies the formatting. We can specify which columns to merge together in the `columns` argument. The string-combining pattern is given in the `pattern` argument. The first column in the `columns` series operates as the target column (i.e., will undergo mutation) whereas all following `columns` will be untouched.

EXAMPLE

Use `sp500` to create a **gt** table; merge the `open` & `close` columns together, and, the `low` & `high` columns (putting an em dash between both); rename the columns.

```

sp500 %>%
  dplyr::slice(50:55) %>%
  dplyr::select(-volume, -adj_close) %>%
  gt() %>%
  cols_merge(
    columns = c(open, close),
    hide_columns = close,
    pattern = "{1}&mdash;{2}"
  ) %>%
  cols_merge(
    columns = c(low, high),
    hide_columns = high,
    pattern = "{1}&mdash;{2}"
  ) %>%
  cols_label(
    open = "open/close",
    low = "low/high"
  )

```

date	open/close	low/high
2015-10-21	2033.47—2018.94	2017.22—2037.97
2015-10-20	2033.13—2030.77	2026.61—2039.12
2015-10-19	2031.73—2033.66	2022.31—2034.45
2015-10-16	2024.37—2033.11	2020.46—2033.54
2015-10-15	1996.47—2023.86	1996.47—2024.15
2015-10-14	2003.66—1994.24	1990.73—2009.56