The *Format Data* Family of Functions

Columns of data can be formatted with the `fmt_*()` functions. We can specify the rows of these columns quite precisely with the `rows` argument. We get to apply these functions exactly once to each data cell (last call wins). Need to do custom formatting? Use the `fmt()` function and define your own formatter within (or, create a wrapper with `fmt()` if you prefer). The `text_transform()` function allows for post-processing of any data, and we provide a function for that transformation.

`fmt_number()`: Format numeric values

With numeric values in a **gt** table, we can perform number-based formatting so that the targeted values are rendered with a higher consideration for tabular presentation.

EXAMPLES

Use `exibble` to create a **gt** table. Format the `num` column as numeric with three decimal places and with no use of digit separators.

```
exibble %>%
  gt() %>%
  fmt_number(
    columns = num,
    decimals = 3,
    use_seps = FALSE
  )
```

num	char	fctr	date	time	datetime	currency	row	group
0.111	apricot	one	2015-01-15	13:35	2018-01-01 02:22	49.950	row_1	grp_a
2.222	banana	two	2015-02-15	14:40	2018-02-02 14:33	17.950	row_2	grp_a
33.330	coconut	three	2015-03-15	15:45	2018-03-03 03:44	1.390	row_3	grp_a
444.400	durian	four	2015-04-15	16:50	2018-04-04 15:55	65100.000	row_4	grp_a
5550.000	NA	five	2015-05-15	17:55	2018-05-05 04:00	1325.810	row_5	grp_b
NA	fig	six	2015-06-15	NA	2018-06-06 16:11	13.255	row_6	grp_b
777000.000	grapefruit	seven	NA	19:10	2018-07-07 05:22	NA	row_7	grp_b
8880000.000	honeydew	eight	2015-08-15	20:20	NA	0.440	row_8	grp_b

Use `countrypops` to create a **gt** table. Format all numeric columns to use large-number suffixing.

```
countrypops %>%
  dplyr::select(country_code_3, year, population) %>%
  dplyr::filter(
    country_code_3 %in% c(
      "CHN", "IND", "USA", "PAK", "IDN")
  ) %>%
  dplyr::filter(year > 1975 & year %% 5 == 0) %>%
  tidyr::spread(year, population) %>%
  dplyr::arrange(desc(`2015`)) %>%
  gt(rowname_col = "country_code_3") %>%
  fmt_number(
    columns = 2:9,
    decimals = 2,
    suffixing = TRUE
  )
```

	1980	1985	1990	1995	2000	2005	2010	2015
CHN	981.23M	1.05B	1.14B	1.20B	1.26B	1.30B	1.34B	1.37B
IND	696.78M	781.67M	870.13M	960.48M	1.05B	1.14B	1.23B	1.31B
USA	227.22M	237.92M	249.62M	266.28M	282.16M	295.52M	309.34M	321.04M
IDN	147.49M	165.01M	181.44M	196.96M	211.54M	226.71M	242.52M	258.16M
PAK	78.07M	92.22M	107.68M	122.83M	138.52M	153.91M	170.56M	189.38M

`fmt_scientific()`: Format values to scientific notation

With numeric values in a **gt** table, we can perform formatting so that the targeted values are rendered in scientific notation.

EXAMPLE

Use `exibble` to create a **gt** table. Format the `num` column as partially numeric and partially in scientific notation.

```
exibble %>%
  gt() %>%
  fmt_number(
    columns = num,
    rows = num > 500,
    decimals = 1,
    scale_by = 1/1000,
    pattern = "{x}K"
  ) %>%
  fmt_scientific(
    columns = num,
    rows = num <= 500,
    decimals = 1
  )
```

num	char	fctr	date	time	datetime	currency	row	group
1.1×10^{-1}	apricot	one	2015-01-15	13:35	2018-01-01 02:22	49.950	row_1	grp_a
2.2	banana	two	2015-02-15	14:40	2018-02-02 14:33	17.950	row_2	grp_a
3.3×10^{1}	coconut	three	2015-03-15	15:45	2018-03-03 03:44	1.390	row_3	grp_a
4.4×10^{2}	durian	four	2015-04-15	16:50	2018-04-04 15:55	65100.000	row_4	grp_a
5.5K	NA	five	2015-05-15	17:55	2018-05-05 04:00	1325.810	row_5	grp_b
NA	fig	six	2015-06-15	NA	2018-06-06 16:11	13.255	row_6	grp_b
777.0K	grapefruit	seven	NA	19:10	2018-07-07 05:22	NA	row_7	grp_b
8,880.0K	honeydew	eight	2015-08-15	20:20	NA	0.440	row_8	grp_b

`fmt_percent()`: Format values as a percentage

With numeric values in a **gt** table, we can perform percentage-based formatting. It is assumed the input numeric values are in a fractional format since the numbers will be automatically multiplied by `100` before decorating with a percent sign.

EXAMPLE

Use `pizzaplace` to create a **gt** table. Format the `frac_of_quota` column to display values as percentages.

```
pizzaplace %>%
  dplyr::mutate(month = as.numeric(substr(date, 6, 7))) %>%
  dplyr::group_by(month) %>%
  dplyr::summarize(pizzas_sold = dplyr::n()) %>%
  dplyr::ungroup() %>%
  dplyr::mutate(frac_of_quota = pizzas_sold / 4000) %>%
  gt(rowname_col = "month") %>%
  fmt_percent(
    columns = frac_of_quota,
    decimals = 1
  )
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

	pizzas_sold	frac_of_quota
1	4232	105.8%
2	3961	99.0%
3	4261	106.5%
4	4151	103.8%
5	4328	108.2%
6	4107	102.7%
7	4392	109.8%
8	4168	104.2%
9	3890	97.2%
10	3883	97.1%
11	4266	106.7%
12	3935	98.4%

`fmt_currency()`: Format values as currencies

With numeric values in a **gt** table, we can perform currency-based formatting. This function supports both automatic formatting with a three-letter or numeric currency code. We can also specify a custom currency that is formatted according to the output context with the `currency()` helper function. Numeric formatting facilitated through the use of a locale ID.

EXAMPLES

Use `exibble` to create a **gt** table. Format the `currency` column to have currency values in euros (`EUR`).

```
exibble %>%
  gt() %>%
  fmt_currency(
    columns = currency,
    currency = "EUR"
  )
```

num	char	fctr	date	time	datetime	currency	row	group
1.111e-01	apricot	one	2015-01-15	13:35	2018-01-01 02:22	€49.95	row_1	grp_a
2.222e+00	banana	two	2015-02-15	14:40	2018-02-02 14:33	€17.95	row_2	grp_a
3.333e+01	coconut	three	2015-03-15	15:45	2018-03-03 03:44	€1.39	row_3	grp_a
4.444e+02	durian	four	2015-04-15	16:50	2018-04-04 15:55	€65,100.00	row_4	grp_a
5.550e+03	NA	five	2015-05-15	17:55	2018-05-05 04:00	€1,325.81	row_5	grp_b
NA	fig	six	2015-06-15	NA	2018-06-06 16:11	€13.26	row_6	grp_b
7.770e+05	grapefruit	seven	NA	19:10	2018-07-07 05:22	NA	row_7	grp_b
8.880e+06	honeydew	eight	2015-08-15	20:20	NA	€0.44	row_8	grp_b

Use `exibble` to create a **gt** table. Keep only the `num` and `currency` , columns, then, format those columns using the `CNY` and `GBP` currencies.

```
exibble %>%
  dplyr::select(num, currency) %>%
  gt() %>%
  fmt_currency(
    columns = num,
    currency = "CNY"
  ) %>%
  fmt_currency(
    columns = currency,
    currency = "GBP"
  )
```

num	currency

num	currency
¥0.11	£49.95
¥2.22	£17.95
¥33.33	£1.39
¥444.40	£65,100.00
¥5,550.00	£1,325.81
NA	£13.26
¥777,000.00	NA
¥8,880,000.00	£0.44

`fmt_bytes()`: Format values as bytes

With numeric values in a **gt** table, we can transform those to values of bytes with human readable units. The `fmt_bytes()` function allows for the formatting of byte sizes to either of two common representations: (1) with decimal units (powers of 1000, examples being `"kB"` and `"MB"`), and (2) with binary units (powers of 1024, examples being `"KiB"` and `"MiB"`).

EXAMPLES

Use `exibble` to create a gt table. Format the `num` column to have byte sizes in the binary standard.

```
exibble %>%
  dplyr::select(num) %>%
  gt() %>%
  fmt_bytes(columns = num)
```

num
0 B
2 B
33 B
444 B
5.5 kB
NA
777 kB
8.9 MB

Create a similar table with the `fmt_bytes()` function, this time showing byte sizes as binary values.

```
exibble %>%
  dplyr::select(num) %>%
  gt() %>%
  fmt_bytes(
    columns = num,
    standard = "binary"
  )
```

num
0 B
2 B
33 B
444 B
5.4 KiB
NA
758.8 KiB
8.5 MiB

`fmt_date()` : Format values as dates

Format input date values that are either of the `Date` type, or, are character-based and expressed according to the ISO 8601 date format (`YYYY-MM-DD`). Once the appropriate data cells are targeted with columns (and, optionally, rows), we can simply apply a preset date style (see table in `info_date_style()` for info) to format the dates.

EXAMPLES

Use `exibble` to create a **gt** table. Keep only the `date` and `time` columns. Format the `date` column to have dates formatted as `month_day_year` (date style `5`).

```
exibble %>%
  dplyr::select(date, time) %>%
  gt() %>%
  fmt_date(
    columns = date,
    date_style = 5
  )
```

date	time
January 15, 2015	13:35

date	time
February 15, 2015	14:40
March 15, 2015	15:45
April 15, 2015	16:50
May 15, 2015	17:55
June 15, 2015	NA
NA	19:10
August 15, 2015	20:20

Use `exibble` to create a **gt** table. keep only the `date` and `time` columns. Format the `date` column to have mixed date formats (dates after April will be different than the others).

```
exibble %>%
  dplyr::select(date, time) %>%
  gt() %>%
  fmt_date(
    columns = date,
    rows = as.Date(date) > as.Date("2015-04-01"),
    date_style = "m_day_year"
  ) %>%
  fmt_date(
    columns = date,
    rows = as.Date(date) <= as.Date("2015-04-01"),
    date_style = "day_m_year"
  )
```

date	time
15 Jan 2015	13:35
15 Feb 2015	14:40
15 Mar 2015	15:45
Apr 15, 2015	16:50
May 15, 2015	17:55
Jun 15, 2015	NA
NA	19:10
Aug 15, 2015	20:20

`fmt_time()` : Format values as times

Format input time values that are character-based and expressed according to the ISO 8601 time format (`HH:MM:SS`). Once the appropriate data cells are targeted with columns (and, optionally, rows), we can simply apply a preset time style (see table in `info_time_style()` for info) to format the times.

EXAMPLES

Use `exibble` to create a **gt** table. Keep only the `date` and `time` columns. Format the `time` column to have times formatted as `hms_p` (time style `3`).

```
exibble %>%
  dplyr::select(date, time) %>%
  gt() %>%
  fmt_time(
    columns = time,
    time_style = 3
  )
```

date	time
2015-01-15	1:35:00 PM
2015-02-15	2:40:00 PM
2015-03-15	3:45:00 PM
2015-04-15	4:50:00 PM
2015-05-15	5:55:00 PM
2015-06-15	NA
NA	7:10:00 PM
2015-08-15	8:20:00 PM

Use `exibble` to create a **gt** table. Keep only the `date` and `time` columns. Format the `time` column to have mixed time formats (times after 16:00 will be different than the others).

```
exibble %>%
  dplyr::select(date, time) %>%
  gt() %>%
  fmt_time(
    columns = time,
    rows = time > "16:00",
    time_style = 3
  ) %>%
  fmt_time(
    columns = time,
    rows = time <= "16:00",
    time_style = 4
  )
```

date	time
2015-01-15	1:35 PM
2015-02-15	2:40 PM
2015-03-15	3:45 PM
2015-04-15	4:50:00 PM
2015-05-15	5:55:00 PM
2015-06-15	NA
NA	7:10:00 PM
2015-08-15	8:20:00 PM

`fmt_datetime()` : Format values as date-times

Format input date-time values that are character-based and expressed according to the ISO 8601 date-time format (`YYYY-MM-DD HH:MM:SS`). Once the appropriate data cells are targeted with columns (and, optionally, rows), we can simply apply preset date and time styles (see tables in `info_date_style()` and `info_time_style()` for more info) to format the data-time values.

EXAMPLE

Use `exibble` to create a **gt** table. keep only the `datetime` column. Format the column to have dates formatted as `month_day_year` and times to be `hms_p` .

```
exibble %>%
  dplyr::select(datetime) %>%
  gt() %>%
  fmt_datetime(
    columns = datetime,
    date_style = 5,
    time_style = 3
  )
```

datetime
January 1, 2018 2:22:00 AM
February 2, 2018 2:33:00 PM
March 3, 2018 3:44:00 AM
April 4, 2018 3:55:00 PM
May 5, 2018 4:00:00 AM
June 6, 2018 4:11:00 PM

datetime
July 7, 2018 5:22:00 AM
NA

`fmt_markdown()`: Format Markdown text

Any Markdown-formatted text in the incoming cells will be transformed to the appropriate output type during render when using `fmt_markdown()`.

EXAMPLE

Create a few Markdown-based text snippets.

```
text_1a <- "
### This is Markdown.

Markdown’s syntax is comprised entirely of
punctuation characters, which punctuation
characters have been carefully chosen so as
to look like what they mean... assuming
you’ve ever used email.
"

text_1b <- "
Info on Markdown syntax can be found
[here](https://daringfireball.net/projects/markdown/).
"

text_2a <- "
The **gt** package has these datasets:

 - `countrypops`
 - `sza`
 - `gtcars`
 - `sp500`
 - `pizzaplace`
 - `exibble`
"

text_2b <- "
There's a quick reference [here](https://commonmark.org/help/).
"
```

Arrange the text snippets as a tibble using the `dplyr::tribble()` function. Then, create a **gt** table and format all columns with `fmt_markdown()`.

```
dplyr::tribble(
  ~Markdown, ~md,
  text_1a,   text_2a,
  text_1b,   text_2b,
) %>%
  gt() %>%
  fmt_markdown(columns = everything()) %>%
  tab_options(table.width = px(400))
```

Markdown	md
# This is Markdown. Markdown's syntax is comprised entirely of punctuation characters, which punctuation characters have been carefully chosen so as to look like what they mean... assuming you've ever used email.	The **gt** package has these datasets: - `countrypops` - `sza` - `gtcars` - `sp500` - `pizzaplace` - `exibble`
Info on Markdown syntax can be found here (https://daringfireball.net/projects/markdown/).	There's a quick reference here (https://commonmark.org/help/).

`fmt_passthrough()`: Format by simply passing data through

Format by passing data through no other transformation other than: (1) coercing to character (as all the `fmt_*()` functions do), and (2) applying text via the pattern argument (the default is to apply nothing). All of this is useful when don't want to modify the input data other than to decorate it within a pattern. Also, this function is useful when used as the formatter function in the `summary_rows()` function, where the output may be text or useful as is.

EXAMPLE

Use `exibble` to create a **gt** table. Keep only the `char` column. Pass the data in that column through but apply a simple pattern that adds an 's' to the non- `NA` values.

```
exibble %>%
  dplyr::select(char) %>%
  gt() %>%
  fmt_passthrough(
    columns = char,
    rows = !is.na(char),
    pattern = "{x}s"
  )
```

char
apricots
bananas

char
coconuts
durians
NA
figs
grapefruits
honeydews

`fmt_missing()` : Format missing values

Wherever there is missing data (i.e., `NA` values) a customizable mark may present better than the standard `NA` text that would otherwise appear. The `fmt_missing()` function allows for this replacement through its `missing_text` argument (where an em dash serves as the default).

EXAMPLE

Use `exibble` to create a **gt** table. The `NA` values in different columns will be given replacement text.

```
exibble %>%
  dplyr::select(-row, -group) %>%
  gt() %>%
  fmt_missing(
    columns = 1:2,
    missing_text = "missing"
  ) %>%
  fmt_missing(
    columns = 4:7,
    missing_text = "nothing"
  )
```

num	char	fctr	date	time	datetime	currency
1.111e-01	apricot	one	2015-01-15	13:35	2018-01-01 02:22	49.950
2.222e+00	banana	two	2015-02-15	14:40	2018-02-02 14:33	17.950
3.333e+01	coconut	three	2015-03-15	15:45	2018-03-03 03:44	1.390
4.444e+02	durian	four	2015-04-15	16:50	2018-04-04 15:55	65100.000
5.550e+03	missing	five	2015-05-15	17:55	2018-05-05 04:00	1325.810
missing	fig	six	2015-06-15	nothing	2018-06-06 16:11	13.255
7.770e+05	grapefruit	seven	nothing	19:10	2018-07-07 05:22	nothing

num	char	fctr	date	time	datetime	currency
8.880e+06	honeydew	eight	2015-08-15	20:20	nothing	0.440

`fmt()` : Set a column format with a formatter function

The `fmt()` function provides greater control in formatting raw data values than any of the specialized `fmt_*()` functions that are available in **gt**. Along with the `columns` and `rows` arguments that provide some precision in targeting data cells, the `fns` argument allows you to define one or more functions for manipulating the raw data.

EXAMPLE

Use `exibble` to create a **gt** table. Format the numeric values in the `num` column with a function supplied to the `fns` argument.

```
exibble %>%
  dplyr::select(-row, -group) %>%
  gt() %>%
  fmt(
    columns = num,
    fns = function(x) {
      paste0("'", x * 1000, "'")
    }
  )
```

num	char	fctr	date	time	datetime	currency
'111.1'	apricot	one	2015-01-15	13:35	2018-01-01 02:22	49.950
'2222'	banana	two	2015-02-15	14:40	2018-02-02 14:33	17.950
'33330'	coconut	three	2015-03-15	15:45	2018-03-03 03:44	1.390
'444400'	durian	four	2015-04-15	16:50	2018-04-04 15:55	65100.000
'5550000'	NA	five	2015-05-15	17:55	2018-05-05 04:00	1325.810
'NA'	fig	six	2015-06-15	NA	2018-06-06 16:11	13.255
'7.77e+08'	grapefruit	seven	NA	19:10	2018-07-07 05:22	NA
'8.88e+09'	honeydew	eight	2015-08-15	20:20	NA	0.440

`text_transform()` : Perform targeted text transformation with a function

Perform targeted text transformation with a function.

EXAMPLE

Use `exibble` to create a **gt** table. Transform the formatted text in the `num` and `currency` columns using a function within `text_transform()`, where `x` is a formatted vector of column values.

```
exibble %>%
  dplyr::select(num, char, currency) %>%
  dplyr::slice(1:4) %>%
  gt() %>%
  fmt_number(columns = num) %>%
  fmt_currency(columns = currency) %>%
  text_transform(
    locations = cells_body(columns = num),
    fn = function(x) {
      paste0(
        x, " (",
        dplyr::case_when(
          x > 20   ~ "large",
          x <= 20  ~ "small"),
        ")")
    }
  )
```

num	char	currency
0.11 (small)	apricot	$49.95
2.22 (small)	banana	$17.95
33.33 (large)	coconut	$1.39
444.40 (large)	durian	$65,100.00

`data_color()`: Set data cell colors using a palette or a color function

It's possible to add color to data cells according to their values. The `data_color()` function colors all rows of any `columns` supplied. There are two ways to define how cells are colored: (1) through the use of a supplied color palette, and (2) through use of a color mapping function available from the **scales** package. The first method colorizes cell data according to whether values are character or numeric. The second method provides more control over how cells are colored since we provide an explicit color function and thus other requirements such as bin counts, cut points, or a numeric domain.

EXAMPLES

Use `countrypops` to create a **gt** table. Apply a color scale to the `population` column with `scales::col_numeric`, four supplied colors, and a domain.

```
countrypops %>%
  dplyr::filter(country_name == "Mongolia") %>%
  dplyr::select(-contains("code")) %>%
  tail(10) %>%
  gt() %>%
  data_color(
    columns = population,
    colors = scales::col_numeric(
      palette = c("red", "orange", "green", "blue"),
      domain = c(0.2E7, 0.4E7)
    )
  )
```

country_name	year	population
Mongolia	2008	2628131
Mongolia	2009	2668289
Mongolia	2010	2712650
Mongolia	2011	2761516
Mongolia	2012	2814226
Mongolia	2013	2869107
Mongolia	2014	2923896
Mongolia	2015	2976877
Mongolia	2016	3027398
Mongolia	2017	3075647

Use `pizzaplace` to create a **gt** table. Apply colors from the `red_material` palette (in the `ggsci` pkg but more easily gotten from the `paletteer` package, info at `info_paletteer()`) to the `sold` and `income` columns. Set the `domain` of `scales::col_numeric()` to `NULL` will use the bounds of the available data as the domain.

```
pizzaplace %>%
  dplyr::filter(type %in% c("chicken", "supreme")) %>%
  dplyr::group_by(type, size) %>%
  dplyr::summarize(
    sold = dplyr::n(),
    income = sum(price)
  ) %>%
  gt(rowname_col = "size") %>%
  data_color(
    columns = c(sold, income),
    colors = scales::col_numeric(
      palette = paletteer::paletteer_d(
        palette = "ggsci::red_material"
      ) %>% as.character(),
      domain = NULL
    )
  )
```

```
## `summarise()` regrouping output by 'type' (override with `.groups` argument)
```

	sold	income
chicken		
L	4932	102339.0
M	3894	65224.5
S	2224	28356.0
supreme		
L	4564	94258.5
M	4046	66475.0
S	3377	47463.5