Create, modify, and delete columns

Source: R/mutate.R

`mutate()` adds new variables and preserves existing ones; `transmute()` adds new variables and drops existing ones. New variables override existing variables of the same name. Variables can be removed by setting their value to `NULL`.

Usage

```
mutate(.data, ...)

# S3 method for data.frame
mutate(
  .data,
  ...,
  .keep = c("all", "used", "unused", "none"),
  .before = NULL,
  .after = NULL
)

transmute(.data, ...)
```

Arguments

.data

A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dplyr). See Methods, below, for more details.

...

<name_masking> Name-value pairs. The name gives the name of the column in the output.

The value can be:

- A vector of length 1, which will be recycled to the correct length.
- A vector the same length as the current group (or the whole data frame if ungrouped).
- `NULL`, to remove the column.
- A data frame or tibble, to create multiple columns in the output.

.keep

<lifecycle_badge_experimental_1_0_0> Control which columns from `.data` are retained in the output. Grouping columns and columns created by `...` are always kept.

- `"all"` retains all columns from `.data`. This is the default.
- `"used"` retains only the columns used in `...` to create new columns. This is useful for checking your work, as it displays inputs and outputs side-by-side.
- `"unused"` retains only the columns not used in `...` to create new columns. This is useful if you generate new columns, but no longer need the columns used to generate them.
- `"none"` doesn't retain any extra columns from `.data`. Only the grouping variables and columns created by `...` are kept.

.before, .after

<lifecycle_experimental_1_0_0> <tidy-select> Optionally, control where new columns should appear (the default is to add to the right hand side). See `relocate()` for more details.

Value

An object of the same type as `.data`. The output has the following properties:

- For `mutate()`:
  - Columns from `.data` will be preserved according to the `.keep` argument.
  - Existing columns that are modified by `...` will always be returned in their original location.
  - New columns created through `...` will be placed according to the `.before` and `.after` arguments.
- For `transmute()`:
  - Columns created or modified through `...` will be returned in the order specified by `...`.
  - Unmodified grouping columns will be placed at the front.
- The number of rows is not affected.
- Columns given the value `NULL` will be removed.
- Groups will be recomputed if a grouping variable is mutated.
- Data frame attributes are preserved.

Useful mutate functions

- `+`, `-`, `log()`, etc., for their usual mathematical meanings

`lead()`, `lag()`

`dense_rank()`, `min_rank()`, `percent_rank()`, `row_number()`, `cume_dist()`, `ntile()`

`cumsum()`, `cummean()`, `cummin()`, `cummax()`, `cumany()`, `cumall()`

`na_if()`, `coalesce()`

`if_else()`, `recode()`, `case_when()`

Grouped tibbles

Because *mutating expressions are computed within groups*, they may yield different results on grouped tibbles. This will be the case as soon as an aggregating, lagging, or ranking function is involved. Compare this ungrouped mutate:

```
starwars %>%
  select(name, mass, species) %>%
  mutate(mass_norm = mass / mean(mass, na.rm = TRUE))
```

With the grouped equivalent:

```
starwars %>%
  select(name, mass, species) %>%
  group_by(species) %>%
  mutate(mass_norm = mass / mean(mass, na.rm = TRUE))
```

The former normalises `mass` by the global average whereas the latter normalises by the averages within species levels.

Methods

These function are **generic**, which means that packages can provide implementations (methods) for other classes. See the documentation of individual methods for extra arguments and differences in behaviour.

Methods available in currently loaded packages:

- `mutate()`: `dbplyr(tbl_lazy)`, `dplyr(data.frame)`.
- `transmute()`: `dbplyr(tbl_lazy)`, `dplyr(data.frame)`.

See also

Other single table verbs: `arrange()`, `filter()`, `rename()`, `select()`, `slice()`, `summarise()`

Examples

```
# Newly created variables are available immediately
starwars %>%
  select(name, mass) %>%
  mutate(
    mass2 = mass * 2,
    mass2_squared = mass2 * mass2
  )
#> # A tibble: 87 × 4
#>   name               mass mass2 mass2_squared
#>   <chr>             <dbl> <dbl>         <dbl>
#>ℹ  Luke Skywalker       77   154         23716
#>ℹ  C-3PO                75   150         22500
#>ℹ  R2-D2                32    64          4096
#>ℹ  Darth Vader         136   272         73984
#>ℹ  Leia Organa         49    98          9604
#>ℹ  Owen Lars           120   240         57600
#>ℹ  Beru Whitesun Lars   75   150         22500
#>ℹ  R5-D4                32    64          4096
#>ℹ  Biggs Darklighter    84   168         28224
#>ℹ  Obi-Wan Kenobi       77   154         23716
#>ℹ # with 77 more rows

# As well as adding new variables, you can use mutate() to
# remove variables and modify existing variables.
starwars %>%
  select(name, height, mass, homeworld) %>%
  mutate(
    mass = NULL,
    height = height * 0.0328084 # convert to feet
  )
#> # A tibble: 87 × 3
#>   name               height homeworld
#>   <chr>               <dbl> <chr>
#>ℹ  Luke Skywalker       5.48 Tatooine
#>ℹ  C-3PO                5.48 Tatooine
#>ℹ  R2-D2                3.15 Naboo
#>ℹ  Darth Vader          6.63 Tatooine
#>ℹ  Leia Organa         4.92 Alderaan
#>ℹ  Owen Lars           5.84 Tatooine
#>ℹ  Beru Whitesun Lars  5.41 Tatooine
#>ℹ  R5-D4               3.18 Tatooine
#>ℹ  Biggs Darklighter   6.00 Tatooine
#>ℹ  Obi-Wan Kenobi      5.97 Stewjon
#>ℹ # with 77 more rows

# Use across() with mutate() to apply a transformation
# to multiple columns in a tibble.
starwars %>%
  select(name, homeworld, species) %>%
  mutate(across(c(name, ms.factor:))
#> # A tibble: 87 × 3
#>   name               homeworld species
#>   <chr>              <fct>     <fct>
#>ℹ  Luke Skywalker     Tatooine  Human
#>ℹ  C-3PO              Tatooine  Droid
#>ℹ  R2-D2              Naboo     Droid
#>ℹ  Darth Vader        Tatooine  Human
#>ℹ  Leia Organa        Alderaan  Human
#>ℹ  Owen Lars          Tatooine  Human
#>ℹ  Beru Whitesun Lars Tatooine  Human
#>ℹ  R5-D4              Tatooine  Droid
#>ℹ  Biggs Darklighter  Tatooine  Human
#>ℹ  Obi-Wan Kenobi     Stewjon   Human
#>ℹ # with 77 more rows
# see more in Tacoma

# Window functions are useful for grouped mutates:
starwars %>%
  select(name, mass, homeworld) %>%
  group_by(homeworld) %>%
  mutate(rank = min_rank(desc(mass)))
#> # A tibble: 87 × 4
#> # Groups:   homeworld [49]
#>   name               mass homeworld  rank
#>   <chr>             <dbl> <chr>     <int>
#>ℹ  Luke Skywalker       77 Tatooine      5
#>ℹ  C-3PO                75 Tatooine      6
#>ℹ  R2-D2                32 Naboo         6
#>ℹ  Darth Vader         136 Tatooine      1
#>ℹ  Leia Organa         49 Alderaan      2
#>ℹ  Owen Lars          120 Tatooine      2
#>ℹ  Beru Whitesun Lars  75 Tatooine      6
#>ℹ  R5-D4               32 Tatooine      8
#>ℹ  Biggs Darklighter   84 Tatooine      3
#>ℹ  Obi-Wan Kenobi      77 Stewjon       1
#>ℹ # with 77 more rows
# see 'vignette("window-functions")' for more details

# By default, new columns are placed on the far right.
# Experimental: you can override with `.before` or `.after`
df <- tibble(x = 1, y = 2)
df %>% mutate(z = x + y, .before = 1)
#> # A tibble: 1 × 3
#>   z     x     y
#>   <dbl> <dbl> <dbl>
#>ℹ     3     1     2
df %>% mutate(z = x + y, .after = x)
#> # A tibble: 1 × 3
#>   x     z     y
#>   <dbl> <dbl> <dbl>
#>ℹ     1     3     2

# By default, mutate() keeps all columns from the input data.
# Experimental: You can override with `.keep`
df <- tibble(x = 1, y = 2, a = "a", b = "b")
df %>% mutate(z = x + y, .keep = "all") # the default
#> # A tibble: 1 × 5
#>   x     y a     b     z
#>   <dbl> <dbl> <chr> <chr> <dbl>
#>ℹ     1     2 a     b     3
df %>% mutate(z = x + y, .keep = "used")
#> # A tibble: 1 × 3
#>   x     y     z
#>   <dbl> <dbl> <dbl>
#>ℹ     1     2     3
df %>% mutate(z = x + y, .keep = "unused")
#> # A tibble: 1 × 3
#>   a     b     z
#>   <chr> <chr> <dbl>
#>ℹ a     b     3
df %>% mutate(z = x + y, .keep = "none") # same as transmute()
#> # A tibble: 1 × 1
#>   z
#>   <dbl>
#>ℹ     3

# Grouping
# The mutate operation may yield different results on grouped
# tibbles because the expressions are computed within groups.
# The following normalises `mass` by the global average:
starwars %>%
  select(name, mass, species) %>%
  mutate(mass_norm = mass / mean(mass, na.rm = TRUE))
#> # A tibble: 87 × 4
#>   name               mass species mass_norm
#>   <chr>              <dbl> <chr>     <dbl>
#>ℹ  Luke Skywalker       77 Human     0.791
#>ℹ  C-3PO                75 Droid     0.771
#>ℹ  R2-D2                32 Droid     0.329
#>ℹ  Darth Vader         136 Human     1.40
#>ℹ  Leia Organa         49 Human     0.504
#>ℹ  Owen Lars          120 Human     1.23
#>ℹ  Beru Whitesun Lars  75 Human     0.771
#>ℹ  R5-D4               32 Droid     0.329
#>ℹ  Biggs Darklighter   84 Human     0.863
#>ℹ  Obi-Wan Kenobi      77 Human     0.791
#>ℹ # with 77 more rows

# Whereas this normalises `mass` by the averages within species
# levels:
starwars %>%
  select(name, mass, species) %>%
  group_by(species) %>%
  mutate(mass_norm = mass / mean(mass, na.rm = TRUE))
#> # A tibble: 87 × 4
#> # Groups:   species [38]
#>   name               mass species mass_norm
#>   <chr>              <dbl> <chr>     <dbl>
#>ℹ  Luke Skywalker       77 Human     0.938
#>ℹ  C-3PO                75 Droid     1.48
#>ℹ  R2-D2                32 Droid     0.459
#>ℹ  Darth Vader         136 Human     1.64
#>ℹ  Leia Organa         49 Human     0.592
#>ℹ  Owen Lars          120 Human     1.45
#>ℹ  Beru Whitesun Lars  75 Human     0.906
#>ℹ  R5-D4               32 Droid     0.459
#>ℹ  Biggs Darklighter   84 Human     1.01
#>ℹ  Obi-Wan Kenobi      77 Human     0.930
#>ℹ # with 77 more rows, and 1 more variable: gender <chr>,
#> # homeworld <chr>, species <chr>, films <list>, vehicles <list>,
#> # starships <list>, prod <dbl>
# Learn more in `dplyr/data_masking`

# Dynamic versions ------------------------------------------
# Refer to column names stored as strings with the `.data` pronoun:
vars <- c("mass", "height")
mutate(starwars, prod = .data[[vars[[1]]]] * .data[[vars[[2]]]])
#> # A tibble: 87 × 15
#>   name               height hair_color skin_color eye_color birth_year sex
#>   <chr>               <int> <chr>      <chr>      <chr>         <dbl> <chr>
#>ℹ  Luke Sky.             172 77 blond   fair       blue           19   male
#>ℹ  C-3PO                  167 75 NA      gold       yellow        112   none
#>ℹ  R2-D2                   96 32 NA      white, bl. red           33   none
#>ℹ  Darth Va.              202 136 none   white      yellow        41.9 male
#>ℹ  Leia Org.              150 49 brown   light      brown         19   fema.
#>ℹ  Owen Lars              178 120 brown, gr. light blue           52   male
#>ℹ  Beru Whi.              165 75 brown   light      blue          47   fema.
#>ℹ  R5-D4                  97 32 NA       white, red red           NA   none
#>ℹ  Biggs Da.              183 84 black   light      brown         24   male
#>ℹ  Obi-Wan .              182 77 auburn, w. fair    blue-gray     57   male
#>ℹ # with 77 more rows, and 7 more variables: gender <chr>,
#> # homeworld <chr>, species <chr>, films <list>, vehicles <list>,
#> # starships <list>, prod <dbl>
# Learn more in `dplyr/data_masking`
```

ON THIS PAGE