Package 'tidytable'

July 16, 2020

Title Tidy Interface to 'data.table'

Version 0.5.3

Description Tidy interface to 'data.table'. 'rlang' compatible, which allows the user to build custom functions much like they would in the tidyverse.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Imports data.table, magrittr, rlang (>= 0.4.5), methods, tidyselect (>= 1.1.0), vctrs (>= 0.3.1), lifecycle (>= 0.2.0)

RoxygenNote 7.1.1

URL https://github.com/markfairbanks/tidytable

BugReports https://github.com/markfairbanks/tidytable/issues

Suggests testthat (>= 2.1.0), bit64, knitr, rmarkdown

NeedsCompilation no

Author Mark Fairbanks [aut, cre],
Tyson Barrett [ctb],
Ivan Leung [ctb],
Ross Kennedy [ctb],
Lionel Henry [ctb],
Matt Carlson [ctb]

Maintainer Mark Fairbanks <mark.t.fairbanks@gmail.com>

Repository CRAN

Date/Publication 2020-07-16 15:10:02 UTC

R topics documented:

`arrange.` *Arrange/reorder rows by variables*

Description

Order rows in ascending or descending order

Usage

```
arrange.(.df, ...)

dt_arrange(.df, ...)
```

Arguments

`.df`	A data.frame or data.table
`...`	Variables to arrange by

Examples

```
test_df <- data.table(
  a = c(1,2,3),
  b = c(4,5,6),
  c = c("a","a","b"))

test_df %>%
  arrange.(c, -a)
```

`as_tidytable` *Coerce an object to a data.table/tidytable*

Description

A tidytable object is simply a data.table with nice printing features.

Note that all tidytable functions automatically convert data.frames & data.tables to tidytables in the background. As such this function will rarely need to be used by the user.

Usage

```
as_tidytable(x)

as_dt(x)
```

Arguments

x	An R object

Examples

```
data.frame(x = 1:3) %>%
  as_tidytable()

data.frame(x = 1:3) %>%
  as_dt()
```

---

bind_rows. *Bind data.tables by row and column*

Description

Bind multiple data.tables into one row-wise or col-wise.

Usage

```
bind_rows.(..., .id = NULL, .use_names = TRUE, .fill = TRUE)

dt_bind_rows(..., .id = NULL, .use_names = TRUE, .fill = TRUE)

bind_cols.(...)

dt_bind_cols(...)
```

Arguments

...	data.tables or data.frames to bind
.id	If TRUE, an integer column is made as a group id
.use_names	If TRUE, makes sure column names align
.fill	If TRUE, fills missing columns with NA

Examples

```
df1 <- data.table(x = c(1,2,3), y = c(3,4,5))
df2 <- data.table(x = c(1,2,3), y = c(3,4,5))

df1 %>%
  bind_rows.(df2)

bind_rows.(list(df1, df2))

df1 %>%
  bind_cols.(df2)

bind_cols.(list(df1, df2))
```

case. *Case when*

Description

This function allows you to use multiple if/else statements in one call.

Note that this function is called differently than 'dplyr::case_when'. See examples.

Usage

```
case.(..., default = NA)

dt_case(..., default = NA)
```

Arguments

...	Sequence of condition/value designations
default	Default value. Set to NA by default.

Examples

```
library(data.table)

test_df <- tidytable(
  a = 1:10,
  b = 11:20,
  c = c(rep("a", 6), rep("b", 4)),
  d = c(rep("a", 4), rep("b", 6)))

test_df %>%
  mutate.(x = case.(b < 13, 3,
                    a > 4, 2,
                    default = 10))
test_df %>%
  mutate.(x = case.(c == "a", "a",
                    default = d))
```

complete. *Complete a data.table with missing combinations of data*

Description

Turns implicit missing values into explicit missing values.

Usage

```
complete.(.df, ..., .fill = list())
```

Arguments

.df	A data.frame or data.table
...	Columns to expand
.fill	A named list of values to fill NAs with.

Examples

```
test_df <- data.table(x = 1:2, y = 1:2, z = 3:4)

test_df %>%
  complete.(x, y)

test_df %>%
  complete.(x, y, .fill = list(z = 10))
```

---

count. *Count observations by group*

Description

Returns row counts of the dataset. If bare column names are provided, `count.()` returns counts by group.

Usage

```
count.(.df, ...)

dt_count(.df, ...)
```

Arguments

.df	A data.frame or data.table
...	Columns to group by. `tidyselect` compatible.

Examples

```
test_df <- data.table(
  x = 1:3,
  y = 4:6,
  z = c("a", "a", "b"))

test_df %>%
  count.()

test_df %>%
  count.(z)

test_df %>%
  count.(where(is.character))
```

crossing. *Create a data.table from all unique combinations of inputs*

Description

`crossing.()` is similar to `expand_grid.()` but de-duplicates and sorts its inputs.

Usage

```
crossing.(..., .name_repair = "check_unique")
```

Arguments

`...`	Variables to get unique combinations of
`.name_repair`	Treatment of problematic names. See `?vctrs::vec_as_names` for options/details

Examples

```
x <- 1:2
y <- 1:2

crossing.(x, y)

crossing.(stuff = x, y)
```

desc. *Deprecated*

Description

This function is deprecated.

Arrange in descending order. Can be used inside of `arrange.()`

Usage

```
desc.(x)
```

Arguments

`x`	Variable to arrange in descending order

Examples

```
test_df <- data.table(
  a = c(1,2,3),
  b = c(4,5,6),
  c = c("a","a","b"))

test_df %>%
  arrange.(c, -a)
```

`distinct.` *Select distinct/unique rows*

Description

Retain only unique/distinct rows from an input df.

Usage

```
distinct.(.df, ..., .keep_all = FALSE)

dt_distinct(.df, ..., .keep_all = FALSE)
```

Arguments

`.df`	A data.frame or data.table
`...`	Columns to select before determining uniqueness. If omitted, will use all columns. `tidyselect` compatible.
`.keep_all`	Only relevant if columns are provided to ... arg. This keeps all columns, but only keeps the first row of each distinct values of columns provided to ... arg.

Examples

```
test_df <- tidytable(
  x = 1:3,
  y = 4:6,
  z = c("a", "a", "b"))

test_df %>%
  distinct.()

test_df %>%
  distinct.(z)
```

drop_na. *Drop rows containing missing values*

Description

Drop rows containing missing values

Usage

```
drop_na.(.df, ...)

dt_drop_na(.df, ...)
```

Arguments

`.df`	A data.frame or data.table
`...`	Optional: A selection of columns. If empty, all variables are selected. `tidyselect` compatible.

Examples

```
df <- data.table(
  x = c(1,2,NA),
  y = c("a",NA,"b"))

df %>%
  drop_na.()

df %>%
  drop_na.(x)

df %>%
  drop_na.(where(is.numeric))
```

dt *Pipeable data.table call*

Description

Pipeable data.table call

Note: This function does not use data.table's modify-by-reference

Usage

```
dt(.df, ...)
```

Arguments

.df	A data.frame or data.table
...	Arguments passed to data.table call. See ?data.table:::‘[.df.table‘

Examples

```
test_df <- tidytable(
  x = c(1,2,3),
  y = c(4,5,6),
  z = c("a", "a", "b"))

test_df %>%
  dt(, ':='(double_x = x * 2)) %>%
  dt(order(-double_x))
```

---

expand. *Expand a data.table to use all combinations of values*

Description

Generates all combinations of variables found in a dataset.

`expand.()` is useful in conjunction with joins:

- use with `right_join.()` to convert implicit missing values to explicit missing values
- use with `anti_join.()` to find out which combinations are missing

Usage

```
expand.(.df, ..., .name_repair = "check_unique")
```

Arguments

.df	A data.frame or data.table
...	Columns to get combinations of
.name_repair	Treatment of problematic names. See `?vctrs::vec_as_names` for options/details

Examples

```
test_df <- tidytable(x = 1:2, y = 1:2)

test_df %>%
  expand.(x, y)
```

expand_grid. *Create a data.table from all combinations of inputs*

Description

Create a data.table from all combinations of inputs

Usage

```
expand_grid.(..., .name_repair = "check_unique")
```

Arguments

...	Variables to get combinations of
.name_repair	Treatment of problematic names. See ?vctrs::vec_as_names for options/details

Examples

```
x <- 1:2
y <- 1:2

expand_grid.(x, y)

expand_grid.(stuff = x, y)
```

fill. *Fill in missing values with previous or next value*

Description

Fills missing values in the selected columns using the next or previous entry. Can be done by group.

Supports tidyselect

Usage

```
fill.(
  .df,
  ...,
  .direction = c("down", "up", "downup", "updown"),
  .by = NULL,
  by = NULL
)

dt_fill(
  .df,
```

```
  ...,
  .direction = c("down", "up", "downup", "updown"),
  .by = NULL,
  by = NULL
)
```

Arguments

.df	A data.frame or data.table
...	A selection of columns. `tidyselect` compatible.
.direction	Direction in which to fill missing values. Currently "down" (the default), "up", "downup" (first down then up), or "updown" (first up and then down)
.by	Columns to group by when filling should be done by group
by	This argument has been renamed to .by and is deprecated

Examples

```
test_df <- tidytable(
  x = c(NA, NA, NA, 4:10),
  y = c(1:6, NA, 8, NA, 10),
  z = c(rep("a", 8), rep("b", 2)))

test_df %>%
  fill.(x, y, .by = z)

test_df %>%
  fill.(x, y, .by = z, .direction = "downup")
```

---

filter. *Filter rows on one or more conditions*

Description

Filters a dataset to choose rows where conditions are true.

Usage

```
filter.(.df, ..., .by = NULL, by = NULL)

dt_filter(.df, ..., .by = NULL, by = NULL)
```

Arguments

.df	A data.frame or data.table
...	Conditions to filter by
.by	Columns to group by if filtering with a summary function
by	This argument has been renamed to .by and is deprecated

Examples

```
test_df <- tidytable(
  a = c(1,2,3),
  b = c(4,5,6),
  c = c("a","a","b"))

test_df %>%
  filter.(a >= 2, b >= 4)

test_df %>%
  filter.(b <= mean(b), .by = c)
```

`get_dummies.` *Convert character and factor columns to dummy variables*

Description

Convert character and factor columns to dummy variables

Usage

```
get_dummies.(
  .df,
  cols = c(where(is.character), where(is.factor)),
  prefix = TRUE,
  prefix_sep = "_",
  drop_first = FALSE,
  dummify_na = TRUE
)

dt_get_dummies(
  .df,
  cols = c(where(is.character), where(is.factor)),
  prefix = TRUE,
  prefix_sep = "_",
  drop_first = FALSE,
  dummify_na = TRUE
)
```

Arguments

`.df`	A data.frame or data.table
`cols`	A single column or a vector of unquoted columns to dummify. Defaults to all character & factor columns using `c(is.character,is.factor)`. `tidyselect` compatible.
`prefix`	TRUE/FALSE - If TRUE, a prefix will be added to new column names
`prefix_sep`	Separator for new column names

drop_first	TRUE/FALSE - If TRUE, the first dummy column will be dropped
dummify_na	TRUE/FALSE - If TRUE, NAs will also get dummy columns

Examples

```
test_df <- tidytable(
  col1 = c("a", "b", "c", NA),
  col2 = as.factor(c("a", "b", NA, "d")),
  var1 = rnorm(4,0,1))

# Automatically does all character/factor columns
test_df %>%
  get_dummies.()

# Can select one column
test_df %>%
  get_dummies.(col1)

# Can select one or multiple columns in a vector of unquoted column names
test_df %>%
  get_dummies.(c(col1, col2))

# Can drop certain columns using
test_df %>%
  get_dummies.(c(where(is.character), -col2))

test_df %>%
  get_dummies.(prefix_sep = ".", drop_first = TRUE)

test_df %>%
  get_dummies.(c(col1, col2), dummify_na = FALSE)
```

---

group_split. *Split data frame by groups*

Description

Split data frame by groups. Returns a list.

Usage

```
group_split.(.df, ..., .keep = TRUE)

dt_group_split(.df, ..., .keep = TRUE)
```

Arguments

.df	A data.frame or data.table
...	Columns to group and split by. `tidyselect` compatible.
.keep	Should the grouping columns be kept

Examples

```
test_df <- tidytable(
  a = 1:5,
  b = 1:5,
  c = c("a","a","a","b","b"),
  d = c("a","a","a","b","b"))

test_df %>%
  group_split.(c, d)

test_df %>%
  group_split.(c, d, .keep = FALSE)
```

---

ifelse. *Vectorized if*

Description

`ifelse.()` utilizes `data.table::fifelse()` in the background, but automatically converts NAs to their proper type

Usage

```
ifelse.(conditions, true, false, na = NA)

dt_ifelse(conditions, true, false, na = NA)
```

Arguments

`conditions`	Conditions to test on
`true`	Values to return if conditions evaluate to TRUE
`false`	Values to return if conditions evaluate to FALSE
`na`	Value to return if an element of test is NA.

Examples

```
x <- 1:5
ifelse.(x > 2, 2, 0)

# Can also be used inside of mutate.()
test_df <- data.table(x = x)

test_df %>%
  mutate.(new_col = ifelse.(x > 2, 2, 1))
```

inv_gc *Run invisible garbage collection*

Description

Run garbage collection without the ‘gc()‘ output. Can also be run in the middle of a long pipe chain. Useful for large datasets or when using parallel processing.

Usage

```
inv_gc(x)
```

Arguments

x	Optional. If missing runs ‘gc()‘ silently. Else returns the same object unaltered.

Examples

```
# Can be run with no input
inv_gc()

df <- tidytable(col1 = 1, col2 = 2)

# Or can be used in the middle of a pipe chain (object is unaltered)
df %>%
  filter.(col1 < 2, col2 < 4) %>%
  inv_gc() %>%
  select.(col1)
```

is_tidytable *Test if the object is a tidytable*

Description

This function returns TRUE for tidytables or subclasses of tidytables, and FALSE for all other objects.

Usage

```
is_tidytable(x)
```

Arguments

x	An object

Examples

```
dt <- data.table(x = 1)

is_tidytable(dt) # Returns FALSE

df <- tidytable(x = 1)

is_tidytable(df) # Returns TRUE
```

`leads.` *Lead and Lag*

Description

Find the "next" or "previous" values in a vector. Useful for comparing values ahead of or behind the current values.

Usage

```
leads.(x, n = 1L, default = NA)

lags.(x, n = 1L, default = NA)
```

Arguments

`x`	a vector of values
`n`	a positive integer of length 1, giving the number of positions to lead or lag by
`default`	value used for non-existent rows. Defaults to NA.

Examples

```
x <- 1:5

leads.(x, 1)
lags.(x, 1)

test_df <- tidytable(x = 1:5)

test_df %>%
  mutate.(lag_x = lags.(x))
```

`left_join.` *Join two data.tables together*

Description

Join two data.tables together

Usage

```
left_join.(x, y, by = NULL)

inner_join.(x, y, by = NULL)

right_join.(x, y, by = NULL)

full_join.(x, y, by = NULL, suffix = c(".x", ".y"))

anti_join.(x, y, by = NULL)

dt_left_join(x, y, by = NULL)

dt_inner_join(x, y, by = NULL)

dt_right_join(x, y, by = NULL)

dt_full_join(x, y, by = NULL, suffix = c(".x", ".y"))

dt_anti_join(x, y, by = NULL)
```

Arguments

`x`	A data.frame or data.table
`y`	A data.frame or data.table
`by`	A character vector of variables to join by. If NULL, the default, the join will do a natural join, using all variables with common names across the two tables.
`suffix`	Append created for duplicated column names when using ‘full_join.()‘

Value

A data.table

Examples

```
df1 <- data.table(x = c("a","a","a","b","b"), y = 1:5)
df2 <- data.table(x = c("a","b"), z = 1:2)

df1 %>% left_join.(df2)
```

```
df1 %>% inner_join.(df2)
df1 %>% right_join.(df2)
df1 %>% full_join.(df2)
df1 %>% anti_join.(df2)
```

`map.` *Apply a function to each element of a vector*

Description

The map functions transform their input by applying a function to each element and returning a list/vector/data.table.

- `map.()` returns a list
- _lgl.(), _int.(), _dbl.(),_chr.(), _df.() variants return their specified type
- _dfr.() & _dfc.() Return all data frame results combined utilizing row or column binding

Usage

```
map.(.x, .f, ...)

dt_map(.x, .f, ...)

map_lgl.(.x, .f, ...)

dt_map_lgl(.x, .f, ...)

map_int.(.x, .f, ...)

dt_map_int(.x, .f, ...)

map_dbl.(.x, .f, ...)

dt_map_dbl(.x, .f, ...)

map_chr.(.x, .f, ...)

dt_map_chr(.x, .f, ...)

map_dfc.(.x, .f, ...)

dt_map_dfc(.x, .f, ...)

map_dfr.(.x, .f, ..., .id = NULL)

dt_map_dfr(.x, .f, ..., .id = NULL)
```

```
map_df.(.x, .f, ..., .id = NULL)

dt_map_df(.x, .f, ...)

walk.(.x, .f, ...)

map2.(.x, .y, .f, ...)

dt_map2(.x, .y, .f, ...)

map2_lgl.(.x, .y, .f, ...)

dt_map2_lgl(.x, .y, .f, ...)

map2_int.(.x, .y, .f, ...)

dt_map2_int(.x, .y, .f, ...)

map2_dbl.(.x, .y, .f, ...)

dt_map2_dbl(.x, .y, .f, ...)

map2_chr.(.x, .y, .f, ...)

dt_map2_chr(.x, .y, .f, ...)

map2_dfc.(.x, .y, .f, ...)

dt_map2_dfc(.x, .y, .f, ...)

map2_dfr.(.x, .y, .f, ..., .id = NULL)

dt_map2_dfr(.x, .y, .f, ..., .id = NULL)

map2_df.(.x, .y, .f, ..., .id = NULL)

dt_map2_df(.x, .y, .f, ..., .id = NULL)
```

Arguments

`.x`	A list or vector
`.f`	A function
`...`	Other arguments to pass to a function
`.id`	Whether `map_dfr.()` should add an id column to the finished dataset
`.y`	A list or vector

Examples

```
map.(c(1,2,3), ~ .x + 1)

map_dbl.(c(1,2,3), ~ .x + 1)

map_chr.(c(1,2,3), as.character)
```

`mutate.` *Mutate*

Description

Add new columns or modify existing ones

Usage

```
mutate.(.df, ..., .by = NULL, by = NULL)

dt_mutate(.df, ..., .by = NULL, by = NULL)
```

Arguments

`.df`	A data.frame or data.table
`...`	Columns to add/modify
`.by`	Columns to group by
`by`	This argument has been renamed to .by and is deprecated

Examples

```
test_df <- data.table(
  a = c(1,2,3),
  b = c(4,5,6),
  c = c("a","a","b"))

test_df %>%
  mutate.(double_a = a * 2,
          a_plus_b = a + b)

test_df %>%
  mutate.(double_a = a * 2,
          avg_a = mean(a),
          .by = c)
```

`mutate_across.` *Mutate multiple columns simultaneously*

Description

Mutate multiple columns simultaneously.

Usage

```
mutate_across.(.df, .cols = everything(), .fns, ..., .by = NULL, by = NULL)

dt_mutate_across(.df, .cols = everything(), .fns, ..., .by = NULL, by = NULL)
```

Arguments

`.df`	A data.frame or data.table
`.cols`	vector `c()` of unquoted column names. `tidyselect` compatible.
`.fns`	Functions to pass. Can pass a list of functions.
`...`	Other arguments for the passed function
`.by`	Columns to group by
`by`	This argument has been renamed to .by and is deprecated

Examples

```
test_df <- data.table(
  x = c(1,1,1),
  y = c(2,2,2),
  z = c("a", "a", "b"))

test_df %>%
  mutate_across.(where(is.numeric), as.character)

test_df %>%
  mutate_across.(c(x, y), ~ .x * 2)

test_df %>%
  mutate_across.(everything(), as.character)

test_df %>%
  mutate_across.(c(x, y), list(new = ~ .x * 2,
                               another = ~ .x + 7))
```

mutate_if. *Deprecated mutate helpers*

Description

These helpers have been deprecated. Please use `mutate_across.()`

Usage

```
mutate_if.(.df, .predicate, .funs, ..., .by = NULL, by = NULL)

mutate_at.(.df, .vars, .funs, ..., .by = NULL, by = NULL)

mutate_all.(.df, .funs, ..., .by = NULL, by = NULL)

dt_mutate_if(.df, .predicate, .funs, ..., .by = NULL, by = NULL)

dt_mutate_at(.df, .vars, .funs, ..., .by = NULL, by = NULL)

dt_mutate_all(.df, .funs, ..., .by = NULL, by = NULL)
```

Arguments

`.df`	A data.frame or data.table
`.predicate`	predicate for `mutate_if.()` to use
`.funs`	Functions to pass. Can pass a list of functions.
`...`	Other arguments for the passed function
`.by`	Columns to group by
`by`	This argument has been renamed to .by and is deprecated
`.vars`	vector `c()` of bare column names for `mutate_at.()` to use

Examples

```
test_df <- data.table(
  x = c(1,1,1),
  y = c(2,2,2),
  z = c("a", "a", "b"))

test_df %>%
  mutate_across.(where(is.numeric), as.character)

test_df %>%
  mutate_across.(c(x, y), ~ .x * 2)

test_df %>%
  mutate_across.(everything(), as.character)
```

```
test_df %>%
  mutate_across.(c(x, y), list(new = ~ .x * 2,
                               another = ~ .x + 7))
```

`n.` *Number of observations in each group*

Description

Helper function that can be used to find counts by group.

Can be used inside ‘summarize.()‘, ‘mutate.()‘, & ‘filter.()‘

Usage

```
n.()

dt_n()
```

Examples

```
test_df <- data.table(
  x = c(1,2,3),
  y = c(4,5,6),
  z = c("a","a","b"))

test_df %>%
  summarize.(count = n.(),
             .by = z)

test_df %>%
  mutate.(count = n.())
```

`nest_by.` *Nest data.tables*

Description

Nest data.tables by group

Usage

```
nest_by.(.df, ..., .key = "data", .keep = FALSE)

dt_group_nest(.df, ..., .key = "data", .keep = FALSE)
```

Arguments

.df	A data.frame or data.table
...	Columns to group by. If empty nests the entire data.table. tidyselect compatible.
.key	Name of the new column created by nesting.
.keep	Should the grouping columns be kept in the list column.

Examples

```
test_df <- data.table(
  a = 1:10,
  b = 11:20,
  c = c(rep("a", 6), rep("b", 4)),
  d = c(rep("a", 4), rep("b", 6)))

test_df %>%
  nest_by.()

test_df %>%
  nest_by.(c, d)

test_df %>%
  nest_by.(where(is.character))

test_df %>%
  nest_by.(c, d, .keep = TRUE)
```

---

pivot_longer. *Pivot data from wide to long*

Description

pivot_wider.() "widens" data, increasing the number of columns and decreasing the number of rows. The inverse transformation is pivot_longer.(). Syntax based on the tidyr equivalents.

Usage

```
pivot_longer.(
  .df,
  cols = everything(),
  names_to = "name",
  values_to = "value",
  values_drop_na = FALSE,
  ...
)

dt_pivot_longer(
```

```
  .df,
  cols = everything(),
  names_to = "name",
  values_to = "value",
  values_drop_na = FALSE,
  ...
)
```

Arguments

.df	The data table to pivot longer
cols	Vector of bare column names. Can add/drop columns. tidyselect compatible.
names_to	Name of the new "names" column. Must be a string.
values_to	Name of the new "values" column. Must be a string.
values_drop_na	If TRUE, rows will be dropped that contain NAs.
...	Additional arguments to pass to melt.df.table()

Examples

```
test_df <- data.table(
  x = c(1,2,3),
  y = c(4,5,6),
  z = c("a", "b", "c"))

test_df %>%
  pivot_longer.(c(x, y))

test_df %>%
  pivot_longer.(cols = -z, names_to = "stuff", values_to = "things")
```

---

pivot_wider. *Pivot data from long to wide*

Description

pivot_wider.() "widens" data, increasing the number of columns and decreasing the number of rows. The inverse transformation is pivot_longer.(). Syntax based on the tidyr equivalents.

Usage

```
pivot_wider.(
  .df,
  names_from = name,
  values_from = value,
  id_cols = NULL,
  names_sep = "_",
  values_fn = NULL
```

```
)

dt_pivot_wider(
  .df,
  names_from = name,
  values_from = value,
  id_cols = NULL,
  names_sep = "_",
  values_fn = NULL
)
```

Arguments

.df	the data table to widen
names_from	A pair of arguments describing which column (or columns) to get the name of the output column (name_from), and which column (or columns) to get the cell values from (values_from). tidyselect compatible.
values_from	A pair of arguments describing which column (or columns) to get the name of the output column (name_from), and which column (or columns) to get the cell values from (values_from). tidyselect compatible.
id_cols	A set of columns that uniquely identifies each observation. Defaults to all columns in the data table except for the columns specified in names_from and values_from. Typically used when you have additional variables that is directly related. tidyselect compatible.
names_sep	the separator between the names of the columns
values_fn	Should the data be aggregated before casting? If the formula doesn't identify a single observation for each cell, then aggregation defaults to length with a message.

Examples

```
test_df <- data.table(
  z = rep(c("a", "b", "c"), 2),
  stuff = c(rep("x", 3), rep("y", 3)),
  things = 1:6)

test_df %>%
  pivot_wider.(names_from = stuff, values_from = things)

test_df %>%
  pivot_wider.(names_from = stuff, values_from = things, id_cols = z)
```

pull. *Pull out a single variable*

Description

Pull a single variable from a data.table as a vector.

Usage

```
pull.(.df, var = NULL)

dt_pull(.df, var = NULL)
```

Arguments

.df	A data.frame or data.table
var	The column to pull from the data.table. If NULL, pulls the last column.

Examples

```
test_df <- data.table(
  x = c(1,2,3),
  y = c(4,5,6))

test_df %>%
  pull.(y)
```

relocate. *Relocate a column to a new position*

Description

Move a column or columns to a new position

Usage

```
relocate.(.df, ..., .before = NULL, .after = NULL)

dt_relocate(.df, ..., .before = NULL, .after = NULL)
```

Arguments

.df	A data.frame or data.table
...	A selection of columns to move. ‘tidyselect‘ compatible.
.before	Column to move selection before
.after	Column to move selection after

Examples

```
test_df <- data.table(
  a = 1:5,
  b = 1:5,
  c = c("a","a","a","b","b"),
  d = c("a","a","a","b","b"))

test_df %>%
  relocate.(c, .before = b)

test_df %>%
  relocate.(a, b, .after = c)

test_df %>%
  relocate.(where(is.numeric), .after = c)
```

`rename.` *Rename variables by name*

Description

Rename variables from a data.table.

Usage

```
rename.(.df, ...)

dt_rename(.df, ...)
```

Arguments

`.df`	A data.frame or data.table
`...`	Rename expression like dplyr::rename()

Examples

```
dt <- data.table(x = c(1,2,3), y = c(4,5,6))

dt %>%
  rename.(new_x = x,
          new_y = y)
```

`rename_all.` *Deprecated rename helpers*

Description

These helpers have been deprecated. Please use `rename_with.()`

Usage

```
rename_all.(.data, .fun, ...)

rename_at.(.data, .vars, .fun, ...)

rename_across.(.data, .cols, .fun, ...)

rename_if.(.data, .predicate, .fun, ...)

dt_rename_across(.data, .cols, .fun, ...)

dt_rename_all(.data, .fun, ...)

dt_rename_if(.data, .predicate, .fun, ...)

dt_rename_at(.data, .vars, .fun, ...)
```

Arguments

`.data`	A data.frame or data.table
`.fun`	Function to pass
`...`	Other arguments for the passed function
`.vars`	vector `c()` of bare column names for `rename_at.()` to use
`.cols`	vector `c()` of bare column names for `rename_across.()` to use
`.predicate`	Predicate to pass to `rename_if.()`

Examples

```
test_df <- data.table(
  x = 1,
  y = 2,
  double_x = 2,
  double_y = 4)

test_df %>%
  rename_with.(~ sub("x", "stuff", .x))

test_df %>%
  rename_with.(~ sub("x", "stuff", .x), .cols = c(x, double_x))
```

rename_with. *Rename multiple columns*

Description

Rename multiple columns with the same transformation

Usage

```
rename_with.(.df, .fn, .cols = everything(), ...)

dt_rename_with(.df, .fn, .cols = everything(), ...)
```

Arguments

.df	A data.table or data.frame
.fn	Function to transform the names with.
.cols	Columns to rename. Defaults to all columns. tidyselect compatible.
...	Other parameters to pass to the function

Examples

```
test_df <- data.table(
  x = 1,
  y = 2,
  double_x = 2,
  double_y = 4)

test_df %>%
  rename_with.(toupper)

test_df %>%
  rename_with.(~ sub("x", "stuff", .x))

test_df %>%
  rename_with.(~ sub("x", "stuff", .x), .cols = c(x, double_x))
```

replace_na. *Replace missing values*

Description

Replace NAs with specified values

Usage

```
replace_na.(.x, replace = NA)

dt_replace_na(.x, replace = NA)
```

Arguments

.x	A data.frame/data.table or a vector
replace	If .x is a data frame, a list() of replacement values for specified columns. If .x is a vector, a single replacement value.

Examples

```
test_df <- data.table(
  x = c(1, 2, NA),
  y = c(NA, 1, 2))

# Using replace_na.() inside mutate()
test_df %>%
  mutate.(x = replace_na.(x, 5))

# Using replace_na.() on a data frame
test_df %>%
  replace_na.(list(x = 5, y = 0))
```

---

row_number. *Return row number*

Description

This function is designed to work inside of mutate.()

Usage

```
row_number.()

dt_row_number()
```

Examples

```
test_df <- data.table(x = c(1,1,1))

test_df %>%
  mutate.(row = row_number.())
```

`select.` *Select or drop columns*

Description

Select or drop columns from a data.table

Usage

```
select.(.df, ...)

dt_select(.df, ...)
```

Arguments

`.df`	A data.frame or data.table
`...`	Columns to select or drop. Use named arguments, e.g. new_name = old_name, to rename selected variables. `tidyselect` compatible.

Examples

```
test_df <- data.table(
  x = c(1,1,1),
  y = c(4,5,6),
  double_x = c(2,2,2),
  z = c("a","a","b"))

test_df %>%
  select.(x, y)

test_df %>%
  select.(x:z)

test_df %>%
  select.(-y, -z)

test_df %>%
  select.(starts_with("x"), z)

test_df %>%
  select.(where(is.character), x)

test_df %>%
  select.(stuff = x, y)
```

`separate.` *Separate a character column into multiple columns*

Description

Separates a single column into multiple columns using a user supplied separator or regex.

If a separator is not supplied one will be automatically detected.

Note: Using automatic detection or regex will be slower than simple separators such as "," or ".".

Usage

```
separate.(.df, col, into, sep = "[^[:alnum:]]+", remove = TRUE, ...)

dt_separate(.df, col, into, sep = "[^[:alnum:]]+", remove = TRUE, ...)
```

Arguments

`.df`	A data.frame or data.table
`col`	The column to split into multiple columns
`into`	New column names to split into. A character vector.
`sep`	Separator to split on. Can be specified or detected automatically
`remove`	If TRUE, remove the input column from the output data.table
`...`	Further argument to pass to data.table::tstrsplit

Examples

```
test_df <- data.table(x = c("a", "a.b", "a.b", NA))

# "sep" can be automatically detected (slower)
test_df %>%
  separate.(x, into = c("c1", "c2"))

# Faster if "sep" is provided
test_df %>%
  separate.(x, into = c("c1", "c2"), sep = ".")
```

separate_rows. *Separate a collapsed column into multiple rows*

Description

If a column contains observations with multiple delimited values, separate them each into their own row.

Usage

```
separate_rows.(.df, ..., sep = "[^[:alnum:].]+", convert = FALSE)
```

Arguments

.df	A data.frame or data.table
...	Columns to separate across multiple rows. ‘tidyselect‘ compatible
sep	Separator delimiting collapsed values
convert	If TRUE, runs ‘type.convert()‘ on the resulting column. Useful if the resulting column should be type integer/double.

Examples

```
test_df <- data.table(
  x = 1:3,
  y = c("a", "d,e,f", "g,h"),
  z = c("1", "2,3,4", "5,6")
)

separate_rows.(test_df, y, z)

separate_rows.(test_df, y, z, convert = TRUE)
```

slice. *Choose rows by position*

Description

Choose rows by their ordinal position in a data.table. Grouped data.tables use the ordinal position within the group.

Usage

```
slice.(.df, rows = 1:5, .by = NULL, by = NULL)

slice_head.(.df, n = 5, .by = NULL, by = NULL)

slice_tail.(.df, n = 5, .by = NULL, by = NULL)

slice_max.(.df, order_by, n = 1, .by = NULL, by = NULL)

slice_min.(.df, order_by, n = 1, .by = NULL, by = NULL)

dt_slice(.df, rows = 1:5, .by = NULL, by = NULL)

dt_slice_head(.df, n = 5, .by = NULL, by = NULL)

dt_slice_tail(.df, n = 5, .by = NULL, by = NULL)

dt_slice_min(.df, order_by, n = 1, .by = NULL, by = NULL)

dt_slice_max(.df, order_by, n = 1, .by = NULL, by = NULL)
```

Arguments

.df	A data.frame or data.table
rows	Integer row values. Provide either positive values to keep, or negative values to drop. The values provided must be either all positive or all negative.
.by	Columns to group by
by	This argument has been renamed to .by and is deprecated
n	Number of rows to grab
order_by	Variable to arrange by

Examples

```
test_df <- data.table(
  x = c(1,2,3,4),
  y = c(4,5,6,7),
  z = c("a","a","a","b"))

test_df %>%
  slice.(1:4)

test_df %>%
  slice.(1, .by = z)

test_df %>%
  slice_head.(1, .by = z)

test_df %>%
```

```
  slice_tail.(1, .by = z)

test_df %>%
  slice_max.(order_by = x, .by = z)

test_df %>%
  slice_min.(order_by = y, .by = z)
```

`starts_with.` *Select helpers*

Description

Please note these functions are leftover from before `tidytable` used `tidyselect`. You can/should use the normal `tidyselect` helpers.

These functions allow you to select variables based on their names.

- `any_of.()`: Select using a character vector
- `contains.()`: Contains a literal string or regex match
- `everything.()`: Matches all variables
- `starts_with.()`: Starts with a prefix
- `ends_with.()`: Ends with a suffix

Usage

```
starts_with.(match, ignore.case = TRUE, vars = peek_vars(fn = "starts_with"))

dt_starts_with(match, ignore.case = TRUE, vars = peek_vars(fn = "starts_with"))

contains.(match, ignore.case = TRUE, vars = peek_vars(fn = "contains"))

dt_contains(match, ignore.case = TRUE, vars = peek_vars(fn = "contains"))

ends_with.(match, ignore.case = TRUE, vars = peek_vars(fn = "ends_with"))

dt_ends_with(match, ignore.case = TRUE, vars = peek_vars(fn = "ends_with"))

everything.(vars = peek_vars(fn = "everything"))

dt_everything(vars = peek_vars(fn = "everything"))

any_of.(x, ..., vars = peek_vars(fn = "any_of"))

dt_any_of(x, ..., vars = peek_vars(fn = "any_of"))
```

Arguments

match	A character vector. If length > 1, the union of the matches is taken.
ignore.case	If TRUE, the default, ignores case when matching names.
vars	A character vector of variable names. When called from inside selecting functions like select.() these are automatically set to the names of the table.
x	An index vector of names or locations.
...	These dots are for future extensions and must be empty.

Examples

```
test_df <- tidytable(
  x = 1,
  y = 2,
  double_x = 2,
  double_y = 4)

test_df %>%
  select.(starts_with("x"))

test_df %>%
  select.(ends_with("y"))
```

---

summarize. *Aggregate data using summary statistics*

Description

Aggregate data using summary statistics such as mean or median. Can be calculated by group.

Usage

```
summarize.(.df, ..., .by = NULL, by = NULL)

summarise.(.df, ..., .by = NULL, by = NULL)

dt_summarise(.df, ..., .by = NULL, by = NULL)

dt_summarize(.df, ..., .by = NULL, by = NULL)
```

Arguments

.df	A data.frame or data.table
...	Aggregations to perform
.by	Columns to group by. • A single column can be passed with by = d.

- Multiple columns can be passed with by = c(c,d)
- tidyselect can be used:
  - Single predicate: by = where(is.character)
  - Multiple predicates: by = c(where(is.character),where(is.factor))
  - A combination of predicates and column names: by = c(where(is.character),b)

by This argument has been renamed to .by and is deprecated

Examples

```
test_df <- data.table(
  a = c(1,2,3),
  b = c(4,5,6),
  c = c("a","a","b"),
  d = c("a","a","b"))

test_df %>%
  summarize.(avg_a = mean(a),
             max_b = max(b),
             .by = c)

test_df %>%
  summarize.(avg_a = mean(a),
             .by = c(c, d))
```

---

summarize_across. *Summarize multiple columns*

Description

Summarize multiple columns simultaneously

Usage

```
summarize_across.(.df, .cols = everything(), .fns, ..., .by = NULL, by = NULL)

summarise_across.(.df, .cols = everything(), .fns, ..., .by = NULL, by = NULL)
```

Arguments

.df	A data.frame or data.table
.cols	vector c() of unquoted column names. tidyselect compatible.
.fns	Functions to pass. Can pass a list of functions.
...	Other arguments for the passed function
.by	Columns to group by
by	This argument has been renamed to .by and is deprecated

Examples

```
test_df <- data.table(a = 1:3,
                      b = 4:6,
                      z = c("a", "a", "b"))

# Single function
test_df %>%
  summarize_across.(c(a, b), mean, na.rm = TRUE)

# Single function using purrr style interface
test_df %>%
  summarize_across.(c(a, b), ~ mean(.x, na.rm = TRUE))

# Passing a list of functions (with .by)
test_df %>%
  summarize_across.(c(a, b), list(mean, max), .by = z)

# Passing a named list of functions (with .by)
test_df %>%
  summarize_across.(c(a, b),
                    list(avg = mean,
                         max_plus_one = ~ max(.x) + 1),
                    .by = z)
```

---

tidytable *Build a data.table/tidytable*

Description

`tidytable()` constructs a data.table, but one with nice printing features. As such it can be used exactly like a data.table would be used.

Usage

```
tidytable(...)
```

Arguments

`...`	Arguments passed to `data.table()`

Examples

```
tidytable(x = c(1,2,3), y = c(4,5,6))
```

top_n. *Select top (or bottom) n rows (by value)*

Description

Select the top or bottom entries in each group, ordered by `wt`.

Usage

```
top_n.(.df, n = 5, wt = NULL, .by = NULL, by = NULL)

dt_top_n(.df, n = 5, wt = NULL, .by = NULL, by = NULL)
```

Arguments

`.df`	A data.frame or data.table
`n`	Number of rows to return
`wt`	Optional. The variable to use for ordering. If NULL uses the last column in the data.table.
`.by`	Columns to group by
`by`	This argument has been renamed to .by and is deprecated

Examples

```
test_df <- data.table(
  x = 1:5,
  y = 6:10,
  z = c(rep("a", 3), rep("b", 2)))

test_df %>%
  top_n.(2, wt = y)

test_df %>%
  top_n.(2, wt = y, .by = z)
```

transmute. *Add new variables and drop all others*

Description

Unlike `mutate.()`, `transmute.()` keeps only the variables that you create

Usage

```
transmute.(.df, ..., .by = NULL, by = NULL)

dt_transmute(.df, ..., .by = NULL, by = NULL)
```

Arguments

.df	A data.frame or data.table
...	Columns to create/modify
.by	Columns to group by
by	This argument has been renamed to .by and is deprecated

Examples

```
mtcars %>%
  transmute.(displ_l = disp / 61.0237)
```

---

uncount. *Uncount a data.table*

Description

Uncount a data.table

Usage

```
uncount.(.df, weights, .remove = TRUE, .id = NULL)
```

Arguments

.df	A data.frame or data.table
weights	A column containing the weights to uncount by
.remove	If TRUE removes the selected weights column
.id	A string name for a new column containing a unique identifier for the newly uncounted rows.

Examples

```
df <- data.table(x = c("a", "b"), n = c(1, 2))

uncount.(df, n)

uncount.(df, n, .id = "id")
```

`unite.` *Unite multiple columns by pasting strings together*

Description

Convenience function to paste together multiple columns into one.

Usage

```
unite.(.df, col = "new_col", ..., sep = "_", remove = TRUE, na.rm = FALSE)

dt_unite(.df, col = "new_col", ..., sep = "_", remove = TRUE, na.rm = FALSE)
```

Arguments

`.df`	A data.frame or data.table
`col`	Name of the new column, as a string.
`...`	Selection of columns. If empty all variables are selected. `tidyselect` compatible.
`sep`	Separator to use between values
`remove`	If TRUE, removes input columns from the data.table.
`na.rm`	If TRUE, NA values will be not be part of the concatenation

Examples

```
test_df <- tidytable(a = c("a", "a", "a"),
                     b = c("b", "b", "b"),
                     c = c("c", "c", NA))

test_df %>%
  unite.("new_col", b, c)

test_df %>%
  unite.("new_col", where(is.character))

test_df %>%
  unite.("new_col", b, c, remove = FALSE)

test_df %>%
  unite.("new_col", b, c, na.rm = TRUE)

test_df %>%
  unite.()
```

unnest. *Unnest a nested data.table*

Description

Unnest a nested data.table.

Usage

```
unnest.(.df, ..., .keep_all = FALSE)

dt_unnest_legacy(.df, ..., .keep_all = FALSE)
```

Arguments

.df	A nested data.table
...	Columns to unnest. If empty, unnests all list columns. `tidyselect` compatible.
.keep_all	Should list columns that were not unnested be kept

Examples

```
nested_df <- data.table(
  a = 1:10,
  b = 11:20,
  c = c(rep("a", 6), rep("b", 4)),
  d = c(rep("a", 4), rep("b", 6))) %>%
  nest_by.(c, d) %>%
  mutate.(pulled_vec = map.(data, ~ pull.(.x, a)))

nested_df %>%
  unnest.(data)

nested_df %>%
  unnest.(data, pulled_vec)
```

where *Select variables with a function*

Description

This selection helper selects the variables for which a function returns ‘TRUE‘.

Usage

```
where(fn)
```

Arguments

fn	A function that returns 'TRUE' or 'FALSE' (technically, a _predicate_ function). Can also be a purrr-like formula.

Examples

```
iris %>% select.(where(is.factor))

iris %>% select.(where(is.numeric))

## The formula shorthand
# These expressions are equivalent:

iris %>% select.(where(is.numeric))

iris %>% select.(where(function(x) is.numeric(x)))

iris %>% select.(where(~ is.numeric(.x)))

# This shortahand is useful for adding logic inline.
# Here we select all numeric variables whose mean is greater than 3.5:
iris %>%
  select.(where(~ is.numeric(.x) && mean(.x) > 3.5))
```

%notin% *notin operator*

Description

notin operator

Usage

```
x %notin% y
```

Arguments

x	vector or NULL
y	vector or NULL

Examples

```
c(1,3,11) %notin% 1:10
```

Index