Package 'tidyfast'

October 14, 2022

Title Fast Tidying of Data

Version 0.2.1

Description Tidying functions built on 'data.table'
to provide quick and efficient data manipulation with
minimal overhead.

Imports data.table (>= 1.12.4), Rcpp

Suggests remotes, magrittr, tidyr, dplyr, testthat (>= 2.1.0), covr

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.1.0

LinkingTo Rcpp

NeedsCompilation yes

Author Tyson Barrett [aut, cre] (<https://orcid.org/0000-0002-2137-1391>),
Mark Fairbanks [ctb]

Maintainer Tyson Barrett <t.barrett@aggiemail.usu.edu>

Repository CRAN

Date/Publication 2020-03-20 10:40:02 UTC

R topics documented:

`dt_case_when` *Case When with data.table*

Description

Does what `dplyr::case_when()` does, with the same syntax, but with `data.table::fifelse()` under the hood

Usage

```
dt_case_when(...)
```

Arguments

`...`	statements of the form: `condition ~ label`, where the label is applied if the condition is met

Value

Vector of the same size as the input vector

Examples

```
x <- rnorm(100)
dt_case_when(
  x < median(x) ~ "low",
  x >= median(x) ~ "high",
  is.na(x) ~ "other"
  )

library(data.table)
temp <- data.table(pseudo_id = c(1, 2, 3, 4, 5),
                   x = sample(1:5, 5, replace = TRUE))
temp[, y := dt_case_when(pseudo_id == 1 ~ x * 1,
                         pseudo_id == 2 ~ x * 2,
                         pseudo_id == 3 ~ x * 3,
                         pseudo_id == 4 ~ x * 4,
                         pseudo_id == 5 ~ x * 5)]
```

dt_count *Count*

Description

Count the numbers of observations within groups

Usage

```
dt_count(dt_, ..., na.rm = FALSE, wt = NULL)
```

Arguments

dt_	the data table to uncount
...	groups
na.rm	should any rows with missingness be removed before the count? Default is FALSE.
wt	the wt assigned to the counts (same number of rows as the data)

Value

A data.table with counts for each group (or combination of groups)

Examples

```
library(data.table)
dt <- data.table(
  x = rnorm(1e5),
  y = runif(1e5),
  grp = sample(1L:3L, 1e5, replace = TRUE),
  wt = runif(1e5, 1, 100)
  )

dt_count(dt, grp)
dt_count(dt, grp, na.rm = TRUE)
dt_count(dt, grp, na.rm = TRUE, wt = wt)
```

dt_fill *Fill with data.table*

Description

Fills in values, similar to `tidyr::fill()`, by within `data.table`. This function relies on the `Rcpp` functions that drive `tidyr::fill()` but applies them within `data.table`.

Usage

```
dt_fill(dt_, ..., id = NULL, .direction = c("down", "up", "downup", "updown"))
```

Arguments

`dt_`	the data table (or if not a data.table then it is coerced with as.data.table)
`...`	the columns to fill
`id`	the grouping variable(s) to fill within
`.direction`	either "down" or "up" (down fills values down, up fills values up), or "downup" (down first then up) or "updown" (up first then down)

Value

A data.table with listed columns having values filled in

Examples

```
set.seed(84322)
library(data.table)

x = 1:10
dt = data.table(v1 = x,
                v2 = shift(x),
                v3 = shift(x, -1L),
                v4 = sample(c(rep(NA, 10), x), 10),
                grp = sample(1:3, 10, replace = TRUE))
dt_fill(dt, v2, v3, v4, id = grp, .direction = "downup")
dt_fill(dt, v2, v3, v4, id = grp)
dt_fill(dt, .direction = "up")
```

dt_hoist *Hoist: Fast Unnesting of Vectors*

Description

Quickly unnest vectors nested in list columns. Still experimental (has some potentially unexpected behavior in some situations)!

Usage

```
dt_hoist(dt_, ...)
```

Arguments

dt_	the data table to unnest
...	the columns to unnest (must all be the sample length when unnested); use bare names of the variables

Examples

```
library(data.table)
dt <- data.table(
   x = rnorm(1e5),
   y = runif(1e5),
   nested1 = lapply(1:10, sample, 10, replace = TRUE),
   nested2 = lapply(c("thing1", "thing2"), sample, 10, replace = TRUE),
   id = 1:1e5
   )

dt_hoist(dt, nested1, nested2)
```

dt_nest *Fast Nesting*

Description

Quickly nest data tables (similar to `dplyr::group_nest()`).

Usage

```
dt_nest(dt_, ..., .key = "data")
```

Arguments

dt_	the data table to nest
...	the variables to group by
.key	the name of the list column; default is "data"

Value

A data.table with a list column containing data.tables

Examples

```
library(data.table)
dt <- data.table(
  x = rnorm(1e5),
  y = runif(1e5),
  grp = sample(1L:3L, 1e5, replace = TRUE)
  )

dt_nest(dt, grp)
```

---

dt_pivot_longer *Pivot data from wide to long*

Description

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

Usage

```
dt_pivot_longer(
  dt_,
  cols = NULL,
  names_to = "name",
  values_to = "value",
  values_drop_na = FALSE,
  ...
)
```

Arguments

dt_	The data table to pivot longer
cols	Column selection. If empty, uses all columns. Can use -colname to unselect column(s)
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.data.table()‘

Value

A reshaped data.table into longer format

Examples

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

dt_pivot_longer(example_dt,
                cols = c(x, y),
                names_to = "stuff",
                values_to = "things")

dt_pivot_longer(example_dt,
                cols = -z,
                names_to = "stuff",
                values_to = "things")
```

---

dt_pivot_wider *Pivot data from long to wide*

Description

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

Usage

```
dt_pivot_wider(dt_, id_cols = NULL, names_from, names_sep = "_", values_from)
```

Arguments

dt_	the data table to widen
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.
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`).

names_sep	the separator between the names of the columns
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).

Value

A reshaped data.table into wider format

Examples

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

dt_pivot_wider(example_dt, names_from = stuff, values_from = things)
dt_pivot_wider(example_dt, names_from = stuff, values_from = things, id_cols = z)
```

---

dt_print_options *Set Print Method*

Description

The function allows the user to define options relating to the print method for data.table.

Usage

```
dt_print_options(
  class = TRUE,
  topn = 5,
  rownames = TRUE,
  nrows = 100,
  trunc.cols = TRUE
)
```

Arguments

class	should the variable class be printed? (options("datatable.print.class"))
topn	the number of rows to print (both head and tail) if nrows(DT) > nrows. (options("datatable.print.to
rownames	should rownames be printed? (options("datatable.print.rownames"))
nrows	total number of rows to print (options("datatable.print.nrows"))
trunc.cols	if TRUE, only the columns that fit in the console are printed (with a message stating the variables not shown, similar to tibbles; options("datatable.print.trunc.cols")). This only works on data.table versions higher than 1.12.6 (i.e. not currently available but anticipating the eventual release).

Value

None. This function is used for its side effect of changing options.

Examples

```
dt_print_options(
  class = TRUE,
  topn = 5,
  rownames = TRUE,
  nrows = 100,
  trunc.cols = TRUE)
```

`dt_separate` *Separate columns with data.table*

Description

Separates a column of data into others, by splitting based a separator or regular expression

Usage

```
dt_separate(
  dt_,
  col,
  into,
  sep = ".",
  remove = TRUE,
  fill = NA,
  fixed = TRUE,
  immutable = TRUE,
  ...
)
```

Arguments

`dt_`	the data table (or if not a data.table then it is coerced with as.data.table)
`col`	the column to separate
`into`	the names of the new columns created from splitting `col`.
`sep`	the regular expression stating how `col` should be separated. Default is `..`
`remove`	should `col` be removed in the returned data table? Default is `TRUE`
`fill`	if empty, fill is inserted. Default is `NA`.
`fixed`	logical. If TRUE match split exactly, otherwise use regular expressions. Has priority over perl.
`immutable`	If `TRUE`, `.dt` is treated as immutable (it will not be modified in place). Alternatively, you can set `immutable = FALSE` to modify the input object.
`...`	arguments passed to `data.table::tstrplit()`

Value

A data.table with a column split into multiple columns.

Examples

```
library(data.table)
d <- data.table(x = c("A.B", "A", "B", "B.A"),
                y = 1:4)

# defaults
dt_separate(d, x, c("c1", "c2"))

# can keep the original column with `remove = FALSE`
dt_separate(d, x, c("c1", "c2"), remove = FALSE)

# need to assign when `immutable = TRUE`
separated <- dt_separate(d, x, c("c1", "c2"), immutable = TRUE)
separated

# don't need to assign when `immutable = FALSE` (default)
dt_separate(d, x, c("c1", "c2"), immutable = FALSE)
d
```

---

dt_starts_with *Select helpers*

Description

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

- `dt_starts_with()`: Starts with a prefix
- `dt_starts_with()`: Ends with a suffix
- `dt_contains()`: Contains a literal string
- `dt_everything()`: Matches all variables

Usage

```
dt_starts_with(match)

dt_contains(match)

dt_ends_with(match)

dt_everything()
```

Arguments

match	a character string to match to variable names

Value

None. To be used within the dt_pivot_* functions.

Examples

```
library(data.table)

# example of using it with `dt_pivot_longer()`
df <- data.table(row = 1, var = c("x", "y"), a = 1:2, b = 3:4)
pv <- dt_pivot_wider(df,
                     names_from = var,
                     values_from = c(dt_starts_with("a"), dt_ends_with("b")))
```

---

dt_uncount *Uncount*

Description

Uncount a counted data table

Usage

```
dt_uncount(dt_, weights, .remove = TRUE, .id = NULL)
```

Arguments

dt_	the data table to uncount
weights	the counts for each
.remove	should the weights variable be removed?
.id	an optional new id variable, providing a unique id for each row

Value

A data.table with a row for each uncounted column.

Examples

```
library(data.table)

dt_count <- data.table(
  x = LETTERS[1:3],
  w = c(2,1,4)
)
uncount <- dt_uncount(dt_count, w, .id = "id")
uncount[]      # note that `[]` forces the printing
```

---

dt_unnest *Unnest: Fast Unnesting of Data Tables*

Description

Quickly unnest data tables, particularly those nested by `dt_nest()`.

Usage

```
dt_unnest(dt_, col, ...)
```

Arguments

`dt_`	the data table to unnest
`col`	the column to unnest
`...`	any of the other variables in the nested table that you want to keep in the unnested table. Bare variable names. If none are provided, all variables are kept.

Examples

```
library(data.table)
dt <- data.table(
  x = rnorm(1e5),
  y = runif(1e5),
  grp = sample(1L:3L, 1e5, replace = TRUE)
  )

nested <- dt_nest(dt, grp)
dt_unnest(nested, col = data)
```

Index