Package ‘tidyfst’

May 28, 2020

Title Tidy Verbs for Fast Data Manipulation
Version 0.9.7
Depends R (>= 3.3.0)
Description A toolkit of tidy data manipulation verbs with 'data.table' as the backend. Combining the merits of syntax elegance from 'dplyr' and computing performance from 'data.table', 'tidyfst' intends to provide users with state-of-the-art data manipulation tools with least pain. This package is an extension of 'data.table'. While enjoying a tidy syntax, it also wraps combinations of efficient functions to facilitate frequently-used data operations.

BugReports https://github.com/hope-data-science/tidyfst/issues
License MIT + file LICENSE
Encoding UTF-8
LazyData true
RoxygenNote 7.1.0
Imports data.table (>= 1.12.8), fst (>= 0.9.0), stringr (>= 1.4.0)
Suggests knitr, rmarkdown, nycflights13, pryr, tidyr, ggplot2
VignetteBuilder knitr
NeedsCompilation no
Author Tian-Yuan Huang [aut, cre] (<https://orcid.org/0000-0002-3591-4203>)
Maintainer Tian-Yuan Huang <huang.tian-yuan@qq.com>
Repository CRAN
Date/Publication 2020-05-28 08:10:03 UTC

R topics documented:

arrange_dt ................................................................. 3
as_fst ................................................................. 3
R topics documented:

- complete_dt ................................................. 4
- count_dt .................................................. 5
- cummean .................................................... 6
- distinct_dt ................................................. 6
- drop_na_dt ................................................ 7
- dummy_dt .................................................. 9
- export_fst ................................................ 10
- filter_dt ................................................... 11
- fst .......................................................... 12
- group_by_dt .............................................. 13
- group_dt ................................................. 15
- impute_dt ................................................ 16
- intersect_dt ............................................... 17
- in_dt ....................................................... 18
- join ......................................................... 19
- lead_dt .................................................... 21
- longer_dt .................................................. 22
- mat_df ...................................................... 23
- mutate_dt ................................................ 24
- mutate_when .............................................. 25
- nest_dt ..................................................... 26
- nth ........................................................ 28
- object_size ............................................... 29
- pull_dt ..................................................... 29
- relocate_dt .............................................. 30
- rename_dt ............................................... 31
- replace_dt ............................................... 32
- rn_col ...................................................... 33
- sample_dt ............................................... 33
- select_dt ................................................. 34
- separate_dt ............................................. 36
- slice_dt ................................................... 37
- summarise_dt .......................................... 38
- sys_time_print ......................................... 39
- top_dt ..................................................... 40
- t_dt ........................................................ 41
- uncount_dt ............................................... 42
- unite_dt .................................................. 42
- utf8_encoding .......................................... 43
- wider_dt .................................................. 44

Index 46
**arrange_dt**  

**Arrange entries in data.frame**

**Description**

Analogous function for `arrange` in `dplyr`.

**Usage**

```
arrange_dt(.data, ...)  
```

**Arguments**

- `.data`: data.frame
- `...`: Arrange by what group? Minus symbol means arrange by descending order.

**Value**

data.table

**See Also**

`arrange`

**Examples**

```
iris %>% arrange_dt(Sepal.Length)

# minus for decreasing order
iris %>% arrange_dt(-Sepal.Length)

# arrange by multiple variables
iris %>% arrange_dt(Sepal.Length,Petal.Length)
```

---

**as_fst**  

**Save a data.frame as a fst table**

**Description**

This function first export the data.frame to a temporal file, and then parse it back as a fst table (class name is "fst_table").

**Usage**

```
as_fst(.data)
```
Arguments

.data A data.frame

Value

An object of class fst_table

Examples

```r
## Not run:
iris %>%
as_fst() -> iris_fst
iris_fst

## End(Not run)
```

complete_dt

**Complete a data frame with missing combinations of data**

Description

Turns implicit missing values into explicit missing values. Analogous function for complete function in tidyr.

Usage

`complete_dt(.data, ..., fill = NA)`

Arguments

- `.data` data.frame
- `...` Specification of columns to expand. The selection of columns is supported by the flexible `select_dt`. To find all unique combinations of provided columns, including those not found in the data, supply each variable as a separate argument. But the two modes (select the needed columns and fill outside values) could not be mixed, find more details in examples.
- `fill` Atomic value to fill into the missing cell, default uses NA.

Details

When the provided columns with addition data are of different length, all the unique combinations would be returned. This operation should be used only on unique entries, and it will always returned the unique entries.

If you supply fill parameter, these values will also replace existing explicit missing values in the data set.
**count_dt**

**Value**

data.table

**See Also**

cOMPlete

**Examples**

df <- data.table(
  group = c(1:2, 1),
  item_id = c(1:2, 2),
  item_name = c("a", "b", "b"),
  value1 = 1:3,
  value2 = 4:6
)

  df %>% complete_dt(item_id, item_name)
  df %>% complete_dt(item_id, item_name, fill = 0)
  df %>% complete_dt("item")
  df %>% complete_dt(item_id = 1:3)
  df %>% complete_dt(item_id = 1:3, group = 1:2)
  df %>% complete_dt(item_id = 1:3, group = 1:3, item_name = c("a", "b", "c"))

---

**count_dt**

*Count observations by group*

**Description**

Analogous function for `count` and `add_count` in `dplyr`.

**Usage**

```r
count_dt(.data, ..., sort = TRUE, .name = "n")
add_count_dt(.data, ..., .name = "n")
```

**Arguments**

- `.data` data.table/data.frame data.frame will be automatically converted to data.table.
- `...` variables to group by.
- `sort` logical. If TRUE result will be sorted in descending order by resulting variable.
- `.name` character. Name of resulting variable. Default uses "n".

**Value**

data.table
See Also
count

Examples

```r
iris %>% count_dt(Species)
iris %>% count_dt(Species,.name = "count")
iris %>% add_count_dt(Species)
iris %>% add_count_dt(Species,.name = "N")

mtcars %>% count_dt(cyl,vs)
mtcars %>% count_dt(cyl,vs,.name = "N",sort = FALSE)
mtcars %>% add_count_dt(cyl,vs)
```

cummean | *Cumulative mean*

**Description**

Returns a vector whose elements are the cumulative mean of the elements of the argument.

**Usage**

cummean(x)

**Arguments**

x | a numeric or complex object, or an object that can be coerced to one of these.

**Examples**

cummean(1:10)

distinct_dt | *Select distinct/unique rows in data.frame*

**Description**

Analogous function for distinct in `dplyr`

**Usage**

distinct_dt(.data, ..., .keep_all = FALSE)
drop_na_dt

Arguments

.data data.frame

... Optional variables to use when determining uniqueness. If there are multiple rows for a given combination of inputs, only the first row will be preserved. If omitted, will use all variables.

.keep_all If TRUE, keep all variables in data.frame. If a combination of ... is not distinct, this keeps the first row of values.

Value
data.table

See Also

distinct

Examples

iris %>% distinct_dt()
iris %>% distinct_dt(Species)
iris %>% distinct_dt(Species,.keep_all = TRUE)
mtcars %>% distinct_dt(cyl,vs)
mtcars %>% distinct_dt(cyl,vs,.keep_all = TRUE)

drop_na_dt Dump, replace and fill missing values in data.frame

Description

Analogous function for drop_na, replace_na and fill in tidyr, but with a different API.

Usage

drop_na_dt(.data, ...)
replace_na_dt(.data, ..., to)
delete_na_cols(.data, prop = NULL, n = NULL)
delete_na_rows(.data, prop = NULL, n = NULL)
fill_na_dt(.data, ..., direction = "down")
shift_fill(x, direction = "down")
Arguments

- data: data.frame
- ...: Columns to be replaced or filled. If not specified, use all columns.
- to: What value should NA replace by?
- prop: If proportion of NAs is larger than or equal to "prop", would be deleted.
- n: If number of NAs is larger than or equal to "n", would be deleted.
- direction: Direction in which to fill missing values. Currently either "down" (the default) or "up".
- x: A vector with missing values to be filled.

Details

drop_na_dt drops the entries with NAs in specific columns. fill_na_dt fill NAs with observations ahead ("down") or below ("up"), which is also known as last observation carried forward (LOCF) and next observation carried backward (NOCB).
delete_na_cols could drop the columns with NA proportion larger than or equal to "prop" or NA number larger than or equal to "n". delete_na_rows works alike but deals with rows.
shift_fill could fill a vector with missing values.

Value

data.table

References

https://stackoverflow.com/questions/2643939/remove-columns-from-dataframe-where-all-values-are-na

See Also

drop_na, replace_na, fill

Examples

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

df %>% replace_na_dt(to = 0)
df %>% replace_na_dt(x,to = 0)
df %>% replace_na_dt(y,to = 0)
df %>% replace_na_dt(x,y,to = 0)
```
df %>% fill_na_dt(x)
df %>% fill_na_dt()  # not specified, fill all columns
df %>% fill_na_dt(y, direction = "up")

x = data.frame(x = c(1, 2, NA, 3), y = c(NA, NA, 4, 5), z = rep(NA, 4))
x
x %>% delete_na_cols()
x %>% delete_na_cols(prop = 0.75)
x %>% delete_na_cols(prop = 0.5)
x %>% delete_na_cols(prop = 0.24)
x %>% delete_na_cols(n = 2)

x %>% delete_na_rows(prop = 0.6)
x %>% delete_na_rows(n = 2)

# shift_fill
y = c("a",NA,"b",NA,"c")
shift_fill(y)  # equals to
shift_fill(y,"down")
shift_fill(y,"up")

dummy_dt  

Fast creation of dummy variables

Description

Quickly create dummy (binary) columns from character and factor type columns in the inputted data (and numeric columns if specified.) This function is useful for statistical analysis when you want binary columns rather than character columns.

Usage

dummy_dt(.data, ..., longname = TRUE)

Arguments

.data  
data.frame

...  
Columns you want to create dummy variables from. Very flexible, find in the examples.

longname  
logical. Should the output column labeled with the original column name? Default uses TRUE.

Details

If no columns provided, will return the original data frame.

This function is inspired by fastDummies package, but provides simple and precise usage, whereas fastDummies::dummy_cols provides more features for statistical usage.
export_fst

Value
data.table

See Also
dummy_cols

Examples

iris %>% dummy_dt(Species)
iris %>% dummy_dt(Species,longname = FALSE)

mtcars %>% head() %>% dummy_dt(vs,am)
mtcars %>% head() %>% dummy_dt("cyl|gear")

Description

Wrapper for read_fst and write_fst from fst, but use a different default. For data import, always return a data.table. For data export, always compress the data to the smallest size.

Usage

export_fst(x, path, compress = 100, uniform_encoding = TRUE)

import_fst(
  path,
  columns = NULL,
  from = 1,
  to = NULL,
  as.data.table = TRUE,
  old_format = FALSE
)

Arguments

x a data frame to write to disk
path path to fst file
compress value in the range 0 to 100, indicating the amount of compression to use. Lower values mean larger file sizes. The default compression is set to 50.
uniform_encoding

If ‘TRUE’, all character vectors will be assumed to have elements with equal encoding. The encoding (latin1, UTF8 or native) of the first non-NA element will be used as encoding for the whole column. This will be a correct assumption for most use cases. If ‘uniform.encoding’ is set to ‘FALSE’, no such assumption
filter_dt

will be made and all elements will be converted to the same encoding. The latter is a relatively expensive operation and will reduce write performance for character columns.

columns  Column names to read. The default is to read all columns.
from    Read data starting from this row number.
to      Read data up until this row number. The default is to read to the last row of the stored dataset.
as.data.table  If TRUE, the result will be returned as a data.table object. Any keys set on dataset x before writing will be retained. This allows for storage of sorted datasets. This option requires data.table package to be installed.
old_format  must be FALSE, the old fst file format is deprecated and can only be read and converted with fst package versions 0.8.0 to 0.8.10.

Value

‘import_fst’ returns a data.table with the selected columns and rows. ‘export_fst’ writes ‘x’ to a ‘fst’ file and invisibly returns ‘x’ (so you can use this function in a pipeline).

See Also

read_fst

Examples

## Not run:
export_fst(iris,"iris_fst_test.fst")
iris_dt = import_fst("iris_fst_test.fst")
iris_dt
unlink("iris_fst_test.fst")

## End(Not run)

---

**filter_dt**  
*Filter entries in data.frame*

**Description**

Analogous function for filter in *dplyr*.

**Usage**

```
filter_dt(.data, ...)
```

**Arguments**

- `.data`  data.frame
- `...`  List of variables or name-value pairs of summary/modifications functions.
Value
data.table

See Also
filter

Examples
iris %>% filter_dt(Sepal.Length > 7)
iris %>% filter_dt(Sepal.Length > 7, Sepal.Width > 3)
iris %>% filter_dt(Sepal.Length > 7 & Sepal.Width > 3)
iris %>% filter_dt(Sepal.Length == max(Sepal.Length))

fst
Parse, inspect and extract data.table from fst file

Description
An API for reading fst file as data.table.

Usage
parse_fst(path)
slice_fst(ft, row_no)
select_fst(ft, ...)
filter_fst(ft, ...)
summary_fst(ft)

Arguments
path path to fst file
ft An object of class fst_table, returned by parse_fst
row_no An integer vector (Positive)
... The filter conditions

Details
summary_fst could provide some basic information about the fst table.

Value
parse_fst returns a fst_table class.
select_fst and filter_fst returns a data.table.
group_by_dt

See Also

fst, metadata_fst

Examples

```r
## Not run:
fst::write_fst(iris,"iris_test.fst")
# parse the file but not reading it
parse_fst("iris_test.fst") -> ft
ft

class(ft)
lapply(ft,class)
names(ft)
dim(ft)
summary_fst(ft)

# get the data by query
ft %>% slice_fst(1:3)
ft %>% slice_fst(c(1,3))

ft %>% select_fst(Sepal.Length)
ft %>% select_fst(Sepal.Length,Sepal.Width)
ft %>% select_fst("Sepal.Length")
ft %>% select_fst(1:3)
ft %>% select_fst(1,3)
ft %>% select_fst("Se")
ft %>% select_fst("nothing")
ft %>% select_fst("Se\Sp")
ft %>% select_fst(cols = names(iris)[2:3])

ft %>% filter_fst(Sepal.Width > 3)
ft %>% filter_fst(Sepal.Length > 6 , Species == "virginica")
ft %>% filter_fst(Sepal.Length > 6 & Species == "virginica" & Sepal.Width < 3)

unlink("iris_test.fst")

## End(Not run)
```

---

**group_by_dt**

*Group by variable(s) and implement operations*

**Description**

Using `setkey` and `setkeyv` in `data.table` to carry out `group_by`-like functionalities in `dplyr`. This is not only convenient but also efficient in computation.
Usage

```r
group_by_dt(.data, ..., cols = NULL)
group_exe_dt(.data, ...)
```

Arguments

- `.data` A data frame
- `...` Variables to group by for `group_by_dt`, namely the columns to sort by. Do not quote the column names. Any data manipulation arguments that could be implemented on a data.frame for `group_exe_dt`. It can receive what `select_dt` receives.
- `cols` A character vector of column names to group by.

Details

`group_by_dt` and `group_exe_dt` are a pair of functions to be used in combination. It utilizes the feature of key setting in data.table, which provides high performance for group operations, especially when you have to operate by specific groups frequently.

Value

A data.table with keys

Examples

```r
# aggregation after grouping using group_exe_dt
as.data.table(iris) -> a
a %>%
  group_by_dt(Species) %>%
  group_exe_dt(head(1))

a %>%
  group_by_dt(Species) %>%
  group_exe_dt(
    head(3) %>%
    summarise_dt(sum = sum(Sepal.Length))
  )

mtcars %>%
  group_by_dt("cyl|am") %>%
  group_exe_dt(
    summarise_dt(mpg_sum = sum(mpg))
  )
# equals to
mtcars %>%
  group_by_dt(cols = c("cyl","am")) %>%
  group_exe_dt(
    summarise_dt(mpg_sum = sum(mpg))
  )
```
group_dt

Data manipulation within groups

Description
Analogous function for group_by and rowwise in dplyr, but in another efficient way.

Usage
group_dt(.data, by = NULL, ...)

rowwise_dt(.data, ...)

Arguments
.data A data.frame
by Variables to group by, unquoted name of grouping variable of list of unquoted names of grouping variables.
... Any data manipulation arguments that could be implemented on a data.frame.

Details
If you want to use summarise_dt and mutate_dt in group_dt, it is better to use the "by" parameter in those functions, that would be much faster because you don’t have to use .SD (which takes extra time to copy).

Value
data.table

References
https://stackoverflow.com/questions/36802385/use-by-each-row-for-data-table

Examples
iris %>% group_dt(by = Species, slice_dt(1:2))
iris %>% group_dt(Species, filter_dt(Sepal.Length == max(Sepal.Length)))
iris %>% group_dt(Species, summarise_dt(new = max(Sepal.Length)))

# you can pipe in the `group_dt`
iris %>% group_dt(Species,
    mutate_dt(max = max(Sepal.Length)) %>%
    summarise_dt(sum = sum(Sepal.Length)))

# for users familiar with data.table, you can work on .SD directly
# following codes get the first and last row from each group
iris %>%
```r
impute_dt(
  .data, ...,
  .func = "mode"
)
```

### Arguments

- **.data**: A data.frame
- **...**: Columns to select
- **.func**: Character, "mode" (default), "mean" or "median". Could also define it by oneself.

---

**Description**

Impute missing values with mean, median or mode.

**Usage**

`impute_dt(.data, ..., .func = "mode")`

---

**Examples**

```r
# for summarise_dt, you can use "by" to calculate within the group
mtcars %>%
  summarise_dt(
    disp = mean(disp),
    hp = mean(hp),
    by = cyl
  )

# but you could also, of course, use group_dt
mtcars %>%
  group_dt(by = .(vs, am),
    summarise_dt(avg = mean(mpg)))

# and list of variables could also be used
mtcars %>%
  group_dt(by = list(vs, am),
    summarise_dt(avg = mean(mpg)))

# examples for 'rowwise_dt'
df <- data.table(x = 1:2, y = 3:4, z = 4:5)

df %>%
  rowwise_dt(
    mutate_dt(m = mean(c(x, y, z)))
  )
```
**intersect_dt**

Value

A data.table

Examples

```r
Pclass <- c(3, 1, 3, 1, 3, 2, 2, 3, NA, NA)
Sex <- c('male', 'male', 'female', 'female', 'female',
         'female', NA, 'male', 'female', NA)
Age <- c(22, 38, 26, 35, NA,
        45, 25, 39, 28, 40)
SibSp <- c(0, 1, 3, 1, 2, 3, 2, 2, NA, 0)
Fare <- c(7.25, 71.3, 7.92, NA, 8.05, 8.46, 51.9, 60, 32, 15)
Embarked <- c('S', NA, 'S', 'Q', 'Q', 'S', 'C', 'S', 'S', 'S')
data <- data.frame(
  'Pclass' = Pclass,
  'Sex' = Sex,
  'Age' = Age,
  'SibSp' = SibSp,
  'Fare' = Fare,
  'Embarked' = Embarked)

data %>% impute_dt() # default uses "mode" as \'.func\'
data %>% impute_dt(is.numeric,.func = "mean")
data %>% impute_dt(is.numeric,.func = "median")

# use user defined function
my_fun = function(x){
  (max(x,na.rm = TRUE) - min(x,na.rm = TRUE))/2
}
data %>% impute_dt(is.numeric,.func = my_fun)
```

---

**intersect_dt**

*Set operations for data frames*

Description

Wrappers of set operations in `data.table`. Only difference is it could be applied to non-data.table data frames by recognizing and coercing them to data.table automatically.

Usage

```r
intersect_dt(x, y, all = FALSE)
union_dt(x, y, all = FALSE)
setdiff_dt(x, y, all = FALSE)
setequal_dt(x, y, all = TRUE)
```
in_dt

Arguments

x A data.frame
y A data.frame
all Logical. When FALSE (default), removes duplicate rows on the result.

Value

A data.table

See Also

setops

Examples

x = iris[c(2,3,3,4),]
x2 = iris[2:4,]
y = iris[c(3:5),]

intersect_dt(x, y) # intersect
intersect_dt(x, y, all=TRUE) # intersect all
setdiff_dt(x, y) # except
setdiff_dt(x, y, all=TRUE) # except all
union_dt(x, y) # union
union_dt(x, y, all=TRUE) # union all
setequal_dt(x, x2, all=FALSE) # setequal
setequal_dt(x, x2) # setequal all

in_dt Short cut to data.table

Description

To use facilities provided by data.table, but do not have to load data.table package.

Usage

in_dt(.data, ...)

as_dt(.data)

Arguments

.data A data.frame
... Recieve B in data.table’s A[B] syntax.
Details

The `as_dt` could turn any data frame to data.table class. If the data is not a data frame, return error.

The `in_dt` function creates a virtual environment in data.table, it could be piped well because it still follows the principals of `tidyfst`, which are: (1) Never use in place replacement and (2) Always receives a data frame (data.frame/tibble/data.table) and returns a data.table. Therefore, the in place functions like `:=` will still return the results.

See Also

data.table

Examples

```r
iris %>% as_dt()
iris %>% in_dt(order(-Sepal.Length),.SD[,N],by=Species)
```

join

Join tables

Description

The mutating joins add columns from ‘y’ to ‘x’, matching rows based on the keys:

- ‘inner_join_dt()’: includes all rows in ‘x’ and ‘y’. * ‘left_join_dt()’: includes all rows in ‘x’. * ‘right_join_dt()’: includes all rows in ‘y’. * ‘full_join_dt()’: includes all rows in ‘x’ or ‘y’.

Filtering joins filter rows from ‘x’ based on the presence or absence of matches in ‘y’:

- ‘semi_join_dt()’ return all rows from ‘x’ with a match in ‘y’. * ‘anti_join_dt()’ return all rows from ‘x’ without a match in ‘y’.

Usage

```r
inner_join_dt(x, y, by = NULL, on = NULL, suffix = c(".x", ".y"))
left_join_dt(x, y, by = NULL, on = NULL, suffix = c(".x", ".y"))
right_join_dt(x, y, by = NULL, on = NULL, suffix = c(".x", ".y"))
full_join_dt(x, y, by = NULL, on = NULL, suffix = c(".x", ".y"))
anti_join_dt(x, y, by = NULL, on = NULL)
semi_join_dt(x, y, by = NULL, on = NULL)
```
Arguments

x  A data.table
y  A data.table
by (Optional) A character vector of variables to join by.

If ‘NULL’, the default, ‘*_join_dt()’ will perform a natural join, using all variables in common across ‘x’ and ‘y’. A message lists the variables so that you can check they’re correct; suppress the message by supplying ‘by’ explicitly.

To join by different variables on ‘x’ and ‘y’, use a named vector. For example, ‘by = c("a" = "b")’ will match ‘x$a’ to ‘y$b’.

To join by multiple variables, use a vector with length > 1. For example, ‘by = c("a", "b")’ will match ‘x$a’ to ‘y$a’ and ‘x$b’ to ‘y$b’. Use a named vector to match different variables in ‘x’ and ‘y’. For example, ‘by = c("a" = "b", "c" = "d")’ will match ‘x$a’ to ‘y$b’ and ‘x$c’ to ‘y$d’.

on (Optional) Indicate which columns in x should be joined with which columns in y. Examples included: 1. by = c("a", "b") (this is a must for set_full_join_dt); 2. by = c(x1="y1", x2="y2"); 3. by = c("x1==y1", "x2==y2"); 4. by = c("a", V2="b"); 5. by = .(a,b); 6. by = c("x<=a", "y<=b") or .by = .(x<=a,y<=b).

suffix If there are non-joined duplicate variables in x and y, these suffixes will be added to the output to disambiguate them. Should be a character vector of length 2.

Value

A data.table

Examples

workers = fread("name company
  Nick Acme
  John Ajax
  Daniela Ajax
")

positions = fread("name position
  John designer
  Daniela engineer
  Cathie manager
")

workers %>% inner_join_dt(positions)
workers %>% left_join_dt(positions)
workers %>% right_join_dt(positions)
workers %>% full_join_dt(positions)

# filtering joins
workers %>% anti_join_dt(positions)
workers %>% semi_join_dt(positions)
# To suppress the message, supply 'by' argument
workers %>% left_join_dt(positions, by = "name")

# Use a named 'by' if the join variables have different names
positions2 = setNames(positions, c("worker", "position")) # rename first column in 'positions'
workers %>% inner_join_dt(positions2, by = c("name" = "worker"))

# the syntax of 'on' could be a bit different
workers %>% inner_join_dt(positions2, on = "name==worker")

lead_dt

**Fast lead/lag for vectors**

**Description**
Analogous function for lead and lag in **dplyr** by wrapping **data.table**’s shift.

**Usage**
lead_dt(x, n = 1L, fill = NA)

lag_dt(x, n = 1L, fill = NA)

**Arguments**
- **x** A vector
- **n** a positive integer of length 1, giving the number of positions to lead or lag by. Default uses 1
- **fill** Value to use for padding when the window goes beyond the input length. Default uses NA

**Value**
A vector

**See Also**

lead, shift

**Examples**

lead_dt(1:5)
lag_dt(1:5)
lead_dt(1:5,2)
lead_dt(1:5,n = 2,fill = 0)
longer_dt

Pivot data from wide to long

Description

Analogous function for pivot_longer in tidy.

Usage

longer_dt(.data, ..., name = "name", value = "value", na.rm = FALSE)

Arguments

.data A data.frame
... Pattern for unchanged group or unquoted names. Pattern can accept regular
equation expression to match column names. It can recieve what select_dt recieves.
name Name for the measured variable names column. The default name is 'name'.
value Name for the molten data values column(s). The default name is 'value'.
na.rm If TRUE, NA values will be removed from the molten data.

Value

A data.table

See Also

wider_dt, melt, pivot_longer

Examples

## Example 1:
stocks = data.frame(
  time = as.Date('2009-01-01') + 0:9,
  X = rnorm(10, 0, 1),
  Y = rnorm(10, 0, 2),
  Z = rnorm(10, 0, 4)
)

stocks
stocks %>%
  longer_dt(time)
stocks %>%
  longer_dt("ti")

# Example 2:
library(tidyr)

billboard %>%
  longer_dt(
    -"wk",
    name = "week",
    value = "rank",
    na.rm = TRUE
  )

# or use:
billboard %>%
  longer_dt(
    artist,track,date.entered,
    name = "week",
    value = "rank",
    na.rm = TRUE
  )

# or use:
billboard %>%
  longer_dt(
    1:3,
    name = "week",
    value = "rank",
    na.rm = TRUE
  )

---

**mat_df**

*Conversion between tidy table and named matrix*

**Description**

Convenient functions to implement conversion between tidy table and named matrix.

**Usage**

`mat_df(m)`

`df_mat(df, row, col, value)`

**Arguments**

- `m`: A matrix
- `df`: A data.frame with at least 3 columns, one for row name, one for column name, and one for values. The names for column and row should be unique.
mutate_dt

mutate dt

row Unquoted expression of column name for row
col Unquoted expression of column name for column
value Unquoted expression of column name for values

Value
For mat_df, a data.frame. For df_mat, a named matrix.

Examples

```r
mm = matrix(c(1:8,NA),ncol = 3,dimnames = list(letters[1:3],LETTERS[1:3]))
mm
tdf = mat_df(mm)
tdf
mat = df_mat(tdf,row,col,value)
setequal(mm,mat)

tdf %>%
  setNames(c("A","B","C")) %>%
da_f_mat(A,B,C)
```

---

mutate_dt Mutate columns in data.frame

Description
Analogous function for mutate and transmute in dplyr.

Usage

```r
mutate_dt(.data, ..., by)
transmute_dt(.data, ..., by)
```

Arguments

- `.data` data.frame
- `...` List of variables or name-value pairs of summary/modifications functions.
- `by` (Optional) Mutate by what group?

Value
data.table

See Also

mutate
mutate_when

Examples

iris %>% mutate_dt(one = 1, Sepal.Length = Sepal.Length + 1)  
iris %>% transmute_dt(one = 1, Sepal.Length = Sepal.Length + 1)  
# add group number with symbol `.GRP`  
iris %>% mutate_dt(id = 1:N, grp = .GRP, by = Species)

---

mutate_when

*Conditional update of columns in data.table*

Description

`mutate_when` integrates `mutate` and `case_when` in `dplyr` and make a new tidy verb for `data.table`. `mutate_vars` is a super function to do updates in specific columns according to conditions.

Usage

`mutate_when(.data, when, ..., by)`

`mutate_vars(.data, .cols = NULL, .func, ..., by)`

Arguments

- `.data` : data.frame
- `when` : An object which can be coerced to logical mode
- `...` : Name-value pairs of expressions for `mutate_when`. Additional parameters to be passed to parameter `.func` in `mutate_vars`.
- `by` : (Optional) Mutate by what group?
- `.cols` : Any types that can be accepted by `select_dt`.
- `.func` : Function to be run within each column, should return a value or vectors with same length.

Value

data.table

See Also

`select_dt, case_when`
Examples

```r
iris[3:8,]
iris[3:8,] %>%
  mutate_when(Petal.Width == .2,
              one = 1, Sepal.Length = 2)
iris %>% mutate_vars("Pe", scale)
iris %>% mutate_vars(is.numeric, scale)
iris %>% mutate_vars(!is.factor, scale)
iris %>% mutate_vars(1:2, scale)
iris %>% mutate_vars(.func = as.character)
```

---

**nest_dt**

Nest and unnest

**Description**

Analogous function for `nest` and `unnest` in `tidyr`. `unnest_dt` will automatically remove other list-columns except for the target list-columns (which would be unnested later). Also, `squeeze_dt` is designed to merge multiple columns into list column.

**Usage**

```
nest_dt(.data, ..., mcols = NULL)
unnest_dt(.data, ...)
squeeze_dt(.data, ...)
chop_dt(.data, ...)
unchop_dt(.data, ...)
```

**Arguments**

- `.data` : data.table, nested or unnested
- `...` : The variables for nest group(for `nest_dt`), columns to be nested(for `squeeze_dt` and `chop_dt`), or column(s) to be unnested(for `unnest_dt`). Could recieve anything that `select_dt` could receive.
- `mcols` : Name-variable pairs in the list, form like `list(petal="^Pe", sepal="^Se")`, see example.

**Details**

In the `nest_dt`, the data would be nested to a column named `ndt`, which is short for nested data.table.

The `squeeze_dt` would not remove the originial columns.
The unchop_dt is the reverse operation of chop_dt.
These functions are experiencing the experimental stage, especially the unnest_dt. If they don’t work on some circumstances, try tidyR package.

Value
data.table, nested or unnested

References

See Also
nest, chop

Examples
# examples for nest_dt
# nest by which columns?
mtcars %>% nest_dt(cyl)
mtcars %>% nest_dt("cyl")
mtcars %>% nest_dt(cyl,vs)
mtcars %>% nest_dt(vs:am)
mtcars %>% nest_dt("cyl|vs")
mtcars %>% nest_dt(c("cyl","vs"))

# nest two columns directly
iris %>% nest_dt(mcols = list(petal="^Pe",sepal="^Se"))

# nest more flexibly
iris %>% nest_dt(mcols = list(ndt1 = 1:3,
    ndt2 = "Pe",
    ndt3 = Sepal.Length:Sepal.Width))

# examples for unnest_dt
# unnest which column?
mtcars %>% nest_dt("cyl|vs") %>%
    unnest_dt(ndt)
mtcars %>% nest_dt("cyl|vs") %>%
    unnest_dt("ndt")

df <- data.table(
a = list(c("a", "b"), "c"),
b = list(c(TRUE,TRUE),FALSE),
c = list(3,c(1,2)),
d = c(11, 22)
)
df
df %>% unnest_dt(a)
df %>% unnest_dt(2)
df %>% unnest_dt("c")
df %>% unnest_dt(cols = names(df)[3])

# You can unnest multiple columns simultaneously
df %>% unnest_dt(1:3)
df %>% unnest_dt(a,b,c)
df %>% unnest_dt("a|b|c")

# examples for squeeze_dt
# nest which columns?
iris %>% squeeze_dt(1:2)
iris %>% squeeze_dt("Se")
iris %>% squeeze_dt(Sepal.Length:Petal.Width)

# examples for chop_dt
df <- data.table(x = c(1, 1, 1, 2, 2, 3), y = 1:6, z = 6:1)
df %>% chop_dt(y,z)
df %>% chop_dt(y,z) %>% unchop_dt(y,z)

---

### nth

**Extract the nth value from a vector**

**Description**

Get the value from a vector with its position.

**Usage**

\[ \text{nth}(v, n = 1) \]

**Arguments**

- **v**: A vector
- **n**: A single integer specifying the position. Default uses 1. Negative integers index from the end (i.e. -1L will return the last value in the vector). If a double is supplied, it will be silently truncated.

**Value**

A single value.
Examples

```r
x = 1:10
nth(x, 1)
nth(x, 5)
nth(x, -2)
```

---

**object_size**  
*Nice printing of report the Space Allocated for an Object*

**Description**

Provides an estimate of the memory that is being used to store an R object. A wrapper of `object.size`, but use a nicer printing unit.

**Usage**

```r
object_size(object)
```

**Arguments**

- `object` an R object.

**Value**

An object of class "object_size"

**Examples**

```r
iris %>% object_size()
```

---

**pull_dt**  
*Pull out a single variable*

**Description**

Analogous function for `pull` in `dplyr`

**Usage**

```r
pull_dt(.data, col)
```
Arguments

.data data.frame

col A name of column or index (should be positive).

Value

vector

See Also

pull

Examples

mtcars %>% pull_dt(2)
mtcars %>% pull_dt(cyl)
mtcars %>% pull_dt("cyl")

Description

Use ‘relocate_dt()’ to change column positions, using the same syntax as ‘select_dt()’. Check similar function as ‘relocate’ in dplyr.

Usage

relocate_dt(.data, ..., how = "first", where = NULL)

Arguments

.data A data.frame

... Columns to move

how The mode of movement, including "first","last","after","before". Default uses "first".

where Destination of columns selected by .... Applicable for "after" and "before" mode.

Value

A data.table with rearranged columns.
rename_dt

Examples

df <- data.table(a = 1, b = 1, c = 1, d = "a", e = "a", f = "a")
df
  df %>% relocate_dt(f)
  df %>% relocate_dt(a, how = "last")
  df %>% relocate_dt(is.character)
  df %>% relocate_dt(is.numeric, how = "last")
  df %>% relocate_dt("[aeiou]")
  df %>% relocate_dt(a, how = "after", where = f)
  df %>% relocate_dt(f, how = "before", where = a)
  df %>% relocate_dt(f, how = "before", where = c)
  df %>% relocate_dt(f, how = "after", where = c)

df2 <- data.table(a = 1, b = "a", c = 1, d = "a")
df2 %>% relocate_dt(is.numeric,
                     how = "after",
                     where = is.character)
  df2 %>% relocate_dt(is.numeric,
                     how = "before",
                     where = is.character)

rename_dt

Rename column in data.frame

Description

Analogous function for rename in dplyr

Usage

rename_dt(.data, ...)

Arguments

.data  data.frame
...
staments of rename, e.g. ‘sl = Sepal.Length’ means the column named as "Sepal.Length"
would be renamed to "sl"

Value

data.table

See Also

rename
Examples

```r
iris %>%
  rename_dt(sl = Sepal.Length, sw = Sepal.Width) %>%
  head()
```

---

**replace_dt**

*Fast value replacement in data frame*

**Description**

While `replace_na_dt` could replace all NAs to another value, `replace_dt` could replace any value(s) to another specific value.

**Usage**

```r
replace_dt(.data, ..., from = is.nan, to = NA)
```

**Arguments**

- `.data` A data.frame
- `...` Columns to be replaced. If not specified, use all columns.
- `from` A value, a vector of values or a function returns a logical value. Defaults to `is.nan`.
- `to` A value. Defaults to NA.

**Value**

A data.table.

**See Also**

`replace_na_dt`

**Examples**

```r
iris %>% mutate_vars(is.factor, as.character) -> new_iris

new_iris %>%
  replace_dt(Species, from = "setosa", to = "SS")
new_iris %>%
  replace_dt(Species, from = c("setosa", "virginica"), to = "sv")
new_iris %>%
  replace_dt(Petal.Width, from = .2, to = 2)
new_iris %>%
  replace_dt(from = .2, to = NA)
new_iris %>%
  replace_dt(is.numeric, from = function(x) x > 3, to = 9999)
```
rn_col

Tools for working with row names

Description

The enhanced data.frame, including tibble and data.table, do not support row names. To link to some base r facilities, there should be functions to save information in row names. These functions are analogous to rownames_to_column and column_to_rownames in tibble.

Usage

rn_col(.data, var = "rowname")

col_rn(.data, var = "rowname")

Arguments

.data A data.frame.

.var Name of column to use for rownames.

Value

rn_col returns a data.table, col_rn returns a data frame.

Examples

mtcars %>% rn_col()

mtcars %>% rn_col("rn")

mtcars %>% rn_col() -> new_mtcars

new_mtcars %>% col_rn() -> old_mtcars

old_mtcars

setequal(mtcars,old_mtcars)

sample_dt

Sample n rows from a table

Description

Analogous function for sample_n and sample_frac in dplyr.

sample_dt is a merged version of sample_n_dt and sample_frac_dt, this could be convenient.
select_dt

Usage

```r
sample_dt(.data, n = NULL, prop = NULL, replace = FALSE)
sample_n_dt(.data, size, replace = FALSE)
sample_frac_dt(.data, size, replace = FALSE)
```

Arguments

- `.data` A data.frame
- `n` Number of rows to select
- `prop` Fraction of rows to select
- `replace` Sample with or without replacement? Default uses FALSE.
- `size` For sample_n_dt, the number of rows to select. For sample_frac_dt, the fraction of rows to select.

Value

data.table

See Also

`sample_n`, `sample_frac`

Examples

```r
sample_n_dt(mtcars, 10)
sample_n_dt(mtcars, 50, replace = TRUE)
sample_frac_dt(mtcars, 0.1)
sample_frac_dt(mtcars, 1.5, replace = TRUE)
sample_dt(mtcars, n=10)
sample_dt(mtcars, prop = 0.1)
```

---

select_dt

Select column from data.frame

Description

Analogous function for select and select_if in dplyr.

Usage

```r
select_dt(.data, ..., cols = NULL, negate = FALSE)
select_mix(.data, ..., rm.dup = TRUE)
```
**Arguments**

- `.data` (data.frame)
  List of variables or name-value pairs of summary/modifications functions. It can also receive conditional function to select columns. When starts with `'-'` (minus symbol) or `'-` (minus symbol), return the negative columns.

- `cols` (Optional) A numeric or character vector.

- `negate` Applicable when regular expression and `"cols"` is used. If `TRUE`, return the non-matched pattern. Default uses `FALSE`.

- `rm.dup` Should duplicated columns be removed? Defaults to `TRUE`.

**Value**

data.table

**See Also**

`select`, `select_if`

**Examples**

```r
iris %>% select_dt(Species)
iris %>% select_dt(Sepal.Length, Sepal.Width)
iris %>% select_dt(Sepal.Length:Petal.Length)
iris %>% select_dt(-Sepal.Length)
iris %>% select_dt(-Sepal.Length, -Petal.Length)
iris %>% select_dt(-(Sepal.Length:Petal.Length))
iris %>% select_dt(c("Sepal.Length", "Sepal.Width"))
iris %>% select_dt(-c("Sepal.Length", "Sepal.Width"))
iris %>% select_dt(1)
iris %>% select_dt(-1)
iris %>% select_dt(1:3)
iris %>% select_dt(-(1:3))
iris %>% select_dt(c("Pe")
iris %>% select_dt(!"Se")
iris %>% select_dt(!"Se")
iris %>% select_dt("Pe", negate = TRUE)
iris %>% select_dt("Pe|Sp")
iris %>% select_dt(cols = 2:3)
iris %>% select_dt(cols = 2:3, negate = TRUE)
iris %>% select_dt(cols = c("Sepal.Length", "Sepal.Width"))
iris %>% select_dt(cols = names(iris)[2:3])
iris %>% select_dt(is.factor)
iris %>% select_dt(-is.factor)
iris %>% select_dt(!is.factor)
```

# select_mix could provide flexible mix selection
```r
select_mix(iris, Species, "Sepal.Length")
select_mix(iris, 1:2, is.factor)
```
select_mix(iris, Sepal.Length, is.numeric)
# set rm.dup to FALSE could save the duplicated column names
select_mix(iris, Sepal.Length, is.numeric, rm.dup = FALSE)

---

**separate_dt**

*Separate a character column into two columns using a regular expression separator*

**Description**

Given either regular expression, `separate_dt()` turns a single character column into two columns. Analogous to `tidyr::separate`, but only split into two columns only.

**Usage**

```r
separate_dt(
  .data,
  separated_colname,
  into,
  sep = "[^[:alnum:]]+",
  remove = TRUE
)
```

**Arguments**

- `.data` A data frame.
- `separated_colname` Column name, string only.
- `into` Character vector of length 2.
- `sep` Separator between columns.
- `remove` If TRUE, remove input column from output data frame.

**See Also**

`separate, unite_dt`

**Examples**

```r
df <- data.frame(x = c(NA, "a.b", "a.d", "b.c"))
df %>% separate_dt(x, c("A", "B"))
# equals to
df %>% separate_dt("x", c("A", "B"))

# If you just want the second variable:
df %>% separate_dt(x, into = c(NA, "B"))
```
slice_dt

Subset rows using their positions

Description
‘slice_dt()’ lets you index rows by their (integer) locations. It allows you to select, remove, and duplicate rows. It is accompanied by a number of helpers for common use cases:
* ‘slice_head_dt()’ and ‘slice_tail_dt()’ select the first or last rows.
* ‘slice_sample_dt()’ randomly selects rows.
* ‘slice_min_dt()’ and ‘slice_max_dt()’ select rows with highest or lowest values of a variable.

Usage
slice_dt(.data, ...)
slice_head_dt(.data, n)
slice_tail_dt(.data, n)
slice_max_dt(.data, order_by, n, with_ties = TRUE)
slice_min_dt(.data, order_by, n, with_ties = TRUE)
slice_sample_dt(.data, n, replace = FALSE)

Arguments
.data A data.table
... Provide either positive values to keep, or negative values to drop. The values provided must be either all positive or all negative.
n When larger than or equal to 1, the number of rows. When between 0 and 1, the proportion of rows to select.
order_by Variable or function of variables to order by.
with_ties Should ties be kept together? The default, ‘TRUE’, may return more rows than you request. Use ‘FALSE’ to ignore ties, and return the first ‘n’ rows.
replace Should sampling be performed with (‘TRUE’) or without (‘FALSE’, the default) replacement.

Value
A data.table

See Also
slice
Examples

```r
a = iris
slice_dt(a,1,2)
slice_dt(a,2:3)
slice_head_dt(a,5)
slice_head_dt(a,0.1)
slice_tail_dt(a,5)
slice_tail_dt(a,0.1)
slice_max_dt(a,Sepal.Length,10)
slice_max_dt(a,Sepal.Length,10,with_ties = FALSE)
slice_min_dt(a,Sepal.Length,10)
slice_min_dt(a,Sepal.Length,10,with_ties = FALSE)
slice_sample_dt(a,10)
slice_sample_dt(a,0.1)
```

<table>
<thead>
<tr>
<th>summarise_dt</th>
<th><strong>Summarise columns to single values</strong></th>
</tr>
</thead>
</table>

### Description

Analogous function for summarise in `dplyr`.

### Usage

```r
summarise_dt(.data, ..., by = NULL)

summarize_dt(.data, ..., by = NULL)

summarise_when(.data, when, ..., by = NULL)

summarize_when(.data, when, ..., by = NULL)

summarise_vars(.data, .cols = NULL, .func, ..., by)

summarize_vars(.data, .cols = NULL, .func, ..., by)
```

### Arguments

- `.data` : data.frame
- `...` : List of variables or name-value pairs of summary/modifications functions for `summarise_dt`. Additional parameters to be passed to parameter `.func` in `summarise_vars`.
- `by` : unquoted name of grouping variable of list of unquoted names of grouping variables. For details see `data.table`.
- `when` : An object which can be coerced to logical mode
- `.cols` : Columns to be summarised.
Function to be run within each column, should return a value or vectors with same length.

Details
summarise_vars could complete summarise on specific columns.

Value
data.table

See Also
summarise

Examples
iris %>% summarise_dt(avg = mean(Sepal.Length))
iris %>% summarise_dt(avg = mean(Sepal.Length), by = Species)
mtcars %>% summarise_dt(avg = mean(hp), by = .(cyl, vs))

# the data.table way
mtcars %>% summarise_dt(cyl_n = .N, by = .(cyl, vs)) # `.` is short for list

iris %>% summarise_vars(is.numeric, min)
iris %>% summarise_vars(!is.factor, min)
iris %>% summarise_vars(1:4, min)
iris %>% summarise_vars(.func = as.character)

iris %>% summarise_vars(is.numeric, min, by = "Species")
mtcars %>% summarise_vars(is.numeric, mean, by = "vs, am")
top_dt

Value

A character vector of the form HH:MM:SS, or SS.MMMsec if under 60 seconds. See examples.

See Also

timetaken, system.time

Examples

sys_time_print(Sys.sleep(1))

a = iris
sys_time_print({
  res = iris %>%
    mutate_dt(one = 1)
})
res

top_dt

Select top (or bottom) n rows (by value)

Description

Analogous function for top_n and top_frac in dplyr, but with a different API.

In top_dt, you can use an API for both functionalities in ‘top_n_dt()’ and ‘top_frac_dt()’.

Usage

top_dt(.data, wt = NULL, n = NULL, prop = NULL)

top_n_dt(.data, n, wt = NULL)

top_frac_dt(.data, prop, wt = NULL)

Arguments

.data data.frame

wt (Optional). The variable to use for ordering. If not specified, defaults to the last variable in the data.frame.

n Number of rows to return. Will include more rows if there are ties. If n is positive, selects the top rows. If negative, select the bottom rows.

prop Fraction of rows to return. Will include more rows if there are ties. If prop is positive, selects the top rows. If negative, select the bottom rows.
t_dt

Value

data.table

See Also

top_n

Examples

iris %>% top_n_dt(10, Sepal.Length)
iris %>% top_n_dt(-10, Sepal.Length)
iris %>% top_frac_dt(.1, Sepal.Length)
iris %>% top_frac_dt(-.1, Sepal.Length)

# For `top_dt`, you can use both modes above
iris %>% top_dt(Sepal.Length, n = 10)
iris %>% top_dt(Sepal.Length, prop = .1)

t_dt

Efficient transpose of data.frame

Description

An efficient way to transpose data frames (data.frame/data.table/tibble).

Usage

t_dt(.data)

Arguments

.data A data.frame/data.table/tibble

Details

This function would return the original data.frame structure, keeping all the row names and column names. If the row names are not available or "V1,V2..." will be provided.

Value

A transposed data.frame

Examples

    t_dt(iris)
    t_dt(mtcars)
**uncount_dt**

"Uncount" a data frame

**Description**

Performs the opposite operation to `dplyr::count()`, duplicating rows according to a weighting variable (or expression). Analogous to `tidyr::uncount`.

**Usage**

```r
uncount_dt(.data, wt, .remove = TRUE)
```

**Arguments**

- `.data`: A data.frame
- `wt`: A vector of weights.
- `.remove`: Should the column for weights be removed? Default uses TRUE.

**See Also**

`count`, `uncount`

**Examples**

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

---

**unite_dt**

Unite multiple columns into one by pasting strings together

**Description**

Convenience function to paste together multiple columns into one. Analogous to `tidyr::unite`.

**Usage**

```r
unite_dt(
  .data,
  united_colname,
  ..., 
  sep = ",",
  remove = FALSE,
  na2char = FALSE
)
```
Arguments

.data A data frame.
united_colname The name of the new column, string only.
... A selection of columns. If want to select all columns, pass "" to the parameter. See example.
sep Separator to use between values.
remove If TRUE, remove input columns from output data frame.
na2char If FALSE, missing values would be merged into NA, otherwise NA is treated as character "NA". This is different from tidyr.

See Also

unite, separate_dt

Examples

df <- expand.grid(x = c("a", NA), y = c("b", NA))
df

# Treat missing value as NA, default
df %>% unite_dt("z", x:y, remove = FALSE)
# Treat missing value as character "NA"
df %>% unite_dt("z", x:y, na2char = TRUE, remove = FALSE)
df %>%
   unite_dt("xy", x:y)

# Select all columns
iris %>% unite_dt("merged_name",""
wider_dt

Value

A data.table with characters in UTF-8 encoding

Description

Analogous function for pivot_wider in tidyr.

Usage

wider_dt(.data, ..., name, value = NULL, fun = NULL, fill = NA)

Arguments

.data A data.frame

... Optional. The unchanged group in the transformation. Could use integer vector, could receive what select_dt receives.

name Character. One column name of class to spread

value Character. One column name of value to spread. If NULL, use all other variables.

fun Should the data be aggregated before casting? Defaults to NULL, which uses length for aggregation. If a function is provided, with aggregated by this function.

fill Value with which to fill missing cells. Default uses NA.

Details

The parameter of ‘name’ and ‘value’ should always be provided and should be explicit called (with the parameter names attached).

Value

data.table

See Also

longer_dt, dcast, pivot_wider
Examples

```r
stocks = data.frame(
  time = as.Date('2009-01-01') + 0:9,
  X = rnorm(10, 0, 1),
  Y = rnorm(10, 0, 2),
  Z = rnorm(10, 0, 4)
) %>%
  longer_dt(time) -> longer_stocks

longer_stocks

longer_stocks %>%
  wider_dt("time",
            name = "name",
            value = "value")

longer_stocks %>%
  mutate_dt(one = 1) %>%
  wider_dt("time",
            name = "name",
            value = "one")

## using "fun" parameter for aggregation
DT <- data.table(v1 = rep(1:2, each = 6),
                 v2 = rep(rep(1:3, 2), each = 2),
                 v3 = rep(1:2, 6),
                 v4 = rnorm(6))

## for each combination of (v1, v2), add up all values of v4
DT %>%
  wider_dt(v1,v2,
            value = "v4",
            name = ".",
            fun = sum)
```
## Index

<table>
<thead>
<tr>
<th>Function</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add_count_dt(count_dt)</code></td>
<td>5</td>
</tr>
<tr>
<td><code>anti_join_dt(join)</code></td>
<td>19</td>
</tr>
<tr>
<td><code>arrange</code></td>
<td>3</td>
</tr>
<tr>
<td><code>arrange_dt</code></td>
<td>3</td>
</tr>
<tr>
<td><code>as_dt(in_dt)</code></td>
<td>18</td>
</tr>
<tr>
<td><code>as_fst</code></td>
<td>3</td>
</tr>
<tr>
<td><code>case_when</code></td>
<td>25</td>
</tr>
<tr>
<td><code>chop</code></td>
<td>27</td>
</tr>
<tr>
<td><code>chop_dt(nest_dt)</code></td>
<td>26</td>
</tr>
<tr>
<td><code>col_rn(rn_col)</code></td>
<td>33</td>
</tr>
<tr>
<td><code>complete</code></td>
<td>5</td>
</tr>
<tr>
<td><code>complete_dt</code></td>
<td>4</td>
</tr>
<tr>
<td><code>count</code></td>
<td>6, 42</td>
</tr>
<tr>
<td><code>count_dt</code></td>
<td>5</td>
</tr>
<tr>
<td><code>cummean</code></td>
<td>6</td>
</tr>
<tr>
<td><code>data.table</code></td>
<td>19, 38</td>
</tr>
<tr>
<td><code>dcast</code></td>
<td>44</td>
</tr>
<tr>
<td><code>delete_na_cols(drop_na_dt)</code></td>
<td>7</td>
</tr>
<tr>
<td><code>delete_na_rows(drop_na_dt)</code></td>
<td>7</td>
</tr>
<tr>
<td><code>df_mat(mat_df)</code></td>
<td>23</td>
</tr>
<tr>
<td><code>distinct</code></td>
<td>7</td>
</tr>
<tr>
<td><code>distinct_dt</code></td>
<td>6</td>
</tr>
<tr>
<td><code>drop_na</code></td>
<td>8</td>
</tr>
<tr>
<td><code>drop_na_dt</code></td>
<td>7</td>
</tr>
<tr>
<td><code>dummy_cols</code></td>
<td>10</td>
</tr>
<tr>
<td><code>dummy_dt</code></td>
<td>9</td>
</tr>
<tr>
<td><code>export_fst</code></td>
<td>10</td>
</tr>
<tr>
<td><code>fill</code></td>
<td>8</td>
</tr>
<tr>
<td><code>fill_na_dt(drop_na_dt)</code></td>
<td>7</td>
</tr>
<tr>
<td><code>filter</code></td>
<td>12</td>
</tr>
<tr>
<td><code>filter_dt</code></td>
<td>11</td>
</tr>
<tr>
<td><code>filter_fst(fst)</code></td>
<td>12</td>
</tr>
<tr>
<td><code>fst</code></td>
<td>12, 13</td>
</tr>
<tr>
<td><code>full_join_dt(join)</code></td>
<td>19</td>
</tr>
<tr>
<td><code>group_by_dt</code></td>
<td>13</td>
</tr>
<tr>
<td><code>group_dt</code></td>
<td>15</td>
</tr>
<tr>
<td><code>group_exe_dt(group_by_dt)</code></td>
<td>13</td>
</tr>
<tr>
<td><code>import_fst(export_fst)</code></td>
<td>10</td>
</tr>
<tr>
<td><code>impute_dt</code></td>
<td>16</td>
</tr>
<tr>
<td><code>in_dt</code></td>
<td>18</td>
</tr>
<tr>
<td><code>inner_join_dt(join)</code></td>
<td>19</td>
</tr>
<tr>
<td><code>intersect_dt</code></td>
<td>17</td>
</tr>
<tr>
<td><code>join</code></td>
<td>19</td>
</tr>
<tr>
<td><code>lag_dt(lead_dt)</code></td>
<td>21</td>
</tr>
<tr>
<td><code>lead</code></td>
<td>21</td>
</tr>
<tr>
<td><code>lead_dt</code></td>
<td>21</td>
</tr>
<tr>
<td><code>left_join_dt(join)</code></td>
<td>19</td>
</tr>
<tr>
<td><code>longer_dt</code></td>
<td>22, 44</td>
</tr>
<tr>
<td><code>mat_df</code></td>
<td>23</td>
</tr>
<tr>
<td><code>melt</code></td>
<td>22</td>
</tr>
<tr>
<td><code>metadata_fst</code></td>
<td>13</td>
</tr>
<tr>
<td><code>mutate</code></td>
<td>24</td>
</tr>
<tr>
<td><code>mutate_dt</code></td>
<td>24</td>
</tr>
<tr>
<td><code>mutate-vars(mutate_when)</code></td>
<td>25</td>
</tr>
<tr>
<td><code>mutate_when</code></td>
<td>25</td>
</tr>
<tr>
<td><code>nest</code></td>
<td>27</td>
</tr>
<tr>
<td><code>nest_dt</code></td>
<td>26</td>
</tr>
<tr>
<td><code>nth</code></td>
<td>28</td>
</tr>
<tr>
<td><code>object_size</code></td>
<td>29</td>
</tr>
<tr>
<td><code>parse_fst(fst)</code></td>
<td>12</td>
</tr>
<tr>
<td><code>pivot_longer</code></td>
<td>22</td>
</tr>
<tr>
<td><code>pivot_wider</code></td>
<td>44</td>
</tr>
<tr>
<td><code>pull</code></td>
<td>30</td>
</tr>
<tr>
<td><code>pull_dt</code></td>
<td>29</td>
</tr>
<tr>
<td><code>read_fst</code></td>
<td>10, 11</td>
</tr>
<tr>
<td><code>relocate_dt</code></td>
<td>30</td>
</tr>
<tr>
<td><code>rename</code></td>
<td>31</td>
</tr>
<tr>
<td><code>rename_dt</code></td>
<td>31</td>
</tr>
<tr>
<td><code>replace_dt</code></td>
<td>32</td>
</tr>
</tbody>
</table>
replace_na, 8
replace_na_dt, 32
replace_na_dt (drop_na_dt), 7
right_join_dt (join), 19
rn_col, 33
rowwise_dt (group_dt), 15
sample_dt, 33
sample_frac, 34
sample_frac_dt (sample_dt), 33
sample_n, 34
sample_n_dt (sample_dt), 33
select, 35
select_dt, 4, 25, 26, 34
select_fst (fst), 12
select_if, 35
select_mix (select_dt), 34
semi_join_dt (join), 19
separate, 36
separate_dt, 36, 43
setdiff_dt (intersect_dt), 17
setequal_dt (intersect_dt), 17
setops, 18
shift, 21
shift_fill (drop_na_dt), 7
slice, 37
slice_dt, 37
slice_fst (fst), 12
slice_head_dt (slice_dt), 37
slice_max_dt (slice_dt), 37
slice_min_dt (slice_dt), 37
slice_sample_dt (slice_dt), 37
slice_tail_dt (slice_dt), 37
squeeze_dt (nest_dt), 26
summarise, 39
summarise_dt, 38
summarise_vars (summarise_dt), 38
summarise_when (summarise_dt), 38
summarize_dt (summarise_dt), 38
summarize_vars (summarise_dt), 38
summarize_when (summarise_dt), 38
summary_fst (fst), 12
sys_time_print, 39
system.time, 40
t_dt, 41
timetaken, 40
top_dt, 40
top_frac_dt (top_dt), 40