Package ‘tidyfst’

April 2, 2020

Title  Tidy Verbs for Fast Data Manipulation
Version  0.9.4
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.
       https://hope-data-science.github.io/tidyfst/
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  2020-04-02 11:50:02 UTC

R topics documented:

  arrange_dt ................................................................. 2
  as_fst ................................................................. 3
### Index

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrange_dt</td>
<td>43</td>
</tr>
<tr>
<td>complete_dt</td>
<td>4</td>
</tr>
<tr>
<td>count_dt</td>
<td>5</td>
</tr>
<tr>
<td>cummean</td>
<td>6</td>
</tr>
<tr>
<td>distinct_dt</td>
<td>6</td>
</tr>
<tr>
<td>drop_na_dt</td>
<td>7</td>
</tr>
<tr>
<td>dummy_dt</td>
<td>9</td>
</tr>
<tr>
<td>export_fst</td>
<td>10</td>
</tr>
<tr>
<td>filter_dt</td>
<td>11</td>
</tr>
<tr>
<td>fst</td>
<td>12</td>
</tr>
<tr>
<td>group_by_dt</td>
<td>13</td>
</tr>
<tr>
<td>group_dt</td>
<td>15</td>
</tr>
<tr>
<td>in_dt</td>
<td>16</td>
</tr>
<tr>
<td>lead_dt</td>
<td>17</td>
</tr>
<tr>
<td>left_join_dt</td>
<td>18</td>
</tr>
<tr>
<td>longer_dt</td>
<td>20</td>
</tr>
<tr>
<td>mat_df</td>
<td>21</td>
</tr>
<tr>
<td>mutate_dt</td>
<td>22</td>
</tr>
<tr>
<td>mutate_when</td>
<td>23</td>
</tr>
<tr>
<td>nest_dt</td>
<td>24</td>
</tr>
<tr>
<td>nth</td>
<td>26</td>
</tr>
<tr>
<td>pull_dt</td>
<td>27</td>
</tr>
<tr>
<td>relocate_dt</td>
<td>28</td>
</tr>
<tr>
<td>rename_dt</td>
<td>29</td>
</tr>
<tr>
<td>replace_dt</td>
<td>29</td>
</tr>
<tr>
<td>rn_col</td>
<td>30</td>
</tr>
<tr>
<td>sample_dt</td>
<td>31</td>
</tr>
<tr>
<td>select_dt</td>
<td>32</td>
</tr>
<tr>
<td>separate_dt</td>
<td>33</td>
</tr>
<tr>
<td>slice_dt</td>
<td>34</td>
</tr>
<tr>
<td>summarise_dt</td>
<td>35</td>
</tr>
<tr>
<td>sys_time_print</td>
<td>36</td>
</tr>
<tr>
<td>top_dt</td>
<td>37</td>
</tr>
<tr>
<td>t_dt</td>
<td>38</td>
</tr>
<tr>
<td>uncount_dt</td>
<td>39</td>
</tr>
<tr>
<td>unite_dt</td>
<td>39</td>
</tr>
<tr>
<td>utf8_encoding</td>
<td>40</td>
</tr>
<tr>
<td>wider_dt</td>
<td>41</td>
</tr>
</tbody>
</table>

### 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

```r
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.

#### 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_dt          Count observations by group

Description

Analogous function for count and add_count in dplyr.

Usage

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**

```r
cummean(x)
```

**Arguments**

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

**Examples**

```r
cummean(1:10)
```

distinct_dt

### Select distinct/unique rows in data.frame

**Description**

Analogous function for `distinct` in `dplyr`

**Usage**

```r
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**

```r
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)
```

---

**Description**

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

**Usage**

```r
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

def <- data.table(x = c(1, 2, NA), y = c("a", NA, "b"))
def %>% drop_na_dt()
def %>% drop_na_dt(x)
def %>% drop_na_dt(y)
def %>% drop_na_dt(x,y)
def %>% replace_na_dt(to = 0)
def %>% replace_na_dt(x, to = 0)
def %>% replace_na_dt(y, to = 0)
def %>% 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")

dummy_cols

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 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

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

'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).

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)
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, ...)

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

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

See Also
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)

# 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

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

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

**Usage**

```r
group_dt(.data, by = NULL, ...) rowwise_dt(.data, ...)
```

**Arguments**

- `.data` A data.frame
- `by` Variables to group by, unquoted name of grouping variable or 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**

```r
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 %>%
```
group_dt(
  by = Species,
  rbind(.SD[1],.SD[.N])
)

#' # 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 %>%
  mutate_dt(m = mean(c(x, y, z)))

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

---

**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

```r
in_dt(.data, ...)
```

```r
as_dt(.data)
```

### Arguments

- `.data`  
  A data.frame

- `...`  
lead_dt

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)
```

---

lead_dt

Fast lead/lag for vectors

Description

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

Usage

```r
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)`

### Description

Join operations.

For functions without suffix _dt, they use X[Y] syntax to join tables, and pass the "by" parameter to "on" in data.table. They have different features and syntax. They start with first letter of left/right/full/inner/semi/anti.

### Usage

- `left_join_dt(x, y, by = NULL, suffix = c(".x", ".y"))`
- `right_join_dt(x, y, by = NULL, suffix = c(".x", ".y"))`
- `inner_join_dt(x, y, by = NULL, suffix = c(".x", ".y"))`
- `full_join_dt(x, y, by = NULL, suffix = c(".x", ".y"))`
- `anti_join_dt(x, y, by = NULL)`
- `semi_join_dt(x, y, by = NULL)`
- `ljoin(x, y, by = NULL)`
- `rjoin(x, y, by = NULL)`
- `ijoin(x, y, by = NULL)`
- `fjoin(x, y, by = NULL)`
- `ajoin(x, y, by = NULL)`
- `sjoin(x, y, by = NULL)`

### Arguments

- `x` data.frame
- `y` data.frame
by  For functions without suffix _dt, this parameter will pass to the on parameter to the data.table. Details could be found at data.table. Examples included: 1. by = c("a", "b") (this is a must for set_full_join); 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

data.table

See Also

table

Examples

```r
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 non-"_dt" suffix joins is a bit different
workers %>% ijoin(positions2, by = "name==worker")
```
longer_dt

Pivot data from wide to long

Description

Analogous function for pivot_longer in tidyr.

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
     expression to match column names. It can receive 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.
- **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)

# df_mat with a data.frame

# df_mat with a named matrix

tdf %>%
  setNames(c("A","B","C")) %>%
df_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?
mutate_when

Value
data.table

See Also
mutate

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, ...)

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**.
- **.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.
- **by**
  
  (Optional) Mutate by what group?

Value

data.table
See Also

select_dt, case_when

Examples

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

```r
# 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"))

# 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**

```r
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.
pull_dt

Examples

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

---

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

```r
mtcars %>% pull_dt(2)
mtcars %>% pull_dt(cyl)
mtcars %>% pull_dt("cyl")
```
relocate_dt  

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.

Examples

```r
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_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**

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

replace_dt

**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**

```
replace_dt(.data, ..., from = 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 `NaN`.
- `to` A value. Defaults to `NA`.

Value

A data.table.

See Also

```
replace_na_dt
```

Examples

```
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**

```r
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.

**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`
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 `!`, 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(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(1, 3)
iris %>% select_dt("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
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**

`separate_dt()`
slice_dt

Slice rows in data.frame

Description
Analogous function for slice in dplyr

Usage
slice_dt(.data, ...)

Arguments
- `.data` data.frame
- `...` Integer row values.

Value
data.table

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_de("x", c("A", "B"))
```
See Also

slice

Examples

iris %>% slice_dt(1:3)
iris %>% slice_dt(1,3)
iris %>% slice_dt(c(1,3))

summarise_dt

Summarise columns to single values

Description

Analogous function for summarise in dplyr.

Usage

summarise_dt(.data, ..., by = NULL)
summarize_dt(.data, ..., 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 vari-
ables. For details see data.table

.cols Columns to be summarised.

.func 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
sys_time_print

**Description**

Convenient printing of time elapsed. A wrapper of data.table::timetaken, but showing the results more directly.

**Usage**

`sys_time_print(expr)`

**Arguments**

- `expr` Valid R expression to be timed.

**Value**

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

**See Also**

`timetaken`, `system.time`
top_dt

Examples

```r
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**

```r
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.

**Value**

data.table

**See Also**

top_n
Examples

```r
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**

```r
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**

```r
t_dt(iris)
t_dt(mtcars)
```
uncountDt

"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
uncountDt(.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))
uncountDt(df, n)
uncountDt(df, n, FALSE)
```

uniteDt

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
uniteDt(.data, united_colname, ..., sep = "_", remove = TRUE, na.rm = FALSE)
```
utf8_encoding

Use UTF-8 for character encoding in a data frame

Description

fread from data.table could not recognize the encoding and return the correct form, this could be inconvenient for text mining tasks. The utf8-encoding could use "UTF-8" as the encoding to override the current encoding of characters in a data frame.

Usage

utf8_encoding(.data)

Arguments

.data A data.frame.

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.
na.rm If TRUE, 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 character "NA"
df %>% unite_dt("z", x:y, remove = FALSE)
# Treat missing value as NA
df %>% unite_dt("z", x:y, na.rm = 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 do not carry out 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

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

add_count_dt(count_dt), 5
ajoin(left_join_dt), 18
anti_join_dt(left_join_dt), 18
arrange, 3
arrange_dt, 2
as_dt(in_dt), 16
as_fst, 3
case_when, 24
chop, 25
chop_dt(nest_dt), 24
col_rn(rn_col), 30
complete, 4
complete_dt, 4
count, 5, 39
count_dt, 5
cummean, 6
data.table, 17, 19, 35
dcast, 41
delete_na_cols(drop_na_dt), 7
delete_na_rows(drop_na_dt), 7
df_mat(mat_df), 21
distinct, 7
distinct_dt, 6
drop_na, 8
drop_na_dt, 7
dummy_cols, 10
dummy_dt, 9
export_fst, 10
fill, 8
fill_na_dt(drop_na_dt), 7
filter, 12
filter_dt, 11
filter_fst(fst), 12
fjoin(left_join_dt), 18
fst, 12, 12
full_join_dt(left_join_dt), 18
group_by_dt, 13
group_dt, 15
group_exe_dt(group_by_dt), 13
ijoin(left_join_dt), 18
import_fst(export_fst), 10
in_dt, 16
inner_join_dt(left_join_dt), 18
lag_dt(lead_dt), 17
lead, 17
lead_dt, 17
left_join, 19
left_join_dt, 18
ljoin(left_join_dt), 18
longer_dt, 20, 41
mat_df, 21
melt, 20
mutate, 23
mutate_dt, 22
mutate_vars(mutate_when), 23
mutate_when, 23
nest, 25
nest_dt, 24
nth, 26
parse_fst(fst), 12
pivot_longer, 20
pivot_wider, 41
pull, 27
pull_dt, 27
read_fst, 10, 11
relocate_dt, 28
rename, 29
rename_dt, 29
replace_dt, 29
replace_na, 8
replace_na_dt, 30
replace_na_dt(drop_na_dt), 7
right_join_dt(left_join_dt), 18
rjoin(left_join_dt), 18
rn_col, 30
rowwise_dt(group_dt), 15

sample_dt, 31
sample_frac, 31
sample_frac_dt(sample_dt), 31
sample_n, 31
sample_n_dt(sample_dt), 31
select, 32
select_dt, 4, 23, 24, 32
select_fst(fst), 12
select_if, 32
select_mix(select_dt), 32
semi_join_dt(left_join_dt), 18
separate, 34
separate_dt, 33, 40
shift, 17
shift_fill(drop_na_dt), 7
sjoin(left_join_dt), 18
slice, 35
slice_dt, 34
slice_fst(fst), 12
squeeze_dt(nest_dt), 24
summarise, 36
summarise_dt, 35
summarise_vars(summarise_dt), 35
summarize_dt(summarise_dt), 35
summarize_vars(summarise_dt), 35
sys_time_print, 36
system.time, 36

t_dt, 38
timetaken, 36
top_dt, 37
top_frac_dt(top_dt), 37
top_n, 37
top_n_dt(top_dt), 37
transmute_dt(mutate_dt), 22

unchop_dt(nest_dt), 24
uncount, 39
uncount_dt, 39
unite, 40
unite_dt, 34, 39
unnest_dt(nest_dt), 24
utf8_encoding, 40

wider_dt, 20, 41
write_fst, 10