Package ‘tidyfst’

March 14, 2020

Title  Tidy Verbs for Fast Data Manipulation
Version  0.8.8
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  7.0.2
Imports  data.table (>= 1.12.8), fst (>= 0.9.0), stringr (>= 1.4.0)
Suggests  knitr, rmarkdown, nycflights13, 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  2020-03-14 20:10:02 UTC

R topics documented:

  all_dt .......................................................... 2
  arrange_dt .................................................. 4
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>complete_dt</td>
<td>4</td>
</tr>
<tr>
<td>count_dt</td>
<td>6</td>
</tr>
<tr>
<td>cummean</td>
<td>7</td>
</tr>
<tr>
<td>distinct_dt</td>
<td>7</td>
</tr>
<tr>
<td>drop_na_dt</td>
<td>8</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>12</td>
</tr>
<tr>
<td>fst</td>
<td>12</td>
</tr>
<tr>
<td>group_by_dt</td>
<td>14</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>19</td>
</tr>
<tr>
<td>mutate_dt</td>
<td>21</td>
</tr>
<tr>
<td>mutate_when</td>
<td>22</td>
</tr>
<tr>
<td>nest_dt</td>
<td>23</td>
</tr>
<tr>
<td>nth</td>
<td>25</td>
</tr>
<tr>
<td>pull_dt</td>
<td>26</td>
</tr>
<tr>
<td>relocate_dt</td>
<td>26</td>
</tr>
<tr>
<td>rename_dt</td>
<td>27</td>
</tr>
<tr>
<td>sample_dt</td>
<td>28</td>
</tr>
<tr>
<td>select_dt</td>
<td>29</td>
</tr>
<tr>
<td>separate_dt</td>
<td>30</td>
</tr>
<tr>
<td>set_dt</td>
<td>31</td>
</tr>
<tr>
<td>slice_dt</td>
<td>33</td>
</tr>
<tr>
<td>summarise_dt</td>
<td>34</td>
</tr>
<tr>
<td>sys_time_print</td>
<td>35</td>
</tr>
<tr>
<td>top_dt</td>
<td>35</td>
</tr>
<tr>
<td>t_dt</td>
<td>36</td>
</tr>
<tr>
<td>uncount_dt</td>
<td>37</td>
</tr>
<tr>
<td>unite_dt</td>
<td>38</td>
</tr>
<tr>
<td>wider_dt</td>
<td>39</td>
</tr>
</tbody>
</table>

**Index**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>all_dt</td>
<td>41</td>
</tr>
</tbody>
</table>

**Description**

In `dplyr`, there are functions like `mutate_at`, `mutate_if` and `mutate_all`, as well as `summarise_at`, `summarise_if` and `summarise_all`. Here we provide a simple way to do both in `all_dt`, `at_dt` and `if_dt`. **Wise mutate or summarise with "all-at-if"**
Usage

all_dt(data, .func, .funcs, ...)
at_dt(data, .at, .func, .funcs, ...)
if_dt(data, .if, .func, .funcs, ...)

Arguments

data       A data.frame
.func      Main function name, either "mutate(_dt)" or "summarise(_dt)"
.funcs     Function to be run within each column, should return a value or vectors with same length.
...        Parameters to be passed to parameter `.funcs`.
.at        Variables to select, could use regular expression or numeric/character vector.
.if        Conditional function to select columns. When starts with `-'(minus symbol), return the negative columns.

Details

Always return the columns with their original names after mutate or summarise.

Value

A data.table

Examples

# all_
all_dt(iris[, -5], mutate_dt, scale)
all_dt(iris[, -5], mutate, scale)
all_dt(iris[, -5], summarise, max)
all_dt(iris[, -5], summarise, min)

# at_
iris %>% at_dt("Se", mutate, scale)
iris %>% at_dt(1:3, mutate, scale)
iris %>% at_dt(c("Petal.Length"), mutate, scale, center = FALSE)

# if_
iris %>%
  if_dt(is.double, mutate_dt, scale, center = FALSE)
  ## support minus symbol to select negative conditions
  iris %>%
  if_dt(!is.factor, mutate_dt, scale, center = FALSE)
  iris %>%
  if_dt(is.factor, mutate, as.character)
  iris %>%
  if_dt(is.numeric, summarise, max, na.rm = TRUE)
**arrange_dt**  
*Arrange entries in data.frame*

**Description**  
Analogous function for `arrange` in `dplyr`.

**Usage**  
```
arrange_dt(data, ...)
```

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

**Value**
- data.table

**See Also**
- `arrange`, `dt_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)
```

---

**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)
```
complete_dt

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 lengths, 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 observations by group*

**Description**

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

**Usage**

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

```r
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")
```

```r
mtcars %>% count_dt(cyl, vs)
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)

**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)
```

---

**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**

```r
drop_na_dt(data, ...)
drop_all_na_cols(data)
drop_all_na_rows(data)
replace_na_dt(data, ..., to)
fill_na_dt(data, ..., direction = c("down", "up"))
```

**Arguments**

- `data` : data.frame
- `...` : Columns to be replaced or filled. If not specified, use all columns.
- `to` : What value should NA replace by?
- `direction` : Direction in which to fill missing values. Currently either "down" (the default) or "up".

**Details**

`drop_all_na_cols` could drop the columns with only NAs, while `drop_all_na_rows` could drop the rows with only NAs.

**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

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")

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? De-
           fault 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.

Value

data.table

See Also

dummy_cols

Examples

```r
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

```r
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
)
```
export_fst

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 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

```r
## 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**

```r
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**

```r
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**

```r
parse_fst(path)
slice_fst(ft, row_no)
select_fst(ft, ...)
filter_fst(ft, ...)
```
**fst**

**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

# 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(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)
```
**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, inplace = FALSE)

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`.
- **cols** A character vector of column names to group by.
- **inplace** Should the grouping implemented by reference? (Modify the original data.frame) Default uses FALSE.

**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

**Examples**

```r
# group by Species in iris data set
as.data.table(iris) -> a
key(a)
group_by_dt(a, Species, inplace = FALSE)
key(a)

# use inplace operation to group by reference
as.data.table(iris) -> a
key(a)
group_by_dt(a, Species, inplace = TRUE)
key(a)
```
# 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))
)

---

**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.

**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,
\[
\text{mutate_dt}(\text{max} = \text{max}(\text{Sepal.Length})) \%\% \\
\text{summarise_dt}(\text{sum} = \text{sum}(\text{Sepal.Length}))
\]

# for users familiar with data.table, you can work on .SD directly
# following codes get the first and last row from each group
\[
\text{iris} \%\%
\text{group_dt}(\\
\quad \text{by} = \text{Species}, \\
\quad \text{rbind(.SD[1],.SD[.N])}
\)
\]

#' # for summarise_dt, you can use "by" to calculate within the group
\[
\text{mtcars} \%\%
\text{summarise_dt(}\\
\quad \text{disp} = \text{mean}(\text{disp}), \\
\quad \text{hp} = \text{mean}(\text{hp}), \\
\quad \text{by} = \text{cyl}
\)
\]

# but you could also, of course, use group_dt
\[
\text{mtcars} \%\%
\text{group_dt(by = .(vs,am),}\\
\quad \text{summarise_dt(avg = mean(mpg))})
\]

# and list of variables could also be used
\[
\text{mtcars} \%\%
\text{group_dt(by = list(vs,am),}\\
\quad \text{summarise_dt(avg = mean(mpg))})
\]

# examples for `rowwise_dt`
\[
\text{df} \leftarrow \text{data.table(x} = 1:2, \text{y} = 3:4, \text{z} = 4:5)
\]
\[
\text{df} \%\% \text{mutate_dt(m} = \text{mean(c(x, y, z)))}
\]
\[
\text{df} \%\% \text{rowwise_dt(}\\
\quad \text{mutate_dt(m} = \text{mean(c(x, y, z)))}
\)
\]

---

**Description**

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

**Usage**

\[
\text{in_dt(data, ...)}
\]
\[
\text{as_dt(data)}
\]
lead_dt

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 recieves 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

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

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)

left_join_dt  Join table by common keys

Description

Join operations.

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)

Arguments

x    data.frame
y    data.frame
by   a character vector of variables to join by. If NULL, the default, *_join() will do a natural join, using all variables with common names across the two tables. A message lists the variables so that you can check they’re right (to suppress the message, simply explicitly list the variables that you want to join). 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.
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
longer_dt

See Also

left_join

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"))
```

Description

Analogous function for `pivot_longer` in `tidyr`.

Usage

```r
longer_dt(
  data,
  ..., 
gathered_name = "name",
```

Pivot data from wide to long

Pivot data from wide to long
longer_dt

gathered_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.

gathered_name name for the measured variable names column. The default name is 'name'.
gathered_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(tidyrr)

billboard %>%
  longer_dt(
    "wk", negate = TRUE,
    gathered_name = "week",
    gathered_value = "rank",
)
mutate_dt

Mutate columns in data.frame

Description
Analogous function for mutate and transmute in dplyr.

Usage
mutate_dt(data, ...)
transmute_dt(data, ...)

Arguments
- data: data.frame
- ...: List of variables or name-value pairs of summary/modifications functions.

Value
data.table

See Also
mutate
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, ...)

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.

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(1:2, scale)
iris %>% mutate_vars(.func = as.character)
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

```r
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 receive 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 original 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"))

# 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?
nth

```
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**

```
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**

```
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(1)  
mtcars %>% pull_dt(cyl)  
mtcars %>% pull_dt("cyl")
```

---

**relocate_dt**  
*Change column order*

**Description**

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

**Usage**

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

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

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, ...)
sample_dt

Arguments

- **data**: data.frame
- **...**: statements 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()
```

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
select_dt

See Also

sample_n, sample_frac

Examples

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

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 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

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(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("Pe", negate = TRUE)
iris %>% select_dt("Pe|Sp")
iris %>% select_dt(cols = 2:3)
iris %>% select_dt(cols = names(iris)[2:3])
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(
  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"))
```

---

### set_dt

*Fast operations of data.table by reference*

**Description**

Combination of set* functions provided by `data.table`. This is memory efficient because no copy is made at all.

**Usage**

```r
set_dt(
  data,
  col_order = NULL,
  order_by = NULL,
  order_dir = 1L,
  old_name = NULL,
  new_name = NULL,
  fill_cols = NULL,
  fill_type = "down",
  fill_value = NA
)
```

`mutate_ref(data, ..., by)`
Arguments

data
A data.frame

col_order
(Optional) Character vector of the new column name ordering. May also be column numbers. This parameter will pass to "neworder" parameter in setcolorder.

order_by
(Optional) A character vector of column names by which to order. This parameter will pass to "cols" parameter in setorder.

order_dir
(Optional) An integer vector with only possible values of 1 and -1, corresponding to ascending and descending order. This parameter will pass to "order" parameter in setorder. Default uses 1.

old_name
(Optional) When new_name is provided, character names or numeric positions of column names to change. This parameter will pass to "old" parameter in setnames.

new_name
(Optional) It can be a function or the new column names. This parameter will pass to "new" parameter in setnames.

fill_cols
(Optional) Numeric or character vector specifying columns to be updated.

fill_type
(Optional) Character, one of "down", "up" or "replace". Defaults to "down".

fill_value
(Optional) Numeric or integer, value to be used to fill when fill_type=="replace". Defaults to NA.

... List of variables or name-value pairs of modifications functions.

by Mutate by which group(s)?

Details

The set_dt() will first set any data.frame to a data.table, then rename, fill NAs, arrange row order, arrange column order. If you want to do the operation in another order, use it separately in multiple set_dt functions in the desired order.

The mutate_ref() will first set any data.frame to a data.table, then mutate values by reference.

Value

The input is modified by reference, and returned (invisibly) so it can be used in compound statements.

See Also

setcolorder, setorder, setnames, setnafill

Examples

# set_dt
x = 1:10
x[c(1:2, 5:6, 9:10)] = NA
dt = data.table(v1=x, v2=lag_dt(x)/2, v3=lead_dt(x, 1L)/2)
dt
set_dt(dt,new_name = c("A","B","C"),fill_cols = names(dt),
slice_dt

order_by = "A", order_dirc = -1, col_order = c("B", "A", "C")

dt

# mutate_ref
copy(iris) -> a
# never use pipe(%) for the `mutate_ref` function
mutate_ref(a, one = 1, two = 2)
a

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

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**

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

**Arguments**

- `data` *data.frame*
- `...` List of variables or name-value pairs of summary/modifications functions.
- `by` unquoted name of grouping variable of list of unquoted names of grouping variables. For details see **data.table**

**Value**

`data.table`

**See Also**

`summarise`

**Examples**

```r
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
```
### sys_time_print

**Convenient print of time taken**

**Description**

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

**Usage**

```r
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`

**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)
```

---

```r
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)
```
uncount_dt

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

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

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

unite_dt(data, united_colname, ..., sep = " ", remove = TRUE, na.rm = 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.
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  

Pivot data from long to wide

Description

Analogous function for pivot_wider in tidyrl.

Usage

wider_dt(
  data,
  ..., 
  name_to_spread,
  value_to_spread = NULL,
  fun = identity,
  fill = NA
)

Arguments

data  
data.table

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

name_to_spread  
Character. One column name of class to spread

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

fun  
Should the data be aggregated before casting? If the formula doesn't identify a single observation for each cell, then aggregation defaults to length with a message. To use multiple aggregation functions, pass a list.

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

Details

The parameter of 'name_to_spread' and 'value_to_spread' 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

```

```r
longer_stocks %>%
  wider_dt("time",
            name_to_spread = "name",
            value_to_spread = "value")

```

```r
longer_stocks %>%
  mutate(one = 1) %>%
  wider_dt("time",
            name_to_spread = "name",
            value_to_spread = "one")
```

```r
## 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_to_spread = "v4",
           name_to_spread = ".",
           fun = sum)
```
Index

add_count_dt (count_dt), 6
all_dt, 2
anti_join_dt (left_join), 18
arrange, 4
arrange_dt, 4
as_dt (in_dt), 16
at_dt (all_dt), 2
case_when, 22
chop, 24
chop_dt (nest_dt), 23
count, 6, 37
count_dt, 6
cummean, 7
data.table, 17, 34
dcast, 39
distinct, 7
distinct_dt, 7
drop_all_na_cols (drop_na_dt), 8
drop_all_na_rows (drop_na_dt), 8
drop_na, 9
drop_na_dt, 8
dt_arrange, 4
dummy_cols, 10
dummy_dt, 9
export_fst, 10
fill, 9
fill_na_dt (drop_na_dt), 8
filter, 12
filter_dt, 12
filter_fst (fst), 12
fst, 12, 13
full_join_dt (left_join), 18
group_by_dt, 14
group_dt, 15
group_exe_dt (group_by_dt), 14
if_dt (all_dt), 2
import_fst (export_fst), 10
in_dt, 16
inner_join_dt (left_join), 18
lag_dt (lead_dt), 17
lead, 18
lead_dt, 17
left_join, 19
left_join_dt, 18
longer_dt, 19, 39
melt, 20
mutate, 21
mutate_dt, 21
mutate_ref (set_dt), 31
mutate_vars (mutate_when), 22
mutate_when, 22
nest, 24
nest_dt, 23
nth, 25
parse_fst (fst), 12
pivot_longer, 20
pivot_wider, 39
pull, 26
pull_dt, 26
read_fst, 10, 11
relocate_dt, 26
rename, 28
rename_dt, 27
replace_na, 9
replace_na_dt (drop_na_dt), 8
right_join_dt (left_join), 18
rowwise_dt (group_dt), 15
sample_dt, 28
INDEX

sample_frac, 29
sample_frac_dt (sample_dt), 28
sample_n, 29
sample_n_dt (sample_dt), 28
select, 29
select_dt, 5, 22, 23, 29
select_fst (fst), 12
select_if, 29
select_mix (select_dt), 29
semi_join_dt (left_join_dt), 18
separate, 31
separate_dt, 30, 38
set_dt, 31
setcolorder, 32
setnafill, 32
setnames, 32
setorder, 32
shift, 18
slice, 33
slice_dt, 33
slice_fst (fst), 12
squeeze_dt (nest_dt), 23
summarise, 34
summarise_dt, 34
summarize_dt (summarise_dt), 34
sys_time_print, 35
system.time, 35
t_dt, 36
timetaken, 35
top_dt, 35
top_frac_dt (top_dt), 35
top_n, 36
top_n_dt (top_dt), 35
transmute_dt (mutate_dt), 21
unchop_dt (nest_dt), 23
uncount, 37
uncount_dt, 37
unite, 38
unite_dt, 31, 38
unnest_dt (nest_dt), 23
wider_dt, 20, 39
write_fst, 10