Package ‘tidyfst’

June 14, 2024

**Title**  Tidy Verbs for Fast Data Manipulation

**Version**  1.7.9

**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.

**URL**  https://github.com/hope-data-science/tidyfst,
https://hope-data-science.github.io/tidyfst/

**BugReports**  https://github.com/hope-data-science/tidyfst/issues

**License**  MIT + file LICENSE

**Encoding**  UTF-8

**RoxygenNote**  7.3.1

**Imports**  data.table (>= 1.13.0), fst (>= 0.9.0), stringr (>= 1.5.0)

**Suggests**  knitr, rmarkdown, nycflights13, pryr, tidyr, ggplot2, dplyr,
bench, testthat

**VignetteBuilder**  knitr

**NeedsCompilation**  no

**Author**  Tian-Yuan Huang [aut, cre] (<https://orcid.org/0000-0002-4151-3764>)

**Maintainer**  Tian-Yuan Huang <huang.tian-yuan@qq.com>

**Repository**  CRAN

**Date/Publication**  2024-06-14 05:00:02 UTC

**Contents**

<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrange_dt</td>
<td>3</td>
</tr>
<tr>
<td>as_fst</td>
<td>3</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------</td>
</tr>
<tr>
<td>col_max</td>
<td>4</td>
</tr>
<tr>
<td>complete_dt</td>
<td>5</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>10</td>
</tr>
<tr>
<td>export_fst</td>
<td>11</td>
</tr>
<tr>
<td>filter_dt</td>
<td>13</td>
</tr>
<tr>
<td>fst</td>
<td>14</td>
</tr>
<tr>
<td>group_by_dt</td>
<td>15</td>
</tr>
<tr>
<td>group_dt</td>
<td>16</td>
</tr>
<tr>
<td>import_fst_chunked</td>
<td>18</td>
</tr>
<tr>
<td>impute_dt</td>
<td>19</td>
</tr>
<tr>
<td>intersect_dt</td>
<td>20</td>
</tr>
<tr>
<td>in_dt</td>
<td>21</td>
</tr>
<tr>
<td>join</td>
<td>22</td>
</tr>
<tr>
<td>lead_dt</td>
<td>24</td>
</tr>
<tr>
<td>longer_dt</td>
<td>25</td>
</tr>
<tr>
<td>mat_df</td>
<td>26</td>
</tr>
<tr>
<td>mutate_dt</td>
<td>27</td>
</tr>
<tr>
<td>mutate_when</td>
<td>28</td>
</tr>
<tr>
<td>nest_dt</td>
<td>29</td>
</tr>
<tr>
<td>nth</td>
<td>31</td>
</tr>
<tr>
<td>object_size</td>
<td>32</td>
</tr>
<tr>
<td>pairwise_count_dt</td>
<td>33</td>
</tr>
<tr>
<td>percent</td>
<td>34</td>
</tr>
<tr>
<td>pkg_load</td>
<td>35</td>
</tr>
<tr>
<td>print_options</td>
<td>36</td>
</tr>
<tr>
<td>pull_dt</td>
<td>37</td>
</tr>
<tr>
<td>rec</td>
<td>38</td>
</tr>
<tr>
<td>relocate_dt</td>
<td>39</td>
</tr>
<tr>
<td>rename_dt</td>
<td>40</td>
</tr>
<tr>
<td>replace_dt</td>
<td>41</td>
</tr>
<tr>
<td>rn_col</td>
<td>42</td>
</tr>
<tr>
<td>sample_dt</td>
<td>42</td>
</tr>
<tr>
<td>select_dt</td>
<td>44</td>
</tr>
<tr>
<td>separate_dt</td>
<td>45</td>
</tr>
<tr>
<td>slice_dt</td>
<td>46</td>
</tr>
<tr>
<td>sql_join</td>
<td>48</td>
</tr>
<tr>
<td>summarise_dt</td>
<td>49</td>
</tr>
<tr>
<td>sys_time_print</td>
<td>50</td>
</tr>
<tr>
<td>t_dt</td>
<td>51</td>
</tr>
<tr>
<td>uncount_dt</td>
<td>52</td>
</tr>
<tr>
<td>unite_dt</td>
<td>52</td>
</tr>
<tr>
<td>utf8_encoding</td>
<td>53</td>
</tr>
<tr>
<td>wider_dt</td>
<td>54</td>
</tr>
</tbody>
</table>

**Index** 56
**arrange_dt**

Arrange entries in data.frame

---

**Description**

Order the rows of a data frame rows by the values of selected columns.

**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)
```

col_max

Get the column name of the max/min number each row

Description

For a data.frame with numeric values, add a new column specifying the column name of the first max/min value each row.

Usage

```r
col_max(.data, .name = "max_col")
col_min(.data, .name = "min_col")
```

Arguments

.data A data.frame with numeric column(s)

.name The column name of the new added column

Value

A data.table

References

https://stackoverflow.com/questions/17735859/for-each-row-return-the-column-name-of-the-largest-value
complete_dt

Examples

```r
set.seed(199057)
DT <- data.table(matrix(sample(10, 100, TRUE), ncol=10))
DT
col_max(DT)
col_max(DT, .name = "max_col_name")
col_min(DT)
col_max(iris)
```

---

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

Description

Turns implicit missing values into explicit missing values. All the combinations of column values (should be unique) will be constructed. Other columns will be filled with NAs or constant value.

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 observations by group

Description

Count the unique values of one or more variables.

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, could receive what `select_dt` receives.

.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
cummean

Examples

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

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

Select only unique/distinct rows from a data frame.

Usage

```r
distinct_dt(.data, ..., .keep_all = FALSE, fromLast = 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.

fromLast Logical indicating if duplication should be considered from the reverse side. Defaults to FALSE.

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

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

Description

A set of tools to deal with missing values in data.frames. It can dump, replace, fill (with next or previous observation) or delete entries according to their missing values.

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)
**drop_na_dt**

```r
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)
```
```r
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. When NA exist in the input column, they would also be considered. If the input character column contains both NA and string "NA", they would be merged.

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

References


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")

# when there are NAs in the column
df <- data.table(x = c("a", "b", NA, NA),y = 1:4)
df %>%
dummy_dt(x)

# when NA and "NA" both exist, they would be merged
df <- data.table(x = c("a", "b", NA, "NA"),y = 1:4)
df %>%
dummy_dt(x)
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
)
```

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`
filter_dt

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

Choose rows where conditions are true.

### 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 == max(Sepal.Length))

# comma is not supported in tidyfst after v0.9.8
# which means you can't use:
## Not run:
iris %>% filter_dt(Sepal.Length > 7, Sepal.Width > 3)

## End(Not run)
# use following code instead
iris %>% filter_dt(Sepal.Length > 7 & Sepal.Width > 3)
```
fst

Parse, inspect and extract data.table from fst file

Description
A toolkit of APIs for reading fst file as data.table, could select by column, row and conditional filtering.

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.

See Also
fst, metadata_fst

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

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**

Carry out data manipulation within specified groups. Different from group_dt, the implementation is split into two operations, namely grouping and implementation.

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))
)
```

Description

Carry out data manipulation within specified groups.
**group_dt**

**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 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**

```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),
```
import_fst_chunked

Read a fst file by chunks

Description

For `import_fst_chunked`, if a large fst file which could not be imported into the memory all at once, this function could read the fst file by chunks and preprocessed the chunk to ensure the results yielded by the chunks are small enough to be summarised in the end. For `get_fst_chunk_size`, this function can measure the memory used by a specified row number.

Usage

```r
import_fst_chunked(
  path,
  chunk_size = 10000L,
  chunk_f = identity,
  combine_f = rbindlist
)
```

```r
get_fst_chunk_size(path, nrows)
```

Arguments

- `path`: Path to fst file
- `chunk_size`: Integer. The number of rows to include in each chunk
- `chunk_f`: A function implemented on every chunk.
**impute_dt**

Impute missing values with mean, median or mode.

**Description**

Impute the columns of data.frame with its mean, median or mode.

**Usage**

`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.

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', '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', 'C', 'S')
data <- data.frame('Pclass' = Pclass,
                   'Sex' = Sex, 'Age' = Age, 'SibSp' = SibSp,
                   'Fare' = Fare, 'Embarked' = Embarked)

data
```

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

```r
my_fun = function(x){
  x[is.na(x)] = (max(x,na.rm = TRUE) - min(x,na.rm = TRUE))/2
  x
}
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

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

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 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)

---

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

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 vari-
   ables 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("
lead_dt

```r
name position
John designer
Daniela engineer
Cathie manager
```

```r
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**

Find the "next" or "previous" values in a vector. It has wrapped `data.table`'s `shift` function.

**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
longer_dt

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

Turning a wide table to its longer form. It takes multiple columns and collapses into key-value pairs.

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 receives.

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

### Example 2:
```r
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
mutate_dt

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

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)

mutate_dt

Mutate columns in data.frame

Description

Adds or updates columns in data.frame.

Usage

mutate_dt(.data, ..., by)

transmute_dt(.data, ..., by)
mutate_when

**Conditional update of columns in data.table**

**Description**
Update or add columns when the given condition is met.

`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**

```r
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**

Create or melt list columns in data.frame.

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, .name = "ndt")

unnest_dt(.data, ...)

squeeze_dt(.data, ..., .name = "ndt")

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.
**nest_dt**

```
mcols Name-variable pairs in the list, form like
.name Character. The nested column name. Defaults to "ndt".
```

**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 `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"))

# change the nested column name
mtcars %>% nest_dt(cyl,.name = "data")

# 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)
iris %>% squeeze_dt(1:2,.name = "data")

# 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**

`nth` get the value from a vector with its position, while `maxth` and `minth` get the nth highest or lowest value from the vector.

**Usage**

nth(v, n = 1)

maxth(v, n = 1)

minth(v, n = 1)
object_size

Arguments

\(v\)  A vector

\(n\)  For \(n\)th, 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. For \(maxth\) and \(minth\), a single integer indicating the \(n\)th highest or lowest value.

Value

A single value.

References


Examples

\[
x = 1:10
\]
\[
nth(x, 1)
nth(x, 5)
nth(x, -2)
\]
\[
y = c(10,3,4,5,2,1,6,9,7,8)
maxth(y,3)
minth(y,3)
\]

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

object_size(object)

Arguments

object  an R object.

Value

An object of class "object_size"
**Examples**

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

---

**pairwise_count_dt**  
*Count pairs of items within a group*

**Description**

Count the number of times each pair of items appear together within a group. For example, this could count the number of times two words appear within documents. This function has referred to `pairwise_count` in `widyr` package, but with very different defaults on several parameters.

**Usage**

```r
pairwise_count_dt(.data, .group, .value, upper = FALSE, diag = FALSE, sort = TRUE)
```

**Arguments**

- `.data` A data.frame.
- `.group` Column name of counting group.
- `.value` Item to count pairs, will end up in V1 and V2 columns.
- `upper` When FALSE (Default), duplicated combinations would be removed.
- `diag` Whether to include diagonal (V1==V2) in output. Default uses FALSE.
- `sort` Whether to sort rows by counts. Default uses TRUE.

**Value**

A data.table with 3 columns (named as "V1","V2" and "n"), containing combinations in "V1" and "V2", and counts in "n".

**See Also**

`pairwise_count`
Examples

dat <- data.table(group = rep(1:5, each = 2),
                  letter = c("a", "b",
                             "a", "c",
                             "a", "c",
                             "b", "e",
                             "b", "f"))
pairwise_count_dt(dat, group, letter)
pairwise_count_dt(dat, group, letter, sort = FALSE)
pairwise_count_dt(dat, group, letter, upper = TRUE)
pairwise_count_dt(dat, group, letter, diag = TRUE)
pairwise_count_dt(dat, group, letter, diag = TRUE, upper = TRUE)

# The column name could be specified using character.
pairwise_count_dt(dat,"group","letter")

percent

Add percentage to counts in data.frame

Description

Add percentage for counts in the data.frame, both numeric and character with ‘

Usage

percent(x, digits = 1)

add_prop(.data, count_name = last(names(.data)), digits = 1)

Arguments

x A number (numeric).
digits How many digits to keep in the percentage. Default uses 1.
.data A data frame.
count_name Column name of counts (Character). Default uses the last column of data.frame.

References

https://stackoverflow.com/questions/7145826/how-to-format-a-number-as-percentage-in-r

Examples

percent(0.9057)
percent(0.9057,3)

iris %>%
count_dt(Species) %>%
add_prop()
iris %>%
  count_dt(Species) %>%
  add_prop(count_name = "n", digits = 2)

pkg_load

Load or unload R package(s)

Description

This function is a wrapper for `require` and `detach`. `pkg_load` checks to see if a package is installed, if not it attempts to install the package from CRAN. `pkg_unload` can detach one or more loaded packages.

Usage

```r
pkg_load(..., pkg_names = NULL)
pkg_unload(..., pkg_names = NULL)
```

Arguments

- `...` Name(s) of package(s).
- `pkg_names` (Optional) Character vector containing packages to load or unload. Default uses `NULL`.

See Also

`require`, `detach`, `p_load`, `p_unload`

Examples

```r
## Not run:
pkg_load(data.table)
pkg_unload(data.table)

pkg_load(stringr,fst)
pkg_unload(stringr,fst)

pkg_load(pkg_names = c("data.table","fst"))
p_unload(pkg_names = c("data.table","fst"))

pkg_load(data.table,stringr,fst)
pkg_unload("all") # shortcut to unload all loaded packages

## End(Not run)
```
**print_options**  
Set global printing method for data.table

**Description**

This function allows you to define how `data.table` is printed.

**Usage**

```r
default_print_options()
print_options(
  topn = 5,
  nrows = 100,
  class = TRUE,
  row.names = TRUE,
  col.names = "auto",
  print.keys = TRUE,
  trunc.cols = FALSE
)
```

**Arguments**

- **topn**: The number of rows to be printed from the beginning and end of tables with more than `nrow` rows.
- **nrows**: The number of rows which will be printed before truncation is enforced.
- **class**: If `TRUE`, the resulting output will include above each column its storage class (or a self-evident abbreviation thereof).
- **row.names**: If `TRUE`, row indices will be printed.
- **col.names**: One of three flavours for controlling the display of column names in output. "auto" includes column names above the data, as well as below the table if `nrow(x) > 20`. "top" excludes this lower register when applicable, and "none" suppresses column names altogether (as well as column classes if `class = TRUE`).
- **print.keys**: If `TRUE`, any key and/or index currently assigned to `x` will be printed prior to the preview of the data.
- **trunc.cols**: If `TRUE`, only the columns that can be printed in the console without wrapping the columns to new lines will be printed (similar to `tibbles`).

**Details**

Notice that `tidyfst` has a slightly different printing default for `data.table`, which is it always prints the keys and variable class (not like `data.table`).

**Value**

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

See Also

print.data.table

Examples

iris %>% as.data.table()
print_options(topn = 3, trunc.cols = TRUE)
iris %>% as.data.table()

# set all settings to default in tidyfst
print_options()
iris %>% as.data.table()

pull_dt(.data, col)

Description

Extract vector from data.frame, works likt `[]`. Analogous function for pull in dplyr

Usage

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")
Recode number or strings

**Description**

Recode discrete variables, including numeric and character variables.

**Usage**

```r
rec_num(x, rec, keep = TRUE)
rec_char(x, rec, keep = TRUE)
```

**Arguments**

- `x` A numeric or character vector.
- `rec` String with recode pairs of old and new values. Find the usage in examples.
- `keep` Logical. Decide whether to keep the original values if not recoded. Defaults to `TRUE`.

**Value**

A vector.

**See Also**

`rec`

**Examples**

```r
x = 1:10
x
rec_num(x, rec = "1=10; 4=2")
rec_num(x, rec = "1:3=1; 4:6=2")
rec_num(x, rec = "1:3=1; 4:6=2", keep = FALSE)

y = letters[1:5]
y
rec_char(y, rec = "a=A; b=B")
rec_char(y, rec = "a,b=A; c,d=B")
rec_char(y, rec = "a,b=A; c,d=B", keep = FALSE)
```
Change column order

Description

Change the position of columns, using the same syntax as `select_dt()`. Check similar function as ‘relocate’ in dplyr.

Usage

```r
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.

See Also

relocate

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",
```
rename_dt

Rename column in data.frame

Description

Rename one or more columns in the data.frame.

Usage

```r
tools::options(set.printer = c(high = TRUE))

rename_dt(.data, ...)
rename_with_dt(.data, .fn, ...)
```

Arguments

- `.data` data.frame
- `...` statements of rename, e.g. `sl = Sepal.Length` means the column named as "Sepal.Length" would be renamed to "sl"
- `.fn` A function used to transform the selected columns. Should return a character vector the same length as the input.

Value

data.table

See Also

tools::options

tools::print

tools::print_high

tools::print_higher

Examples

```r
iris %>%
  rename_dt(sl = Sepal.Length, sw = Sepal.Width) %>%
  head()
iris %>% rename_with_dt(toupper)
iris %>% rename_with_dt(toupper, "Pe")
```
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 = 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
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**

```r
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 rows randomly from a table*

**Description**

Select a number or proportion of rows randomly from the data frame  

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

Usage

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

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.
by (Optional) Character. Specify if you want to sample by group.
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

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)

# sample by group(s)
iris %>% sample_n_dt(2, by = "Species")
iris %>% sample_frac_dt(.1, by = "Species")

mtcars %>% sample_n_dt(1, by = c("cyl","vs"))
Description

Select specific column(s) via various ways. One can select columns by their column names, indexes or regular expression recognizing the column name(s).

Usage

```r
select_dt(.data, ..., cols = NULL, negate = FALSE)
```

```r
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(-(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)
```
separate_dt

separate_dt(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.

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

Arguments
.data A data frame.
separated_colname Column to be separated, can be a character or alias.
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

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

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

a = iris
slice_dt(a,1,2)
slice_dt(a,2:3)
slice_dt(a,141:.N)
slice_dt(a,1,.N)
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)

# use by to slice by group

## following codes get the same results
slice_dt(a,1:3,by = "Species")
slice_dt(a,1:3,by = Species)
slice_dt(a,1:3,by = .(Species))
slice_head_dt(a,2,by = Species)
slice_tail_dt(a,2,by = Species)
slice_max_dt(a,3,by = Species)
slice_max_dt(a,Sepal.Length,3,by = Species,with_ties = FALSE)
slice_min_dt(a,Sepal.Length,3,by = Species)
slice_min_dt(a,Sepal.Length,3,by = Species,with_ties = FALSE)

# in `slice_sample_dt`, "by" could only take character class
slice_sample_dt(a,.1,by = "Species")
slice_sample_dt(a,3,by = "Species")
slice_sample_dt(a,51,replace = TRUE,by = "Species")
sql_join

Case insensitive table joining like SQL

Description

Work like the `*_join_dt` series functions, joining tables with common or customized keys in various ways. The only difference is the joining is case insensitive like SQL.

Usage

```r
sql_join_dt(x, y, by = NULL, type = "inner", suffix = c(".x", ".y"))
```

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'.
  Notice that in 'sql_join', the joining variables would turn to upper case in the output table.
- `type` Which type of join would you like to use? Default uses "inner", other options include "left", "right", "full", "anti", "semi".
- `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

See Also

join

Examples

```r
dt1 = data.table(x = c("A", "b"), y = 1:2)
dt2 = data.table(x = c("a", "B"), z = 4:5)
sql_join_dt(dt1, dt2)
```
**summarise_dt**

Summarise columns to single values

---

**Description**

Summarise group of values into one value for each group. If there is only one group, then only one value would be returned. The summarise function should always return a single value.

**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](https://www.r老乡.com/)

- `when`  
  An object which can be coerced to logical mode

- `.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

**See Also**

- [summarise](https://www.r老乡.com/)
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(is.numeric, min, by = "Species")
mtcars %>% summarise_vars(is.numeric, mean, by = c("vs", "am"))

# use multiple functions on multiple columns
iris %>%
  summarise_vars(is.numeric, .func = list(mean, sd, median))
iris %>%
  summarise_vars(is.numeric, .func = list(mean, sd, median), by = Species)

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

sys_time_print(expr)
pst(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 (invisibly for show_time). See examples.

See Also
timetaken, system.time
t_dt

Examples

sys_time_print(Sys.sleep(1))

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

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**

Duplicating rows according to a weighting variable. This is the opposite operation of 'count_dt'. 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.

**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",".")

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.
Value

A data.table with characters in UTF-8 encoding

**wider_dt**

*Pivot data from long to wide*

Description

Transform a data frame from long format to wide by increasing the number of columns and decreasing the number of rows.

Usage

```r
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`
wider_dt

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)
```
add_count_dt (count_dt), 6
add_prop (percent), 34
anti_join_dt (join), 22
arrange, 3
arrange_dt, 3
as_dt (in_dt), 21
as_fst, 3
case_when, 29
chop, 30
chop_dt (nest_dt), 29
col_max, 4
col_min (col_max), 4
col_rn (rn_col), 42
complete, 5
complete_dt, 5
count, 6, 52
count_dt, 6
cummean, 7
data.table, 22, 49
dcast, 54
delete_na_cols (drop_na_dt), 8
delete_na_rows (drop_na_dt), 8
detach, 35
df_mat (mat_df), 26
distinct, 8
distinct_dt, 7
drop_na, 9
drop_na_dt, 8
dummy_cols, 11
dummy_dt, 10
export_fst, 11
fill, 9
fill_na_dt (drop_na_dt), 8
filter, 13
filter_dt, 13
filter_fst (fst), 14
fst, 14, 14
full_join_dt (join), 22
get_fst_chunk_size
   (import_fst_chunked), 18
group_by_dt, 15
group_dt, 16
group_exe_dt (group_by_dt), 15
import_fst (export_fst), 11
import_fst_chunked, 18
impute_dt, 19
in_dt, 21
index, 36
inner_join_dt (join), 22
intersect_dt, 20
join, 22, 48
key, 36
lag_dt (lead_dt), 24
lead, 25
lead_dt, 24
left_join_dt (join), 22
longer_dt, 25, 54
mat_df, 26
maxth (nth), 31
melt, 25
metadata_fst, 14
minth (nth), 31
mutate, 28
mutate_dt, 27
mutate_vars (mutate_when), 28
mutate_when, 28
nest, 30
nest_dt, 29
nth, 31
INDEX

object_size, 32
p_load, 35
p_unload, 35
pairwise_count, 33
pairwise_count_dt, 33
parse_fst(fst), 14
percent, 34
pivot_longer, 25
pivot_wider, 54
pkg_load, 35
pkg_unload(pkg_load), 35
print.data.table, 37
print_options, 36
pst(sys_time_print), 50
pull, 37
pull_dt, 37

read_csv_CHUNKED, 19
read_fst, 11, 12
rec, 38, 38
rec_char(rec), 38
rec_num(rec), 38
relocate, 39
relocate_dt, 39
rename, 40
rename_dt, 40
rename_with_dt(rename_dt), 40
replace_dt, 41
replace_na, 9
replace_na_dt, 41
replace_na_dt(drop_na_dt), 8
require, 35
right_join_dt(join), 22
rn_col, 42
rowwise_dt(group_dt), 16

sample_dt, 42
sample_frac, 43
sample_frac_dt(sample_dt), 42
sample_n, 43
sample_n_dt(sample_dt), 42
select, 44
select_dt, 5, 28, 29, 44
select_fst(fst), 14
select_if, 44
select_mix(select_dt), 44
semi_join_dt(join), 22
separate, 46
separate_dt, 45, 53
setdiff_dt(intersect_dt), 20
setequal_dt(intersect_dt), 20
setops, 21
shift, 25
shift_fill(drop_na_dt), 8
slice, 47
slice_dt, 46
slice_fst(fst), 14
slice_head_dt(slice_dt), 46
slice_max_dt(slice_dt), 46
slice_min_dt(slice_dt), 46
slice_sample_dt(slice_dt), 46
slice_tail_dt(slice_dt), 46
sql_join, 48
sql_join_dt(sql_join), 48
squeeze_dt(nest_dt), 29
summarise, 49
summarise_dt, 49
summarise_vars(summarise_dt), 49
summarise_when(summarise_dt), 49
summarize_dt(summarize_dt), 49
summarize_vars(summarize_dt), 49
summarize_when(summarize_dt), 49
summary_fst(fst), 14
system_time, 50
system.time, 50
timetaken, 50
t_dt, 51
transmute_dt(mutate_dt), 27
unchop_dt(nest_dt), 29
uncount, 52
uncount_dt, 52
union_dt(intersect_dt), 20
unite, 53
unite_dt, 46, 52
unnest_dt(nest_dt), 29
utf8_encoding, 53

wider_dt, 25, 54
write_fst, 11