Package ‘tidyft’

April 10, 2020

Title Tidy Verbs for Fast Data Operations by Reference

Version 0.4.5

Description Tidy syntax for 'data.table', using modification by reference whenever possible.
   This toolkit is designed for big data analysis in high-performance desktop or laptop computers.
   The syntax of the package is similar or identical to 'tidyverse'.
   It is user friendly, memory efficient and time saving. For more information,
   check its ancestor package 'tidyfst'.

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

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

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.1.0

Imports data.table (>= 1.12.8), stringr (>= 1.4.0), fst (>= 0.9.0)

Suggests knitr, rmarkdown, profvis, dplyr

VignetteBuilder knitr

NeedsCompilation no

Author Tian-Yuan Huang [aut, cre] (<https://orcid.org/0000-0002-3591-4203>)

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

Repository CRAN

Date/Publication 2020-04-10 10:20:02 UTC

R topics documented:

  arrange ................................................................. 2
  as_fst ..................................................................... 3
  complete .................................................................. 4
  count ..................................................................... 5
arrange

Arrange entries in data.frame

Description

Analogous function for arrange in dplyr.

Usage

arrange(.data, ..., cols = NULL, order = 1L)
**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**

```r
as_fst(.data)
```

**Arguments**

- `.data` A data.frame
Value

An object of class `fst_table`

Examples

```r
iris %>%
  as_fst() -> iris_fst
iris_fst
```

Description

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

Usage

```r
complete(.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
**count**

**See Also**

- `complete`

**Examples**

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

---

**count** | *Count observations by group*

**Description**

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

**Usage**

```r
count(.data, ..., sort = FALSE, name = "n")
add_count(.data, ..., name = "n")
```

**Arguments**

- `.data` 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
Examples

```r
a = as.data.table(mtcars)
count(a, cyl)
count(a, cyl, sort = TRUE)
a

b = as.data.table(iris)
b %>% add_count(Species, name = "N")
b
```

cummean

<table>
<thead>
<tr>
<th>cummean</th>
<th>Cumulative mean</th>
</tr>
</thead>
</table>

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.

Value

A numeric vector

Examples

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

distinct

<table>
<thead>
<tr>
<th>distinct</th>
<th>Select distinct/unique rows in data.table</th>
</tr>
</thead>
</table>

Description

Analogous function for distinct in `dplyr`

Usage

```r
distinct(.data, ..., .keep_all = FALSE)
```
drop_na

Arguments

- `.data` data.table
- `...` 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.table. If a combination of ... is not distinct, this keeps the first row of values.

Value
data.table

See Also
distinct

Examples

```r
a = as.data.table(iris)
b = as.data.table(mtcars)
a %>% distinct(Species)
b %>% distinct(cyl, vs, .keep_all = TRUE)
```

---

**drop_na**

*Drop or delete data by rows or columns*

**Description**
drop_na drops entries by specified columns. delete_na deletes rows or columns with too many NAs.

**Usage**
drop_na(.data, ...)

delete_na(.data, MARGIN, n)

**Arguments**

- `.data` A data.table
- `...` Columns to be dropped or deleted.
- `MARGIN` 1 or 2. 1 for deleting rows, 2 for deleting columns.
- `n` If number (proportion) of NAs is larger than or equal to "n", the columns/rows would be deleted. When smaller than 1, use as proportion. When larger or equal to 1, use as number.
Value
A data.table

Examples

```r
data.table(x = c(1, 2, NA, 3), y = c(NA, NA, 4, 5), z = rep(NA, 4))
x %>% delete_na(2, 0.75)
```

```r
data.table(x = c(1, 2, NA, 3), y = c(NA, NA, 4, 5), z = rep(NA, 4))
x %>% delete_na(2, 0.5)
```

```r
data.table(x = c(1, 2, NA, 3), y = c(NA, NA, 4, 5), z = rep(NA, 4))
x %>% delete_na(2, 0.24)
```

```r
data.table(x = c(1, 2, NA, 3), y = c(NA, NA, 4, 5), z = rep(NA, 4))
x %>% delete_na(2, 2)
```

```r
data.table(x = c(1, 2, NA, 3), y = c(NA, NA, 4, 5), z = rep(NA, 4))
x %>% delete_na(1, 0.6)
```

```r
data.table(x = c(1, 2, NA, 3), y = c(NA, NA, 4, 5), z = rep(NA, 4))
x %>% delete_na(1, 2)
```

---

**dummy**  
*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(.data, ..., longname = TRUE)

Arguments

- `.data` data.frame
- `...` Columns you want to create dummy variables from. Very flexible, find in the examples.
- `longname` logical. Should the output column labeled with the original column name? Default uses TRUE.
Details

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

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

Value
data.table

See Also
dummy_cols

Examples

iris = as.data.table(iris)
iris %>% dummy(Species)
iris %>% dummy(Species,longname = FALSE)

mtcars = as.data.table(mtcars)
mtcars %>% head() %>% dummy(vs,am)
mtcars %>% head() %>% dummy("cyl|gear")

export_fst

Read and write fst files

Description

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

Usage

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

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

- **x**: a data frame to write to disk
- **path**: path to fst file
- **compress**: value in the range 0 to 100, indicating the amount of compression to use. Lower values mean larger file sizes. The default compression is set to 50.
- **uniform_encoding**: If `TRUE`, all character vectors will be assumed to have elements with equal encoding. The encoding (latin1, UTF8 or native) of the first non-NA element will be used as encoding for the whole column. This will be a correct assumption for most use cases. If `uniform_encoding` is set to `FALSE`, no such assumption 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
export_fst(iris,"iris_fst_test.fst")
iris_dt = import_fst("iris_fst_test.fst")
iris_dt
unlink("iris_fst_test.fst")
```
**fill**  
*Fill in missing values with previous or next value*

**Description**

Fills missing values in selected columns using the next or previous entry.

**Usage**

```r
fill(.data, ..., direction = "down")
shift_fill(x, direction = "down")
```

**Arguments**

- `.data` A data.table
- `...` A selection of columns.
- `direction` Direction in which to fill missing values. Currently either "down" (the default), "up".
- `x` A vector.

**Details**

fill is filling data.table’s columns, shift_fill is filling any vectors.

**Value**

A filled data.table

**Examples**

```r
df <- data.table(Month = 1:12, Year = c(2000, rep(NA, 10), 2001))
df
df %>% fill(Year)

df <- data.table(Month = 1:12, Year = c(2000, rep(NA, 10), 2001))
df %>% fill(Year, direction = "up")
```
Description

Analogous function for filter in dplyr.

Usage

filter(.data, ...)

Arguments

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

Details

Currently data.table is not able to delete rows by reference.

Value

A data.table

References

https://github.com/Rdatatable/data.table/issues/635

See Also

filter

Examples

iris = as.data.table(iris)
iris %>% filter(Sepal.Length > 7)
iris %>% filter(Sepal.Length > 7, Sepal.Width > 3)
iris %>% filter(Sepal.Length > 7 & Sepal.Width > 3)
iris %>% filter(Sepal.Length == max(Sepal.Length))
**Parse, inspect and extract data.table from fst file**

**Description**

An API for reading fst file as data.table.

**Usage**

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

**Arguments**

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

**Details**

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

**Value**

- `parse_fst` returns a `fst_table` class.
- `select_fst` and `filter_fst` returns a `data.table`.

**See Also**

- `fst`, `metadata_fst`

**Examples**

```r
# write the file first
path = tempfile(fileext = "fst")
fst::write_fst(iris, path)
# parse the file but not reading it
parse_fst(path) -> ft
```
ft

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

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

ft %>% select_fst(Sepal.Length)
ft %>% select_fst(Sepal.Length,Sepal.Width)
ft %>% select_fst("Sepal.Length")
ft %>% select_fst(1:3)
ft %>% select_fst(1,3)
ft %>% select_fst("Se")

# return a warning with message
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)

---

**group_by**  
*Group by one or more variables*

**Description**
Most data operations are done on groups defined by variables. `group_by` will group the data.table by selected variables (setting them as keys), and arrange them in ascending order. `group_exe` could do computations by group, it receives an object returned by `group_by`.

**Usage**
```
group_by(.data, ...)
group_exe(.data, ...)
groups(x)
ungroup(x)
```
**inner_join**

**Arguments**

- `.data` A data.table
- `...` For `group_by`: Variables to group by. For `group_exe`: Any data manipulation arguments that could be implemented on a data.table.
- `x` A data.table

**Details**

For `mutate` and `summarise`, it is recommended to use the innate "by" parameter, which is faster. Once the data.table is grouped, the order is changed forever.

- `groups()` could return a character vector of specified groups.
- `ungroup()` would delete the keys in data.table.

**Value**

A data.table with keys

**Examples**

```r
a = as.data.table(iris)
a
a %>%
group_by(Species) %>%
group_exe(
  head(3)
) groups(a)
ungroup(a)
groups(a)
```

---

### inner_join  Join tables

**Description**

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

- 'inner_join()': includes all rows in 'x' and 'y'. * 'left_join()': includes all rows in 'x'. * 'right_join()': includes all rows in 'y'. * 'full_join()': 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()' return all rows from 'x' with a match in 'y'. * 'anti_join()' return all rows from 'x' without a match in 'y'.

Usage

inner_join(x, y, by = NULL, on = NULL)
left_join(x, y, by = NULL, on = NULL)
right_join(x, y, by = NULL, on = NULL)
full_join(x, y, by = NULL, on = NULL)
anti_join(x, y, by = NULL, on = NULL)
semi_join(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()' will perform a natural join, using all variables in common across 'x' and 'y'. A message lists the variables so that you can check they're correct; suppress the message by supplying 'by' explicitly.
To join by different variables on 'x' and 'y', use a named vector. For example, 'by = c("a" = "b")' will match 'x$a' to 'y$b'.
To join by multiple variables, use a vector with length > 1. For example, 'by = c("a", "b")' will match 'x$a' to 'y$a' and 'x$b' to 'y$b'. Use a named vector to match different variables in 'x' and 'y'. For example, 'by = c("a" = "b", "c" = "d")' will match 'x$a' to 'y$b' and 'x$c' to 'y$d'.
on (Optional) Indicate which columns in x should be joined with which columns in y. Examples included: 1. by = c("a", "b") (this is a must for set_full_join);
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).

Value

A data.table

Examples

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

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

workers %>% inner_join(positions)
workers %>% left_join(positions)
workers %>% right_join(positions)
workers %>% full_join(positions)

# filtering joins
workers %>% anti_join(positions)
workers %>% semi_join(positions)

# To suppress the message, supply 'by' argument
workers %>% left_join(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(positions2, by = c("name" = "worker"))

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

---

**lead**

*Fast lead/lag for vectors*

**Description**

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

**Usage**

```r
lead(x, n = 1L, fill = NA)
lag(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(1:5)
lag(1:5)
lead(1:5, 2)
lead(1:5, n = 2, fill = 0)

longer(.data, ..., name = "name", value = "value", na.rm = FALSE)
wider(.data, ..., name, value = NULL, fun = NULL, fill = NA)

Arguments

.data A data.table
... Columns for unchanged group. Flexible, see examples.
name Name for the measured variable names column.
value Name for the data values column(s).
na.rm If TRUE, NA values will be removed from the molten data.
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.

Value

A data.table

See Also

longer_dt, wider_dt
### Examples

```r
stocks <- data.table(
  time = as.Date('2009-01-01') + 0:9,
  X = rnorm(10, 0, 1),
  Y = rnorm(10, 0, 2),
  Z = rnorm(10, 0, 4)
)
stocks %>% longer(time)
stocks %>% longer(-(2:4)) # same
stocks %>% longer(~"X|Y|Z") # same
long_stocks = longer(stocks,"ti") # same as above except for assignment
long_stocks %>% wider(time, name = "name", value = "value")
# the unchanged group could be missed if all the rest will be used
long_stocks %>% wider(name = "name", value = "value")
```

### mat_df

**Conversion between tidy table and named matrix**

**Description**

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

**Usage**

```r
mat_df(m)
df_mat(df, row, col, value)
```

**Arguments**

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

**Value**

For `mat_df`, a data.frame. For `df_mat`, a named matrix.
Examples

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

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

**mutate**  
Create or transform variables

**Description**

`mutate()` adds new variables and preserves existing ones; `transmute()` adds new variables and drops existing ones. Both functions preserve the number of rows of the input. New variables overwrite existing variables of the same name.

`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.

If you mutate a data.table, it is forever changed. No copies made, which is efficient, but should be used with caution. If you still want the keep the original data.table, use `copy` first.

**Usage**

```r
mutate(.data, ..., by)
transmute(.data, ..., by)
mutable_when(.data, when, ..., by)
mutable_vars(.data, .cols = NULL, .func, ..., by)
```

**Arguments**

- `.data` A data.table
- `...` Name-value pairs of expressions
- `by` (Optional) Mutate by what group?
- `when` An object which can be coerced to logical mode
- `.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

A data.table

Examples

# Newly created variables are available immediately
a = as.data.table(mtcars)
copy(a) %>% mutate(cyl2 = cyl * 2)
a

# change forever
a %>% mutate(cyl2 = cyl * 2)
a

# You can also use mutate() to remove variables and
# modify existing variables
a %>% mutate(
  mpg = NULL,
  disp = disp * 0.0163871 # convert to litres
)
a %>% transmute(cyl, one = 1)
a

iris[3:8,] %>%
as.data.table() %>%
mutate_when(Petal.Width == .2,
  one = 1, Sepal.Length = 2)

iris[3:8,] %>%
as.data.table() %>%
mutate_vars("Pe", scale)

---

nest **Nest and unnest**

Description

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

Usage

`nest(.data, ..., mcols = NULL)`

`unnest(.data, ...)`
squeeze(.data, ...)

chop(.data, ...)

unchop(.data, ...)

**Arguments**

- **.data**: data.table, nested or unnested
- **...**: The variables for nest group (for `nest`), columns to be nested (for `squeeze` and `chop`), or column(s) to be unnested (for `unnest`). 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`, the data would be nested to a column named `ndt`, which is short for nested data.table.

The `squeeze` would not remove the original columns.

The `unchop` is the reverse operation of `chop`.

These functions are experiencing the experimental stage, especially the `unnest`. If they don’t work on some circumstances, try `tidyr` package.

**Value**

data.table, nested or unnested

**References**


**See Also**

- `nest`, `chop`

**Examples**

```r
mtcars = as.data.table(mtcars)
iris = as.data.table(iris)

# examples for nest

# nest by which columns?
mtcars %>% nest(cyl)
mtcars %>% nest("cyl")
```
mtcars %>% nest(cyl, vs)
mtcars %>% nest(vs:am)
mtcars %>% nest("cyl|vs")
mtcars %>% nest(c("cyl","vs"))

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

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

# examples for unnest
# unnest which column?
mtcars %>% nest("cyl|vs") %>%
    unnest(ndt)
mtcars %>% nest("cyl|vs") %>%
    unnest("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(a)
df %>% unnest(2)
df %>% unnest("c")
df %>% unnest(cols = names(df)[3])

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

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

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

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"
pull

Examples

iris %>% object_size()

pull

Pull out a single variable

Description

Analogous function for pull in dplyr

Usage

pull(.data, col)

Arguments

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

Value

A vector

See Also

pull

Examples

mtcars %>% pull(2)
mtcars %>% pull(cyl)
mtcars %>% pull("cyl")
**read_csv**  
*Convenient file reader*

**Description**
A wrapper of *fread* in *data.table*. Highlighting the encoding.

**Usage**

```r
read_csv(path, utf8 = FALSE, ...)
```

**Arguments**
- `path`  
  File name in working directory, path to file.
- `utf8`  
  Should "UTF-8" used as the encoding? (Defaults to FALSE)
- `...`  
  Other parameters passed to `data.table::fread`.

**Value**
A data.table

**relocate**  
*Change column order*

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

**Usage**

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

**Arguments**
- `.data`  
  A data.table
- `...`  
  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.

**Details**
Once you relocate the columns, the order changes forever.
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(f)
df %>% relocate(a, how = "last")

df %>% relocate(is.character)
df %>% relocate(is.numeric, how = "last")
df %>% relocate("[aeiou]"),

df %>% relocate(a, how = "after", where = f)
df %>% relocate(f, how = "before", where = a)
df %>% relocate(f, how = "before", where = c)
df %>% relocate(f, how = "after", where = c)

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

replace_vars

Fast value replacement in data frame

Description

replace_vars could replace any value(s) or values that match specific patterns to another specific value in a data.table.

Usage

replace_vars(.data, ..., from = is.na, to)

Arguments

.data A data.table
... Columns to be replaced. If not specified, use all columns.
from A value, a vector of values or a function returns a logical value. Defaults to NaN.
to A value.

Value

A data.table.
See Also

replace_dt

Examples

```r
iris %>% as.data.table() %>%
  mutate(Species = as.character(Species)) -> new_iris

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

Description

Compute on a data frame a row-at-a-time. This is most useful when a vectorised function doesn’t exist. Only mutate and summarise are supported so far.

Usage

```r
rowwise_mutate(.data, ...)
rowwise_summarise(.data, ...)
```

Arguments

- `.data` A data.table
- `...` Name-value pairs of expressions

Value

A data.table

See Also

rowwise
Examples

# without rowwise
df <- data.table(x = 1:2, y = 3:4, z = 4:5)
df %>% mutate(m = mean(c(x, y, z)))

# with rowwise
df <- data.table(x = 1:2, y = 3:4, z = 4:5)
df %>% rowwise_mutate(m = mean(c(x, y, z)))

# # rowwise is also useful when doing simulations
params = fread(" sim n mean sd
1 1 1 1
2 2 2 4
3 3 -1 2")

params %>%
  rowwise_summarise(sim, z = rnorm(n, mean, sd))

---

select

Select/rename variables by name

Description

Choose or rename variables from a data.table. select() keeps only the variables you mention; rename() keeps all variables.

Usage

select(.data, ...)

select_vars(.data, ..., rm.dup = TRUE)

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

select_mix(.data, ..., rm.dup = TRUE)

rename(.data, ...)

Arguments

.data

A data.table

... One or more unquoted expressions separated by commas. Very flexible, same as tidyfst::select_dt and tidyfst::select_mix. details find select_dt.

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

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.
Details

No copy is made. Once you select or rename a data.table, they would be changed forever. `select_vars` could select across different data types, names and index. See examples.

`select_dt` and `select_mix` is the safe mode of `select` and `select_vars`, they keep the original copy but are not memory-efficient when dealing with large data sets.

Value

A data.table

See Also

`select_dt`, `rename_dt`

Examples

```r
a = as.data.table(iris)
a %>% select(1:3)
a

a = as.data.table(iris)
a %>% select_vars(is.factor,"Se")
a

a = as.data.table(iris)
a %>% select("Se") %>%
  rename(sl = Sepal.Length,
         sw = Sepal.Width)
a

DT = data.table(a=1:2,b=3:4,c=5:6)
DT
DT %>% rename(B=b)
```

---

**separate**

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

Description

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

Usage

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

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.

Value

A data.table

See Also

separate, unite_dt

Examples

df <- data.table(x = c(NA, "a.b", "a.d", "b.c"))
df %>% separate(x, c("A", "B"))  
# equals to
df <- data.table(x = c(NA, "a.b", "a.d", "b.c"))
df %>% separate("x", c("A", "B"))

slice Subset rows using their positions

Description

’slice()’ lets you index rows by their (integer) locations. It allows you to select, remove, and duplicate rows. It is accompanied by a number of helpers for common use cases:

* ‘slice_head()’ and ‘slice_tail()’ select the first or last rows.  
* ‘slice_sample()’ randomly selects rows.  
* ‘slice_min()’ and ‘slice_max()’ select rows with highest or lowest values of a variable.

Usage

slice(.data, ...)
slice_head(.data, n)
slice_tail(.data, n)
slice_max(.data, order_by, n, with_ties = TRUE)
slice_min(.data, order_by, n, with_ties = TRUE)
slice_sample(.data, n, replace = FALSE)
summarise

Arguments

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

Value

A data.table

See Also

slice

Examples

```r
a = as.data.table(iris)
slice(a,1,2)
slice(a,2:3)
slice_head(a,5)
slice_head(a,0.1)
slice_tail(a,5)
slice_tail(a,0.1)
slice_max(a,Sepal.Length,10)
slice_max(a,Sepal.Length,10,with_ties = FALSE)
slice_min(a,Sepal.Length,10)
slice_min(a,Sepal.Length,10,with_ties = FALSE)
slice_sample(a,10)
slice_sample(a,0.1)
```

summarise

Summarise columns to single values

Description

Create one or more scalar variables summarizing the variables of an existing data.table.
Usage

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

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

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

Arguments

.data A data.table

... List of variables or name-value pairs of summary/modifications functions for summarise_dt. Additional parameters to be passed to parameter '.func' in summarise_vars.

by Unquoted name of grouping variable of list of unquoted names of grouping variables. For details see data.table

when An object which can be coerced to logical mode

.cols Columns to be summarised.

.func Function to be run within each column, should return a value or vectors with same length.

Value

A data.table

Examples

a = as.data.table(iris)
a %>% summarise(sum = sum(Sepal.Length), avg = mean(Sepal.Length))

a %>%
  summarise_when(Sepal.Length > 5, avg = mean(Sepal.Length), by = Species)

a %>%
  summarise_vars(is.numeric, min, by = Species)

Description

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

sys_time_print(expr)

Arguments

expr

Valid R expression to be timed.

Value

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

See Also

timetaken, system.time

Examples

sys_time_print(Sys.sleep(1))
a = as.data.table(iris)
sys_time_print({
  res = a %>%
    mutate(one = 1)
})
res

uncount
"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(.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.

Value

A data.table
unite

See Also
count, uncount

Examples

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

unite

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(.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.

Value

A data.table

See Also

unite, separate
Examples

```r
df <- CJ(x = c("a", NA), y = c("b", NA))
df

# Treat missing value as NA, default
df %>% unite("z", x:y, remove = FALSE)
# Treat missing value as character "NA"
df %>% unite("z", x:y, na2char = TRUE, remove = FALSE)
# the unite has memory, "z" would not be removed in new operations
# here we remove the original columns ("x" and "y")
df %>% unite("xy", x:y, remove = TRUE)

# Select all columns
iris %>% as.data.table %>% unite("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**

```r
utf8_encoding(.data, .cols)
```

**Arguments**

- `.data` : A `data.frame`.
- `.cols` : The columns you want to convert, usually a character column.

**Value**

A `data.table` with characters in UTF-8 encoding

**Examples**

```r
iris %>%
  as.data.table() %>%
  utf8_encoding(Species) # could also use `is.factor`
```
Index

add_count (count), 5
anti_join (inner_join), 15
arrange, 2, 3
as_fst, 3

chop, 22
drop_na (drop_na), 7
dummy, 8
dummy_cols, 9

export_fst, 9
fill, 11
group_by, 14
group_exe (group_by), 14

import_fst (export_fst), 9

lag (lead), 17
lead, 17, 18
left_join (inner_join), 15
longer, 18
longer_dt, 18

mat_df, 19
metadata_fst, 13
mutate, 20
mutate_vars (mutate), 20
mutate_when (mutate), 20

nest, 21, 22
nth, 23

object_size, 24

parse_fst (fst), 13
pull, 25, 25

read_csv, 26
read_fst, 9, 10

rename (select), 29
rename_dt, 30
replace_dt, 28
replace_vars, 27

right_join (inner_join), 15
rowwise, 28
rowwise_mutate, 28
rowwise_summarise (rowwise_mutate), 28

select, 29
select_dt, 4, 20, 22, 29, 30

select_fst (fst), 13
groups (group_by), 14

semi_join (inner_join), 15

setorder, 3
shift, 18
shift_fill (fill), 11
slice, 31, 32
slice_fst (fst), 13
slice_head (slice), 31
slice_max (slice), 31
slice_min(slice), 31
slice_sample(slice), 31
slice_tail(slice), 31
squeeze(nest), 21
summarise, 32
summarise_vars(summarise), 32
summarise_when(summarise), 32
summary_fst(fst), 13
sys_time_print, 33
system.time, 34

timetaken, 34
transmute(mutate), 20

unchop(nest), 21
uncount, 34, 35
ungroup(group_by), 14
unite, 35, 35
unite_dt, 31
unnest(nest), 21
utf8_encoding, 36

wider(longer), 18
wider_dt, 18
write_fst, 9