# Package 'baizer'

May 8, 2023

**Title** Useful Functions for Data Processing

**Version** 0.5.0

**Description**
In ancient Chinese mythology, Bai Ze is a divine creature that knows the needs of everything. 'baizer' provides data processing functions frequently used by the author. Hope this package also knows what you want!

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.2.3

**Imports** curl, dplyr (>= 1.1.0), grDevices, magrittr, openxlsx, purrr, rlang (>= 0.4.11), stats, stringr, tibble (>= 3.1), tidyr, utils

**Suggests** covr, roxygen2, testthat (>= 3.0.0), withr

**Config/testthat/edition** 3

**Depends** R (>= 3.5.0)

**LazyData** true

**URL** https://github.com/william-swl/baizer

**BugReports** https://github.com/william-swl/baizer/issues

**NeedsCompilation** no

**Author** William Song [aut, cre]

**Maintainer** William Song <william_swl@163.com>

**Repository** CRAN

**Date/Publication** 2023-05-08 13:20:02 UTC

## R topics documented:

---

`adjacent_div` *expand a number vector according to the adjacent two numbers*

---

## Description

expand a number vector according to the adjacent two numbers

## Usage

```
adjacent_div(v, n_div = 10, .unique = FALSE)
```

## Arguments

| | |
|---|---|
| `v` | number vector |
| `n_div` | how many divisions expanded by two numbers |
| `.unique` | only keep unique numbers |

## Value

new number vector

## Examples

```
adjacent_div(10^c(1:3), n_div = 10)
```

## alias_arg *use aliases for function arguments*

### Description

use aliases for function arguments

### Usage

```
alias_arg(..., default = NULL)
```

### Arguments

| | |
|---|---|
| ... | aliases of an argument |
| default | a alias with a default value |

### Value

the finally value of this argument across all aliases

### Examples

```
# set y, z as aliases of x when create a function
func <- function(x = 1, y = NULL, z = NULL) {
  x <- alias_arg(x, y, z, default = x)
  return(x)
}
```

## as_md_table *trans a tibble into markdown format table*

### Description

trans a tibble into markdown format table

### Usage

```
as_md_table(x, show = TRUE)
```

### Arguments

| | |
|---|---|
| x | tibble |
| show | show result instead of return the markdown string, TRUE as default |

## Value

NULL or markdown string

## Examples

```
mini_diamond %>%
  head(5) %>%
  as_md_table()
```

---

`as_tibble_md` *trans a table in markdown format into tibble*

---

## Description

trans a table in markdown format into tibble

## Usage

```
as_tibble_md(x)
```

## Arguments

| | |
|---|---|
| `x` | character string |

## Value

tibble

## Examples

```
x <- "
col1 | col2 | col3 |
| ---- | ---- | ---- |
| v1   | v2   | v3   |
| r1   | r2   | r3   |
"

as_tibble_md(x)
```

## atomic_expr *whether the expression is an atomic one*

### Description

whether the expression is an atomic one

### Usage

```
atomic_expr(ex)
```

### Arguments

| | |
|---|---|
| ex | expression |

### Value

logical value

### Examples

```
atomic_expr(rlang::expr(x))

atomic_expr(rlang::expr(!x))

atomic_expr(rlang::expr(x + y))

atomic_expr(rlang::expr(x > 1))

atomic_expr(rlang::expr(!x + y))

atomic_expr(rlang::expr(x > 1 | y < 2))
```

## c2r *wrapper of tibble::column_to_rownames*

### Description

wrapper of tibble::column_to_rownames

### Usage

```
c2r(df, col = "")
```

### Arguments

| | |
|---|---|
| df | tibble |
| col | a col name |

### Value

data.frame

### Examples

```
mini_diamond %>% c2r("id")
```

---

## cmdargs *get the command line arguments*

---

### Description

get the command line arguments

### Usage

```
cmdargs(x = NULL)
```

### Arguments

| | |
|---|---|
| x | one of 'wd, R_env, script_path, script_dir, env_configs' |

### Value

list of all arguments, or single value of select argument

### Examples

```
cmdargs()
```

## collapse_vector *dump a named vector into character*

### Description

dump a named vector into character

### Usage

```
collapse_vector(named_vector, front_name = TRUE, collapse = ",")
```

### Arguments

| | |
|---|---|
| named_vector | a named vector |
| front_name | if TRUE, put names to former |
| collapse | collapse separator |

### Value

character

### Examples

```
collapse_vector(c(e = 1:4), front_name = TRUE, collapse = ";")
```

## correct_ratio *correct the numbers to a target ratio*

### Description

correct the numbers to a target ratio

### Usage

```
correct_ratio(raw, target, digits = 0)
```

### Arguments

| | |
|---|---|
| raw | the raw numbers |
| target | the target ratio |
| digits | the result digits |

### Value

corrected number vector

## Examples

```
correct_ratio(c(10, 10), c(3, 5))

# support ratio as a float
correct_ratio(c(100, 100), c(0.2, 0.8))

# more numbers
correct_ratio(10:13, c(2, 3, 4, 6))

# with digits after decimal point
correct_ratio(c(10, 10), c(1, 4), digits = 1)
```

---

`cross_count` *count two columns as a cross-tabulation table*

---

## Description

count two columns as a cross-tabulation table

## Usage

```
cross_count(df, row, col, method = "n", digits = 2)
```

## Arguments

| | |
|---|---|
| `df` | tibble |
| `row` | the column as rownames in the output |
| `col` | the column as colnames in the output |
| `method` | one of `n\|count, rowr\|row_ratio, colr\|col_ratio` |
| `digits` | the digits of ratios |

## Value

data.frame

## Examples

```
cross_count(mini_diamond, cut, clarity)

# show the ratio in the row
cross_count(mini_diamond, cut, clarity, method = "rowr")

# show the ratio in the col
cross_count(mini_diamond, cut, clarity, method = "colr")
```

## detect_dup *detect possible duplication in a vector, ignore case, blank and special character*

### Description

detect possible duplication in a vector, ignore case, blank and special character

### Usage

```
detect_dup(vector, index = FALSE)
```

### Arguments

| | |
|---|---|
| vector | vector possibly with duplication |
| index | return duplication index |

### Value

duplication sub-vector

### Examples

```
detect_dup(c("a", "C_", "c -", "#A"))
```

## diff_index *the index of different character*

### Description

the index of different character

### Usage

```
diff_index(s1, s2, nth = NULL, ignore_case = FALSE)
```

### Arguments

| | |
|---|---|
| s1 | string1 |
| s2 | string2 |
| nth | just return nth index |
| ignore_case | ignore upper or lower cases |

### Value

list of different character indices

## Examples

```
diff_index("AAAA", "ABBA")
```

---

`empty_dir` *detect whether directory is empty recursively*

---

## Description

detect whether directory is empty recursively

## Usage

```
empty_dir(dir)
```

## Arguments

| | |
|---|---|
| `dir` | the directory |

## Value

logical value

## Examples

```
# create an empty directory
dir.create("some/deep/path/in/a/folder", recursive = TRUE)
empty_dir("some/deep/path/in/a/folder")

# create an empty file
file.create("some/deep/path/in/a/folder/there_is_a_file.txt")
empty_dir("some/deep/path/in/a/folder")
empty_file("some/deep/path/in/a/folder/there_is_a_file.txt", strict = TRUE)

# create a file with only character of length 0
write("", "some/deep/path/in/a/folder/there_is_a_file.txt")
empty_file("some/deep/path/in/a/folder/there_is_a_file.txt", strict = TRUE)
empty_file("some/deep/path/in/a/folder/there_is_a_file.txt")

# clean
unlink("some", recursive = TRUE)
```

empty_file *detect whether file is empty recursively*

## Description

detect whether file is empty recursively

## Usage

```
empty_file(path, strict = FALSE)
```

## Arguments

| | |
|---|---|
| path | the path of file |
| strict | FALSE as default. If TRUE, a file with only one character of length 0 will be considered as not empty |

## Value

logical value

## Examples

```
# create an empty directory
dir.create("some/deep/path/in/a/folder", recursive = TRUE)
empty_dir("some/deep/path/in/a/folder")

# create an empty file
file.create("some/deep/path/in/a/folder/there_is_a_file.txt")
empty_dir("some/deep/path/in/a/folder")
empty_file("some/deep/path/in/a/folder/there_is_a_file.txt", strict = TRUE)

# create a file with only character of length 0
write("", "some/deep/path/in/a/folder/there_is_a_file.txt")
empty_file("some/deep/path/in/a/folder/there_is_a_file.txt", strict = TRUE)
empty_file("some/deep/path/in/a/folder/there_is_a_file.txt")

# clean
unlink("some", recursive = TRUE)
```

## expr_pileup *pileup the subexpressions which is atomic*

### Description

pileup the subexpressions which is atomic

### Usage

```
expr_pileup(ex)
```

### Arguments

| | |
|---|---|
| ex | expression |

### Value

the character vector of subexpressions

### Examples

```
ex <- rlang::expr(a == 2 & b == 3 | !b & x + 2)
expr_pileup(ex)
```

## extract_kv *extract key and values for a character vector*

### Description

extract key and values for a character vector

### Usage

```
extract_kv(v, sep = ": ", key_loc = 1, value_loc = 2)
```

### Arguments

| | |
|---|---|
| v | character vector |
| sep | separator between key and value |
| key_loc | key location |
| value_loc | value location |

### Value

a named character vector

## Examples

```
extract_kv(c("x: 1", "y: 2"))
```

---

`fancy_count` *fancy count to show an extended column*

---

## Description

fancy count to show an extended column

## Usage

```
fancy_count(df, ..., ext = NULL, ext_fmt = "count", sort = FALSE, digits = 2)
```

## Arguments

| | |
|---|---|
| `df` | tibble |
| `...` | other arguments from `dplyr::count()` |
| `ext` | extended column |
| `ext_fmt` | `count|ratio|clean`, output format of extended column |
| `sort` | sort by frequency or not |
| `digits` | if `ext_fmt=ratio`, the digits of ratio |

## Value

count tibble

## Examples

```
fancy_count(mini_diamond, cut, ext = clarity)

fancy_count(mini_diamond, cut, ext = clarity, ext_fmt = "ratio")

fancy_count(mini_diamond, cut, ext = clarity, ext_fmt = "clean")

fancy_count(mini_diamond, cut, ext = clarity, sort = FALSE)

fancy_count(mini_diamond, cut, clarity, ext = id) %>% head(5)
```

## fetch_char *fetch character from strings*

### Description

fetch character from strings

### Usage

```
fetch_char(s, index_list, na.rm = FALSE, collapse = FALSE)
```

### Arguments

| | |
|---|---|
| s | strings |
| index_list | index of nth character, can be output of diff_index or same_index |
| na.rm | remove NA values from results or not |
| collapse | optional string used to combine the characters from a same string |

### Value

list of characters

### Examples

```
fetch_char(rep("ABC", 3), list(1, 2, 3))
```

## filterC *apply tbflt on dplyr filter*

### Description

apply tbflt on dplyr filter

### Usage

```
filterC(.data, tbflt = NULL, .by = NULL, usecol = TRUE)
```

### Arguments

| | |
|---|---|
| .data | tibble |
| tbflt | tbflt object |
| .by | group by, same as .by argument in dplyr::filter |
| usecol | if TRUE (default), use the default behavior of dplyr::filter(), which allows the usage of same variable in colnames, and filter by the data column. If FALSE, will check whether the variables on the right side of ==,>,<,>=,<= have same names as columns and raise error, for the sake of more predictable results. You can always ignore this argument if you know how to use .env or !! |

## Value

tibble

## Examples

```
c1 <- tbflt(cut == "Fair")

c2 <- tbflt(x > 8)

mini_diamond %>%
  filterC(c1) %>%
  head(5)

mini_diamond %>% filterC(c1 & c2)



x <- 8
cond <- tbflt(y > x)

# variable `x` not used because of column `x` in `mini_diamond`
filterC(mini_diamond, cond)

# will raise error because `x`  is on the right side of `>`
# filterC(mini_diamond, cond, usecol=FALSE)

# if you know how to use `.env` or `!!`, forget argument `usecol`!
cond <- tbflt(y > !!x)
filterC(mini_diamond, cond)

cond <- tbflt(y > .env$x)
filterC(mini_diamond, cond)
```

---

# fix_to_regex *trans fixed string into regular expression string*

---

## Description

trans fixed string into regular expression string

## Usage

```
fix_to_regex(p)
```

## Arguments

| | |
|---|---|
| p | raw fixed pattern |

### Value

regex pattern

### Examples

```
fix_to_regex("ABC|?(*)")
```

---

## float_to_percent *from float number to percent number*

---

### Description

from float number to percent number

### Usage

```
float_to_percent(x, digits = 2)
```

### Arguments

| | |
|---|---|
| x | number |
| digits | hold n digits after the decimal point |

### Value

percent character of x

### Examples

```
float_to_percent(0.12)
```

---

## fps_vector *farthest point sampling (FPS) for a vector*

---

### Description

farthest point sampling (FPS) for a vector

### Usage

```
fps_vector(v, n, method = "round")
```

### Arguments

| | |
|---|---|
| v | vector |
| n | sample size |
| method | round\|floor\|ceiling, the method used when trans to integer |

### Value

sampled vector

### Examples

```
fps_vector(1:10, 4)
```

---

## generate_ticks *generate ticks for a number vector*

---

### Description

generate ticks for a number vector

### Usage

```
generate_ticks(x, expect_ticks = 10)
```

### Arguments

| | |
|---|---|
| x | number vector |
| expect_ticks | expected number of ticks, may be a little different from the result |

### Value

ticks number

### Examples

```
generate_ticks(c(176, 198, 264))
```

---

## gen_outlier *generate outliers from a series of number*

---

### Description

generate outliers from a series of number

### Usage

```
gen_outlier(
  x,
  n,
  digits = 0,
  side = "both",
  lim = NULL,
  assign_n = NULL,
  only_out = TRUE
)
```

### Arguments

| | |
|---|---|
| x | number vector |
| n | number of outliers to generate |
| digits | the digits of outliers |
| side | should be one of both, low, high |
| lim | a two-length vector to assign the limitations of the outliers if method is both, the outliers will be limited in [lim[1], low_outlier_threshold] and [high_outlier_threshold, lim[2]] ; if method is low, the outliers will be limited in [lim[1], min(low_outlier_threshold, lim[2])] ; if method is high, the outliers will be limited in [max(high_outlier_threshold, lim[1]), lim[2]] |
| assign_n | manually assign the number of low outliers or high outliers when method is both |
| only_out | only return outliers |

### Value

number vector of outliers

### Examples

```
x <- seq(0, 100, 1)

gen_outlier(x, 10)

# generation limits
gen_outlier(x, 10, lim = c(-80, 160))

# assign the low and high outliers
gen_outlier(x, 10, lim = c(-80, 160), assign_n = c(0.1, 0.9))

# just generate low outliers
gen_outlier(x, 10, side = "low")

# return with raw vector
gen_outlier(x, 10, only_out = FALSE)
```

---

## geom_mean *geometric mean*

---

### Description

geometric mean

### Usage

```
geom_mean(x, na.rm = TRUE)
```

### Arguments

| | |
|---|---|
| x | value |
| na.rm | remove NA or not |

### Value

geometric mean value

### Examples

```
geom_mean(1, 9)
```

---

## group_vector *group chracter vector by a regex pattern*

---

### Description

group chracter vector by a regex pattern

### Usage

```
group_vector(x, pattern = "\\w")
```

### Arguments

| | |
|---|---|
| x | character vector |
| pattern | regex pattern, '\w' as default |

### Value

list

### Examples

```
v <- c(
  stringr::str_c("A", c(1, 2, 9, 10, 11, 12, 99, 101, 102)),
  stringr::str_c("B", c(1, 2, 9, 10, 21, 32, 99, 101, 102))
) %>% sample()

group_vector(v)

group_vector(v, pattern = "\\w\\d")

group_vector(v, pattern = "\\w(\\d)")

# unmatched part will alse be stored
group_vector(v, pattern = "\\d{2}")
```

## hist_bins *separate numeric x into bins*

### Description

separate numeric x into bins

### Usage

```
hist_bins(x, bins = 10, lim = c(min(x), max(x)), breaks = NULL, sort = FALSE)
```

### Arguments

| | |
|---|---|
| x | numeric vector |
| bins | bins number, defaults to 10 |
| lim | the min and max limits of bins, default as c(min(x), max(x)) |
| breaks | assign breaks directly and will ignore bins and lim |
| sort | sort the result tibble |

### Value

tibble

### Examples

```
x <- dplyr::pull(mini_diamond, price, id)

hist_bins(x, bins = 20)
```

## int_digits *trans numbers to a fixed integer digit length*

### Description

trans numbers to a fixed integer digit length

### Usage

```
int_digits(x, digits = 2, scale_factor = FALSE)
```

### Arguments

| | |
|---|---|
| x | number |
| digits | integer digit length |
| scale_factor | return the scale_factor instead of value |

### Value

number

### Examples

```
int_digits(0.0332, 1)
```

---

## is.zero *if a number only have zeros*

---

### Description

if a number only have zeros

### Usage

```
is.zero(x)
```

### Arguments

| | |
|---|---|
| x | number |

### Value

all zero or not

### Examples

```
is.zero(c("0.000", "0.102", NA))
```

## mini_diamond *Minimal tibble dataset adjusted from diamond*

### Description

Minimal tibble dataset adjusted from diamond

### Usage

```
mini_diamond
```

### Format

`mini_diamond`**:**
A data frame with 100 rows and 7 columns:

**id** unique id
**cut, clarity** 2 category variables
**carat, price, x, y** 4 continuous variables ...

### Source

adjusted from `ggplot2`

## move_row *move selected rows to target location*

### Description

move selected rows to target location

### Usage

```
move_row(df, rows, .after = FALSE, .before = FALSE)
```

### Arguments

| | |
|---|---|
| `df` | tibble |
| `rows` | selected rows indexes |
| `.after` | `TRUE` will move selected rows to the last row, or you can pass a target row index |
| `.before` | `TRUE` will move selected rows to the first row, or you can pass a target row index |

### Value

reordered tibble

## Examples

```
move_row(mini_diamond, 3:5, .after = 8)
```

---

`nearest_tick` *the nearest ticks around a number*

---

## Description

the nearest ticks around a number

## Usage

```
nearest_tick(x, side = "both", level = NULL, div = 2)
```

## Arguments

| | |
|---|---|
| `x` | number |
| `side` | default as 'both', can be 'both\|left\|right' |
| `level` | the level of ticks, such as 1, 10, 100, etc. |
| `div` | number of divisions |

## Value

nearest tick number

## Examples

```
nearest_tick(3462, level = 10)
```

---

`near_ticks` *the ticks near a number*

---

## Description

the ticks near a number

## Usage

```
near_ticks(x, level = NULL, div = 2)
```

## Arguments

| | |
|---|---|
| `x` | number |
| `level` | the level of ticks, such as 1, 10, 100, etc. |
| `div` | number of divisions |

### Value

number vector of ticks

### Examples

```
near_ticks(3462, level = 10)
```

---

## number_fun_wrapper *wrapper of the functions to process number string with prefix and suffix*

---

### Description

wrapper of the functions to process number string with prefix and suffix

### Usage

```
number_fun_wrapper(
  x,
  fun = ~.x,
  prefix_ext = NULL,
  suffix_ext = NULL,
  verbose = FALSE
)
```

### Arguments

| | |
|---|---|
| x | number string vector with prefix and suffix |
| fun | process function |
| prefix_ext | prefix extension |
| suffix_ext | suffix extension |
| verbose | print more details |

### Value

processed number with prefix and suffix

### Examples

```
number_fun_wrapper(">=2.134%", function(x) round(x, 2))
```

## ordered_slice *slice a tibble by an ordered vector*

### Description

slice a tibble by an ordered vector

### Usage

```
ordered_slice(df, by, ordered_vector, na.rm = FALSE, dup.rm = FALSE)
```

### Arguments

| | |
|---|---|
| df | tibble |
| by | slice by this column, this value must has no duplicated value |
| ordered_vector | ordered vector |
| na.rm | remove NA or unknown values from ordered vector |
| dup.rm | remove duplication values from ordered vector |

### Value

sliced tibble

### Examples

```
ordered_slice(mini_diamond, id, c("id-3", "id-2"))
```

## percent_to_float *from percent number to float number*

### Description

from percent number to float number

### Usage

```
percent_to_float(x, digits = 2, to_double = FALSE)
```

### Arguments

| | |
|---|---|
| x | percent number character |
| digits | hold n digits after the decimal point |
| to_double | use double output |

### Value

float character or double of x

### Examples

```
percent_to_float("12%")
```

---

## pileup_logical *pileup another logical vector on the TRUE values of first vector*

---

### Description

pileup another logical vector on the TRUE values of first vector

### Usage

```
pileup_logical(x, v)
```

### Arguments

| | |
|---|---|
| x | logical vector |
| v | another logical vector |

### Value

logical vector

### Examples

```
# first vector have 2 TRUE value
v1 <- c(TRUE, FALSE, TRUE)

# the length of second vector should also be 2
v2 <- c(FALSE, TRUE)

pileup_logical(v1, v2)
```

## pkginfo *information of packages*

### Description

information of packages

### Usage

```
pkginfo(...)
```

### Arguments

| | |
|---|---|
| ... | case-insensitive package names |

### Examples

```
baizer::pkginfo(dplyr)
```

## pkglib *load packages as a batch*

### Description

load packages as a batch

### Usage

```
pkglib(...)
```

### Arguments

| | |
|---|---|
| ... | pkgs |

### Examples

```
baizer::pkglib(dplyr, purrr)
```

## pkgver *versions of packages*

### Description

versions of packages

### Usage

```
pkgver(...)
```

### Arguments

| | |
|---|---|
| ... | case-insensitive package names |

### Examples

```
baizer::pkgver(dplyr, purrr)
```

## pos_int_split *split a positive integer number as a number vector*

### Description

split a positive integer number as a number vector

### Usage

```
pos_int_split(x, n, method = "average")
```

### Arguments

| | |
|---|---|
| x | positive integer |
| n | length of the output |
| method | should be one of `average`, `random`, or a number vector which length is n |

### Value

number vector

### Examples

```
pos_int_split(12, 3, method = "average")

pos_int_split(12, 3, method = "random")

pos_int_split(12, 3, method = c(1, 2, 3))
```

## r2c *wrapper of tibble::rownames_to_column*

### Description

wrapper of tibble::rownames_to_column

### Usage

```
r2c(df, col = "")
```

### Arguments

| | |
|---|---|
| df | tibble |
| col | a col name |

### Value

tibble

### Examples

```
mini_diamond %>%
  c2r("id") %>%
  r2c("id")
```

## ref_level *relevel a target column by another reference column*

### Description

relevel a target column by another reference column

### Usage

```
ref_level(x, col, ref)
```

### Arguments

| | |
|---|---|
| x | tibble |
| col | target column |
| ref | reference column |

### Value

tibble

## Examples

```
cut_level <- mini_diamond %>%
  dplyr::pull(cut) %>%
  unique()

mini_diamond %>%
  dplyr::mutate(cut = factor(cut, cut_level)) %>%
  dplyr::mutate(cut0 = stringr::str_c(cut, "xxx")) %>%
  ref_level(cut0, cut)
```

---

`reg_join` *join the matched parts into string*

---

## Description

join the matched parts into string

## Usage

```
reg_join(x, pattern, sep = "")
```

## Arguments

| | |
|---|---|
| `x` | character |
| `pattern` | regex pattern |
| `sep` | separator |

## Value

character

## Examples

```
reg_join(c("A_12.B", "C_3.23:2"), "[A-Za-z]+")

reg_join(c("A_12.B", "C_3.23:2"), "\\w+")

reg_join(c("A_12.B", "C_3.23:2"), "\\d+", sep = ",")

reg_join(c("A_12.B", "C_3.23:2"), "\\d", sep = ",")
```

## reg_match *regex match*

### Description

regex match

### Usage

```
reg_match(x, pattern, group = 1)
```

### Arguments

| | |
|---|---|
| x | vector |
| pattern | regex pattern |
| group | regex gruop, 1 as default. when group=-1, return full matched tibble |

### Value

vector or tibble

### Examples

```
v <- stringr::str_c("id", 1:3, c("A", "B", "C"))

reg_match(v, "id(\\d+)(\\w)")

reg_match(v, "id(\\d+)(\\w)", group = 2)

reg_match(v, "id(\\d+)(\\w)", group = -1)
```

## remove_nacol *remove columns by the ratio of NA*

### Description

remove columns by the ratio of NA

### Usage

```
remove_nacol(df, max_ratio = 1)
```

### Arguments

| | |
|---|---|
| df | tibble |
| max_ratio | max NA ratio, default as 1 which remove the columns only have NA |

### Value

tibble

### Examples

```
# remove_nacol(df)
```

---

## remove_narow *remove rows by the ratio of NA*

---

### Description

remove rows by the ratio of NA

### Usage

```
remove_narow(df, max_ratio = 1)
```

### Arguments

| | |
|---|---|
| df | tibble |
| max_ratio | max NA ratio, default as 1 which remove the rows only have NA |

### Value

tibble

### Examples

```
# remove_narow(df)
```

---

## replace_item *replace the items of one object by another*

---

### Description

replace the items of one object by another

### Usage

```
replace_item(x, y, keep_extra = FALSE)
```

### Arguments

| | |
|---|---|
| x | number, character or list |
| y | another object, the class of y should be same as x |
| keep_extra | whether keep extra items in y |

## Value

replaced object

## Examples

```
x <- list(A = 1, B = 3)
y <- list(A = 9, C = 10)

replace_item(x, y)

replace_item(x, y, keep_extra = TRUE)
```

---

# round_string *from float number to fixed digits character*

---

## Description

from float number to fixed digits character

## Usage

```
round_string(x, digits = 2)
```

## Arguments

| | |
|---|---|
| x | number |
| digits | hold n digits after the decimal point |

## Value

character

## Examples

```
round_string(1.1, 2)
```

## same_index *the index of identical character*

### Description

the index of identical character

### Usage

```
same_index(s1, s2, nth = NULL, ignore_case = FALSE)
```

### Arguments

| | |
|---|---|
| s1 | string1 |
| s2 | string2 |
| nth | just return nth index |
| ignore_case | ignore upper or lower cases |

### Value

list of identical character indices

### Examples

```
same_index("AAAA", "ABBA")
```

## sftp_connect *connection parameters to remote server via sftp*

### Description

connection parameters to remote server via sftp

### Usage

```
sftp_connect(
  server = "localhost",
  port = 22,
  user = NULL,
  password = NULL,
  wd = "~"
)
```

### Arguments

| | |
|---|---|
| server | remote server |
| port | SSH port, 22 as default |
| user | username |
| password | password |
| wd | workdir |

### Value

sftp_connection object

### Examples

```
# sftp_con <- sftp_connect(server='remote_host', port=22,
#     user='username', password = "password", wd='~')
```

---

## sftp_download *download file from remote server via sftp*

---

### Description

download file from remote server via sftp

### Usage

```
sftp_download(sftp_con, path = NULL, to = basename(path))
```

### Arguments

| | |
|---|---|
| sftp_con | sftp_connection created by sftp_connect() |
| path | remote file path |
| to | local target path |

### Examples

```
# sftp_download(sftp_con,
#   path=c('t1.txt', 't2.txt'),
#   to=c('path1.txt', 'path2.txt')
```

## signif_ceiling *signif while use ceiling*

### Description

signif while use ceiling

### Usage

```
signif_ceiling(x, digits = 2)
```

### Arguments

| | |
|---|---|
| x | number |
| digits | digits |

### Value

number

### Examples

```
signif_ceiling(3.11, 2)
```

## signif_floor *signif while use floor*

### Description

signif while use floor

### Usage

```
signif_floor(x, digits = 2)
```

### Arguments

| | |
|---|---|
| x | number |
| digits | digits |

### Value

number

### Examples

```
signif_floor(3.19, 2)
```

## signif_round_string *signif or round string depend on the character length*

### Description

signif or round string depend on the character length

### Usage

```
signif_round_string(
  x,
  digits = 2,
  format = "short",
  full_large = TRUE,
  full_small = FALSE
)
```

### Arguments

| | |
|---|---|
| x | number |
| digits | signif or round digits |
| format | short or long |
| full_large | keep full digits for large number |
| full_small | keep full digits for small number |

### Value

signif or round strings

### Examples

```
signif_round_string(1.214, 2)
```

## signif_string *from float number to fixed significant digits character*

### Description

from float number to fixed significant digits character

### Usage

```
signif_string(x, digits = 2)
```

### Arguments

| | |
|---|---|
| x | number |
| digits | hold n significant digits |

### Value

character

### Examples

```
signif_string(1.1, 2)
```

---

## sortf *sort by a function*

---

### Description

sort by a function

### Usage

```
sortf(x, func, group_pattern = NULL)
```

### Arguments

| | |
|---|---|
| x | vector |
| func | a function used by the sort |
| group_pattern | a regex pattern to group by, only aviable if x is a character vector |

### Value

vector

### Examples

```
sortf(c(-2, 1, 3), abs)

v <- stringr::str_c("id", c(1, 2, 9, 10, 11, 12, 99, 101, 102)) %>% sample()

sortf(v, function(x) reg_match(x, "\\d+") %>% as.double())

sortf(v, ~ reg_match(.x, "\\d+") %>% as.double())

v <- c(
  stringr::str_c("A", c(1, 2, 9, 10, 11, 12, 99, 101, 102)),
  stringr::str_c("B", c(1, 2, 9, 10, 21, 32, 99, 101, 102))
) %>% sample()

sortf(v, ~ reg_match(.x, "\\d+") %>% as.double(), group_pattern = "\\w")
```

## split_column *split a column and return a longer tibble*

### Description

split a column and return a longer tibble

### Usage

```
split_column(df, name_col, value_col, sep = ",")
```

### Arguments

| | |
|---|---|
| df | tibble |
| name_col | repeat this as name column |
| value_col | expand by this value column |
| sep | separator in the string |

### Value

expanded tibble

### Examples

```
fancy_count(mini_diamond, cut, ext = clarity) %>%
  split_column(name_col = cut, value_col = clarity)
```

## split_path *split a path into ancestor paths recursively*

### Description

split a path into ancestor paths recursively

### Usage

```
split_path(path)
```

### Arguments

| | |
|---|---|
| path | path to split |

### Value

character vectors of ancestor paths

### Examples

```
split_path("/home/someone/a/test/path.txt")
```

---

## split_vector *split vector into list*

---

### Description

split vector into list

### Usage

```
split_vector(vector, breaks, bounds = "(]")
```

### Arguments

| | |
|---|---|
| vector | vector |
| breaks | split breaks |
| bounds | "(]" as default, can also be "[), []" |

### Value

list

### Examples

```
split_vector(1:10, c(3, 7))
split_vector(stringr::str_split("ABCDEFGHIJ", "") %>% unlist(),
  c(3, 7),
  bounds = "[)"
)
```

---

## stat_fc *fold change calculation which returns a extensible tibble*

---

### Description

fold change calculation which returns a extensible tibble

## Usage

```
stat_fc(
  df,
  y,
  x,
  method = "mean",
  .by = NULL,
  rev_div = FALSE,
  digits = 2,
  fc_fmt = "short"
)
```

## Arguments

| | |
|---|---|
| df | tibble |
| y | value |
| x | sample test group |
| method | 'mean'\|'median'\|'geom_mean', the summary method |
| .by | super-group |
| rev_div | reverse division |
| digits | fold change digits |
| fc_fmt | fold change format, one of short, signif, round |

## Value

fold change result tibble

## Examples

```
stat_fc(mini_diamond, y = price, x = cut, .by = clarity)
```

---

# stat_test *statistical test which returns a extensible tibble*

---

## Description

statistical test which returns a extensible tibble

## Usage

```
stat_test(
  df,
  y,
  x,
  .by = NULL,
  trans = "identity",
  paired = FALSE,
  alternative = "two.sided",
  method = "wilcoxon",
  ns_symbol = "NS"
)
```

## Arguments

| | |
|---|---|
| df | tibble |
| y | value |
| x | sample test group |
| .by | super-group |
| trans | scale transformation |
| paired | paired samples or not |
| alternative | one of "two.sided" (default), "greater" or "less" |
| method | test method, 'wilcoxon' as default |
| ns_symbol | symbol of nonsiginficant, 'NS' as default |

## Value

test result tibble

## Examples

```
stat_test(mini_diamond, y = price, x = cut, .by = clarity)
```

---

# tbflt *create a tbflt object to save filter conditions*

---

## Description

`tbflt()` can save a series of filter conditions, and support logical operating among conditions

## Usage

```
tbflt(x = expression(), .env = NULL)
```

### Arguments

| | |
|---|---|
| x | any expression |
| .env | environment |

### Value

tbflt

### Examples

```
c1 <- tbflt(cut == "Fair")

c2 <- tbflt(x > 8)

!c1

c1 | c2

c1 & c2
```

---

## uniq *only keep unique vector values and its names*

---

### Description

only keep unique vector values and its names

### Usage

```
uniq(x)
```

### Arguments

| | |
|---|---|
| x | vector |

### Value

vector

### Examples

```
x <- c(a = 1, b = 2, c = 3, b = 2, a = 1)

uniq(x)
```

## write_excel *write a tibble into an excel file*

### Description

write a tibble into an excel file

### Usage

```
write_excel(df, filename, sheetname = NULL, creator = "")
```

### Arguments

| | |
|---|---|
| df | tibble or a list of tibbles |
| filename | the output filename |
| sheetname | the names of sheets. If not given, will use 'sheet1', or the names of list |
| creator | creator |

### Value

return status

### Examples

```
# write_excel(mini_diamond, "mini_diamond.xlsx")
```

## %neq% *not equal calculation operator, support NA*

### Description

not equal calculation operator, support NA

### Usage

```
x %neq% y
```

### Arguments

| | |
|---|---|
| x | value x |
| y | value y |

### Value

logical value, TRUE if x and y are not equal

## Examples

```
1 %neq% NA
```

---

`%nin%` *not in calculation operator*

---

## Description

not in calculation operator

## Usage

```
left %nin% right
```

## Arguments

| | |
|---|---|
| `left` | left element |
| `right` | right element |

## Value

logical value, TRUE if left is not in right

## Examples

```
0 %nin% 1:4
```

# Index