Package ‘do’

April 1, 2020

Type Package
Title Data Operator
Version 1.2.0.0
Description Flexibly convert data between long and wide format using just two functions: reshape_toLong() and reshape_toWide().
Author Jing Zhang, Zhi Jin
Maintainer Jing Zhang<zj391120@163.com>
License GPL-3
Encoding UTF-8
LazyData true
Imports data.table, plyr, tmcn
RoxygenNote 6.1.1
URL https://github.com/yikeshu0611/do
BugReports https://github.com/yikeshu0611/do/issues
NeedsCompilation no
Repository CRAN
Date/Publication 2020-04-01 14:40:02 UTC

R topics documented:

  chinese_utf8 ......................................................... 2
  col_split ......................................................... 3
  common .............................................................. 3
  delete_left ....................................................... 4
  delete_up ........................................................... 5
  dup.connect ....................................................... 5
  equal_length ...................................................... 6
  expand ............................................................ 7
  get_names .......................................................... 7
  Grepl .............................................................. 8
### Description
UTF8 Code for Chinese

### Usage
```r
chinese_utf8(x)
```

### Arguments
- **x**: chinese characters

### Value
an expression with UTF8 code.
**col_split**

*Split A Vector into Columns*

**Description**

Split A Vector into Columns

**Usage**

\[
\text{col\_split}(x, \text{split, reg\_expr, colnames})
\]

**Arguments**

- **x**: a vector
- **split**: one or more characters. Split exactly
- **reg\_expr**: character. Split by regular expressions
- **colnames**: optional. Column names for outcome

**Value**

A dataframe with several columns.

**Examples**

```r
x=c('1a2','3a4','4a4')
col_split(x,split='a')
col_split(x=x,reg\_expr = '^[a-z]$')

#two splits
df=data.frame(result=c('A, B-C',
'A, C-D',
'E, F-G'))
col_split(x = df[,1],split = c(',','-'))
```

**common**

*Find Common Objects from Vectors*

**Description**

Find Common Objects from Vectors

**Usage**

\[
\text{common}(\ldots)
\]
Arguments

... must be several vectors

Value

common objects

Examples

x1=c('a','e','d')
x2=c('a','c','e')
x3=c('a','e','j','d')
common(x1,x2,x3)

delete_left(x=df,NA)

---

delete_left  Delete and Move Left the rest Values

Description

Delete and Move Left the rest Values

Usage

delete_left(x, delete)

Arguments

x  dataframe or matrix

delete  one delete object

Value

dataframe or matrix

Examples

a=c(1,NA,7,NA)
b=c(NA,2,2,7)
d=c(1,NA,40,7)
df=data.frame(a,b,d)
delete_left(x=df,NA)
**delete_up**

*Delete and Move Up the Rest Values*

**Description**
Delete and Move Up the Rest Values

**Usage**
```
delete_up(x, delete)
```

**Arguments**
- `x` : dataframe or matrix
- `delete` : one delete object

**Value**
- dataframe or matrix

**Examples**
```
a=c(1,NA,7,NA)
b=c(NA,2,2,7)
d=c(1,NA,40,7)
df=data.frame(a,b,d)

delete_up(x = df, delete = NA)
```

**dup.connect**

*Connect Duplicated Values*

**Description**
Connect Duplicated Values

**Usage**
```
dup.connect(data, id, dup.var)
```

**Arguments**
- `data` : dataframe or matrix
- `id` : id column names or indexes
- `dup.var` : duplicated column names or indexes
Value
dataframe contains id and duplicated values

Examples
dup.connect(data = mtcars,id = 'am',dup.var = 'cyl')
dup.connect(data = mtcars,
  id = c('am','gear'),
  dup.var = c('cyl','qsec'))

equal_length  Equal Length

Description
Equal Length

Usage
equal_length(x, suffix = " ", nchar, colname = FALSE, rowname = FALSE)

Arguments
  x  can be number, strings, vectors, dataframe or matrix.
  suffix  suffix
  nchar  maximum length
  colname  a logistic value, default is FALSE
  rowname  a logistic value, default is FALSE

Value
equal length results

Examples
a=c(123,1,24,5,1.22554)
equal_length(a,0)
df = data.frame(
  a=c(12,1,1.23),
  b=c('a','abcd','d'))
equal_length(x = df,suffix = 'x')
equal_length(x = df,suffix = 0,nchar =5)
**expand**

*Expand Data by Weight*

**Description**

Expand Data by Weight

**Usage**

expand(x, weight)

**Arguments**

- `x`: dataframe or matrix
- `weight`: weight column names or index

**Value**

expanded data

**Examples**

```r
df = data.frame(v=c(1,2,3),
                x=c(7,8,9),
                n=c(2,3,4))
expand(x = df, weight = 3)
expand(x = df, weight = 'n')
```

---

**get_names**

*Get Names of Object*

**Description**

Return the names of input. For example: if you input a, you will get 'a'.

**Usage**

get_names(...) 

**Arguments**

- `...`: any type of data object

**Value**

names of object
Examples

```r
a <- c(1,2,3)
grepl(a, mtcars)
```

Description
Judge for Included Character

Usage

```r
grepl(pattern, x)
```

Arguments

- `pattern` one or more vectors
- `x` one or more vectors

Details

`

Value

a matrix with logical words

Examples

```r
a <- c("abcd", "agj", "abcu")

# Grepl for one vector
pat1 <- "b"
grepl(pat1, a)

# Grepl for two vectors
pat2 <- c("c", "d")
grepl(pat2, a)

# use %or% in pattern
pat3 <- c("a%or%c", "d")
grepl(pat3, a)

# use %and% in pattern
pat4 <- c("a%and%c", "d")
grepl(pat4, a)
```
**inner_Add_Symbol**  
*Concatenate Strings*

---

**Description**

Concatenate vectors by adding a symbol.

**Usage**

```r
inner_Add_Symbol(x, symbol = "+")
```

**Arguments**

- `x`: vectors
- `symbol`: default is `+`

**Value**

a concatenated string

**Examples**

```r
inner_Add_Symbol(c("Var a", "Var b"))
inner_Add_Symbol(c("Var a", "Var b"), ")
inner_Add_Symbol(c("Var a", "Var b"), "")
```

---

**left**  
*Truncate Characters from the Left*

---

**Description**

Truncate Characters from the Left

**Usage**

```r
left(x, n)
```

**Arguments**

- `x`: can be number, strings, vectors, dataframe or matrix.
- `n`: length

**Value**

substring
Examples

left("abcd", 3)
left(c("abc", "gjh"), 2)
df = data.frame(
    a = c(123, 234, 456),
    b = c("abc", "bcd", "hjg")
)
left(df, 2)

mid
Truncate Characters from the Inside

Description

Truncate Characters from the Inside

Usage

mid(x, start, n)

Arguments

x can be number, strings, vectors, dataframe or matrix.
start starting position
n length, n can be less than zero

Value

substring

Examples

mid("abcd", 3, 1)
mid(c("abc", "gjh"), 2, 2)
df = data.frame(
    a = c(123, 234, 456),
    b = c("abc", "bcd", "hjg")
)
mid(df, 2, 1)
mid(df, 2, -2)
### Nchar

**Number of Characters**

#### Description
Number of Characters

#### Usage
Nchar(x)

#### Arguments
- **x**
  can be number, strings, vectors, dataframe or matrix.

#### Value
number of characters in each location

#### Examples
- Nchar("abcd")
- Nchar(c("abc","gjh"))
- df = data.frame(
  a = c(1,12,12.3),
  b = c("a","ab","abc")
)
- Nchar(df)

### Replace

**Replace**

#### Description
There are two methods in this function. You can use replace many objects to one by form and to.

#### Usage
Replace(data, from, to, pattern)

#### Arguments
- **data**
  can be number, strings, vectors, dataframe or matrix.
- **from**
  replaced strings
- **to**
  replacements
- **pattern**
  like from:to
Value

replaced data

Examples

Replace(data = 232, from = 2, to = 1)
Replace(data = c(232, 'a4b'),
        from = c(2, '\*4'), to = 1,
        pattern = c('a:e', 'b:h'))
df = data.frame(
    a = c(232, 452),
    b = c("nba", "cba")
)
Replace(data = df,
        from = 2, to = 1,
        pattern = c('a:e', 'b:h'))

Replace0

Replaced by Empty

Description

Replaced by Empty

Usage

Replace0(data, from)

Arguments

data can be number, strings, vectors, dataframe or matrix.
from replaced strings

Value

replaced data

Examples

Replace0(data = 232, from = 2)
Replace0(data = c(232, 'a4b'), from = c(2, '\*4'))

df = data.frame(
    a = c(232, 452),
    b = c("nba", "cba")
)
Replace0(data = df, from = c(2, 'a'))
Replace_ex  

Replace Exactly

Description
Replace Exactly

Usage

Replace_ex(x, from, to, pattern)

Arguments

x vector, dataframe or matrix
from replaced stings
to replacements
pattern a special pattern, see examples for detail

Value
replaced data

Examples

a=c(1,2,3,1,4)
Replace_ex(x = a, from = c(1,2), to=5)
Replace_ex(x=a,pattern = c('1:5','2:5'))
Replace_ex(x=a,pattern = '[12]:5')

a=data.frame(v=c(1,2,3,2,4),
b=c(7,8,9,4,6))
Replace_ex(x = a, from = c(1,2), to=5)
Replace_ex(x=a,pattern = c('1:5','2:5'))
Replace_ex(x=a,pattern = '[12]:5')

reshape_toLong  

Convert Wide Data to Long

Description
It is easy to convert wide data to long in this function. Be careful, id must be unique. prefix, suffix and var.names can be used together.
reshape_toLong

Usage

reshape_toLong(data, prefix, suffix, var.names)

Arguments

data   wide data
prefix prefix of value variables
suffix suffix of value variables
var.names names of value variables, do.value will be created as the name of value column

Value

long data

Examples

df = data.frame(
id = c(101,102,103,104),
w1 = c(1,2,3,4),
w2 = c(6,7,8,9),
h1 = c(5,6,7,8),
names = c('s1','s2','s3','s4')
)
reshape_toLong(data = df,prefix = c('w','h'))

# using suffix
df = data.frame(
id = c(102,103,104,105),
t1w = c(1,2,3,4),
t2w = c(6,7,8,9),
t1h = c(5,6,7,8),
t2h = c(1,3,5,7),
sex=c('female','male','male','female')
)
reshape_toLong(data = df,suffix = c('w','h'))

# using prefix and suffix together
df = data.frame(
id = c(102,103,104,105),
wt1 = c(1,2,3,4),
wt2 = c(6,7,8,9),
t1h = c(5,6,7,8),
t2h = c(1,3,5,7),
sex=c('female','male','male','female')
)
reshape_toLong(data = df,prefix = 'w',suffix = 'h')

# using var.names
df = data.frame(
```r
id = c(102,103,104,105),
w = c(1,2,3,4),
h = c(1,3,5,7),
sex=c('female','male','male','female')
)  
reshape_toLong(data = df,var.names = c('w','h'))
```

---

**reshape_towide**

Reshape to Wide Format

**Description**

Reshape to Wide Format

**Usage**

```r
reshape_towide(data, id, col_change, prefix, suffix, sep = "_")
```

**Arguments**

- **data**: long data
- **id**: column names for id, which can be one or more
- **col_change**: column names for exchange, which can be one or more
- **prefix**: column names for prefix, which can be one or more
- **suffix**: column names for suffix, which can be one or more
- **sep**: separation

**Value**

A wide data.

**Examples**

```r
df = data.frame(id = c(1,1,2,2,3,3,4,4),
    time = c(1,2,1,2,1,2,1,2),
    w = c(1,6,2,7,3,8,4,9))

reshape_towide(data = df,
    id = 'id',
    col_change = 'time',
    prefix = 'w')

df = data.frame(id = c(1,1,2,2,3,3,4,4),
    time = c(1,2,1,2,1,2,1,2),
    w = c(1,6,2,7,3,8,4,9),
    h = c(5,1,6,3,7,5,8,7),
    n = c(2,2,3,3,4,4,5,5))

reshape_towide(data = df,
    id = 'id',
    col_change = 'time',
    prefix = 'w',
    suffix = 'h')
```

reverse

Reverse String Order

Usage

reverse(x)

Arguments

x can be number, strings, vectors, dataframe or matrix.

Value

reversed string

Examples

reverse(123)
reverse(c(123, 'abc'))

df = data.frame(
  a = c(123, 456),
  b = c("abc", "def")
)
reverse(df)
right

Truncate Characters from the Right

Description
Truncate Characters from the Right

Usage
right(x, n)

Arguments
x can be number, strings, vectors, dataframe or matrix.
n length

Value
substring

Examples
right("abcd",3)
right(c("abc","gjh"),2)
df = data.frame(
a = c(123,234,456),
b = c("abc","bcd","hjg")
)
right(df,2)

row.freq

Row Frequency

Description
Row Frequency

Usage
row.freq(x)

Arguments
x dataframe or matrix
split_expanded

**Value**

data with frequency column

**Examples**

```r
row.freq(x=mtcars[,8:11])
```

---

**split_expanded**  
*Split One Column and Expand*

**Description**

Split One Column and Expand

**Usage**

```r
split_expanded(data, variable, sep)
```

**Arguments**

- `data` : dataframe or matrix
- `variable` : one column name with connected values
- `sep` : seperated symbol, which can be one or more

**Value**

expanded dataframe or matrix

**Examples**

```r
df=data.frame(a=c(1,0),
b=c('a','n'),
cyl=c('6;6;4;4;4','6;8;'))
split_expanded(data=df,variable='cyl',sep=';')
```
take_out

Description
Extract Some String

Usage
take_out(x, ..., type = "c")

Arguments
- x: string
- ...: patterns of c('begin','after')
- type: any left characters of character or list

Value
characters

Examples
x='abdghtyu'
take_out(x,c('a','d'),c('h','u'))

Trim

Description
Trim

Usage
trim(x, pattern = " ")
trim_left(x, pattern = " ")
trim_right(x, pattern = " ")

Arguments
- x: can be vector or dataframe or matrix
- pattern: one or more pattern pattern
Value

a trimmed string

---

<table>
<thead>
<tr>
<th>unique_no.NA</th>
<th>Unique Without NA</th>
</tr>
</thead>
</table>

Description

Unique Without NA

Usage

unique_no.NA(x)

Arguments

x vector

Value

unique values with no NA

Examples

x=c(1,2,3,1,NA)
unique(x)
unique_no.NA(x)

---

%==%  Locate Accurately

Description

Locate Accurately

Usage

a %==% b

Arguments

a vector for matching
b vector for searching
Value
If length of a is one, a vector will be return. If length of a is more than one, a list for each element will be return.

Examples
```
a=c(1,2,3,4)
b=c(1,2,3,4,1,5,6,1,4,1)
a %==% b
```

%==% Locate Similarly by grep()

Description
Locate Similarly by grep()

Usage
```
a %==% b
```

Arguments
```
a vector for matching
b vector for searching
```

Value
A list contains location information.

Examples
```
1 %==% c(1,12,3)
c(1,2) %==% c(1,12,3)
```
Index

\%\%=, 20
\%s=\%, 21

chinese_utf8, 2
col_split, 3
col_split, 3

delete_left, 4
delete_up, 5
dup.connect, 5
equal_length, 6
expand, 7
get_names, 7

Grepl, 8
inner_Add_Symbol, 9

left, 9

mid, 10

Nchar, 11

Replace, 11
Replace0, 12
Replace_ex, 13
resdfape_toLong, 13
reshape_towide, 15
reverse, 16
right, 17
row.freq, 17

split_expand, 18

take_out, 19

Trim, 19
trim-left (Trim), 19
trim-right (Trim), 19
Trim_left(Trim), 19
Trim_right(Trim), 19

unique_no.NA, 20