Package ‘tidyr’

February 20, 2015

Title Easily Tidy Data with spread() and gather() Functions.

Version 0.2.0

Description An evolution of reshape2. It's designed specifically for data tidying (not general reshaping or aggregating) and works well with dplyr data pipelines.

Depends R (>= 3.1.0)

License MIT + file LICENSE

LazyData true

Imports reshape2, dplyr (>= 0.2), stringi, lazyeval

URL `https://github.com/hadley/tidyr`

BugReports `https://github.com/hadley/tidyr/issues`

Suggests knitr, testthat

VignetteBuilder knitr

Author Hadley Wickham [aut, cre],
RStudio [cph]

Maintainer Hadley Wickham `<hadley@rstudio.com>`

NeedsCompilation no

Repository CRAN

Date/Publication 2014-12-05 18:03:26

R topics documented:

`expand` *Expand data frame to include all combinations of levels.*

Description

Expand data frame to include all combinations of levels.

Usage

```
expand(data, ...)
```

Arguments

`data`	A data frame
`...`	Specification of columns to expand. These can either be bare column names, or transformations of a column.

See Also

`expand_` for a version that uses regular evaluation and is suitable for programming with.

Examples

```
expand(mtcars, vs, cyl)
expand(mtcars, cyl, mpg = seq_range(mpg, 2))
expand(mtcars, cyl, mpg = seq_range(mpg, 5))

df <- data.frame(a = c(1, 2, 5), b = c(3, 5, 3), c = c(1, 2, 3))
expand(df)
expand(df, a, b)
expand(df, a, c)
expand(df, b, c)
```

extract — *Extract one column into multiple columns.*

Description

Given a regular expression with capturing groups, `extract()` turns each group into a new column.

Usage

```
extract(data, col, into, regex = "([[:alnum:]]+)", remove = TRUE,
  convert = FALSE, ...)
```

Arguments

`data`	A data frame.
`col`	Bare column name.
`into`	Names of new variables to create as character vector.
`regex`	a regular expression used to extract the desired values.
`remove`	If `TRUE`, remove input column from output data frame.
`convert`	If `TRUE`, will run `type.convert` with `as.is = TRUE` on new columns. This is useful if the component columns are integer, numeric or logical.
`...`	Other arguments passed on to `regexec` to control how the regular expression is processed.

Examples

```
library(dplyr)
df <- data.frame(x = c("a.b", "a.d", "b.c"))
df %>% extract(x, "A")
df %>% extract(x, c("A", "B"), "([[:alnum:]]+)\\.([[:alnum:]]+)")
```

extract_numeric — *Extract numeric component of variable.*

Description

This uses a regular expression to strip all non-numeric character from a string and then coerces the result to a number. This is useful for strings that are numbers with extra formatting (e.g. $1,200.34).

Usage

```
extract_numeric(x)
```

Arguments

`x`	A character vector (or a factor).

Examples

```
extract_numeric("$1,200.34")
extract_numeric("-2%")

# The heuristic is not perfect - it won't fail for things that
# clearly aren't numbers
extract_numeric("-2-2")
extract_numeric("12abc34")
```

`gather` *Gather columns into key-value pairs.*

Description

Gather takes multiple columns and collapses into key-value pairs, duplicating all other columns as needed. You use `gather()` when you notice that you have columns that are not variables.

Usage

```
gather(data, key, value, ..., na.rm = FALSE, convert = FALSE)
```

Arguments

`data`	A data frame.
`key,value`	Names of key and value columns to create in output.
`...`	Specification of columns to gather. Use bare variable names. Select all variables between x and z with `x:z`, exclude y with `-y`. For more options, see the select documentation.
`na.rm`	If `TRUE`, will remove rows from output where the value column in `NA`.
`convert`	If `TRUE` will automatically run `type.convert` on the key column. This is useful if the column names are actually numeric, integer, or logical.

See Also

`gather_` for a version that uses regular evaluation and is suitable for programming with.

Examples

```
library(dplyr)
# From http://stackoverflow.com/questions/1181060
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)
)

gather(stocks, stock, price, -time)
stocks %>% gather(stock, price, -time)
```

`separate` *Separate one column into multiple columns.*

Description

Given either regular expression or a vector of character positions, `separate()` turns a single character column into multiple columns.

Usage

```
separate(data, col, into, sep = "[^[:alnum:]]+", remove = TRUE,
  convert = FALSE, extra = "error", ...)
```

Arguments

`data`	A data frame.
`col`	Bare column name.
`into`	Names of new variables to create as character vector.
`sep`	Separator between columns. If character, is interpreted as a regular expression. The default value is a regular expression that matches any sequence of non-alphanumeric values. If numeric, interpreted as positions to split at. Positive values start at 1 at the far-left of the string; negative value start at -1 at the far-right of the string. The length of `sep` should be one less than `into`.
`remove`	If `TRUE`, remove input column from output data frame.
`convert`	If `TRUE`, will run `type.convert` with `as.is = TRUE` on new columns. This is useful if the component columns are integer, numeric or logical.
`extra`	If `sep` is a character vector, this controls what happens when the number of pieces doesn't match `into`. There are three valid options: • "error" (the default): throws error if pieces aren't right length • "drop": always returns `length(into)` pieces by dropping or expanding as necessary

	• "merge": only splits at most `length(into)` times
`...`	Other arguments passed on to `strsplit` to control how the regular expression is processed.

Examples

```
library(dplyr)
df <- data.frame(x = c("a.b", "a.d", "b.c"))
df %>% separate(x, c("A", "B"))

# If every row doesn't split into the same number of pieces, use
# the extra argument to control what happens
df <- data.frame(x = c("a", "a b", "a b c", NA))
df %>% separate(x, c("a", "b"), extra = "merge")
df %>% separate(x, c("a", "b"), extra = "drop")

# If only want to split specified number of times use extra = "merge"
df <- data.frame(x = c("x: 123", "y: error: 7"))
df %>% separate(x, c("key", "value"), ": ", extra = "merge")
```

`seq_range` *Create an evenly spaced sequence of values from highest to lowest.*

Description

Create an evenly spaced sequence of values from highest to lowest.

Usage

```
seq_range(x, n)
```

Arguments

`x`	A numeric vector
`n`	Number of values

Examples

```
seq_range(1:100, 5)
```

spread *Spread a key-value pair across multiple columns.*

Description

Spread a key-value pair across multiple columns.

Usage

```
spread(data, key, value, fill = NA, convert = FALSE, drop = TRUE)
```

Arguments

`data`	A data frame.
`key,value`	Bare (unquoted) names of key and value columns.
`fill`	If there isn't a value for every combination of the other variables and the key column, this value will be substituted.
`convert`	If `TRUE`, `type.convert` with `asis = TRUE` will be run on each of the new columns. This is useful if the value column was a mix of variables that was coerced to a string.
`drop`	If `FALSE`, will keep factor levels that don't appear in the data, filling in missing combinations with `fill`.

Examples

```
library(dplyr)
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)
)
stocksm <- stocks %>% gather(stock, price, -time)
stocksm %>% spread(stock, price)
stocksm %>% spread(time, price)

# Spread and gather are complements
df <- data.frame(x = c("a", "b"), y = c(3, 4), z = c(5, 6))
df %>% spread(x, y) %>% gather(x, y, a:b, na.rm = TRUE)
```

spread_ *Standard-evaluation version of* `spread`.

Description

This is a S3 generic.

Usage

```
spread_(data, key_col, value_col, fill = NA, convert = FALSE, drop = TRUE)
```

Arguments

`data`	A data frame.
`key_col,value_col`	Strings giving names of key and value cols.
`fill`	If there isn't a value for every combination of the other variables and the key column, this value will be substituted.
`convert`	If `TRUE`, `type.convert` with `asis = TRUE` will be run on each of the new columns. This is useful if the value column was a mix of variables that was coerced to a string.
`drop`	If `FALSE`, will keep factor levels that don't appear in the data, filling in missing combinations with `fill`.

unite *Unite multiple columns into one.*

Description

Convenience function to paste together multiple functions into one.

Usage

```
unite(data, col, ..., sep = "_", remove = TRUE)
```

Arguments

`data`	A data frame.
`col`	(Bare) name of column to add
`...`	Specification of columns to unite. Use bare variable names. Select all variables between x and z with `x:z`, exclude y with `-y`. For more options, see the select documentation.
`sep`	Separator to use between values.
`remove`	If `TRUE`, remove input columns from output data frame.

See Also

separate(), the complement.

Examples

```
library(dplyr)
unite_(mtcars, "vs_am", c("vs","am"))

# Separate is the complement of unite
mtcars %>%
  unite(vs_am, vs, am) %>%
  separate(vs_am, c("vs", "am"))
```

---

unnest *Unnest a list column.*

Description

If you have a list-column, this makes each element of the list it's own row.

Usage

```
unnest(data, col = NULL)
```

Arguments

data	A data frame.
col	Name of column that needs to be unnested.

Examples

```
library(dplyr)
df <- data.frame(
  x = 1:3,
  y = c("a", "d,e,f", "g,h"),
  stringsAsFactors = FALSE
)
df %>%
  transform(y = strsplit(y, ",")) %>%
  unnest(y)

# You can also unnest lists
my_list <- lapply(split(subset(iris, select = -Species), iris$Species), "[", 1:2, )
unnest(my_list)
unnest(my_list, Species)
```

`unnest_` *Standard-evaluation version of* `unnest`.

Description

This is a S3 generic.

Usage

```
unnest_(data, col)
```

Arguments

`data`	A data frame.
`col`	Name of column that needs to be unnested.

Index