Package ‘splitstackshape’

August 29, 2016

Type Package
Title Stack and Reshape Datasets After Splitting Concatenated Values
Version 1.4.2
Date 2014-10-23
Author Ananda Mahto
Maintainer Ananda Mahto <ananda@mahto.info>
Description Online data collection tools like Google Forms often export multiple-response questions with data concatenated in cells. The concat.split (cSplit) family of functions splits such data into separate cells. The package also includes functions to stack groups of columns and to reshape wide data, even when the data are "unbalanced"—something which reshape (from base R) does not handle, and which melt and dcast from reshape2 do not easily handle.
License GPL-3
LazyData TRUE
LazyLoad yes
Depends R (>= 2.10), data.table (>= 1.9.4)
URL http://github.com/mrdwab/splitstackshape
BugReports http://github.com/mrdwab/splitstackshape/issues
NeedsCompilation no
Repository CRAN
Date/Publication 2014-10-23 17:31:07

R topics documented:
splitstackshape-package .................................................. 2
charMat ................................................................. 4
concat.split .......................................................... 5
concat.split.compact .................................................. 7
concat.split.expanded ................................................ 8
splitstackshape-package

Description

Stack and Reshape Datasets After Splitting Concatenated Values

Details

Package: splitstackshape
Type: Package
Version: 1.4.2
Date: 2014-10-23
License: GPL-3

Online data collection tools like Google Forms often export multiple-response questions with data concatenated in cells. The `concat.split` family of functions splits such data into separate cells. The package also includes functions to `stack` groups of columns and to `reshape` wide data, even when the data are "unbalanced"—something which `reshape` does not handle, and which `melt` and `dcast` from `reshape2` do not easily handle.
Author(s)
Ananda Mahto
Maintainer: Ananda Mahto <ananda@mahto.info>

Examples

## concat.split
```r
head(cSplit(concat.test, "Likes", drop = TRUE))
```

## Reshape
```r
set.seed(1)
mydf <- data.frame(id_1 = 1:6, id_2 = c("A", "B"),
    varA.1 = sample(letters, 6),
    varA.2 = sample(letters, 6),
    varA.3 = sample(letters, 6),
    varB.2 = sample(10, 6),
    varB.3 = sample(10, 6),
    varC.3 = rnorm(6))
```
```r
mydf
Reshape(mydf, id.vars = c("id_1", "id_2"),
    var.stubs = c("varA", "varB", "varC"))
```

## Stacked
```r
Stacked(data = mydf, id.vars = c("id_1", "id_2"),
    var.stubs = c("varA", "varB", "varC"),
    sep = ".")
```

## Not run:
## Processing times
```r
set.seed(1)
Nrow <- 1000000
Ncol <- 10
mybigdf <- cbind(id = 1:Nrow, as.data.frame(matrix(rnorm(Nrow*Ncol),
    nrow=Nrow)))
```
```r
head(mybigdf)
dim(mybigdf)
tail(mybigdf)
A <- names(mybigdf)
names(mybigdf) <- c("id", paste("var", 1:3, sep = "_"),
    paste("varB", 1:4, sep = "_"),
    paste("varC", 1:3, sep = "_"))
```
```r
system.time(
    O1 <- Reshape(mybigdf, id.vars = "id",
        var.stubs = c("varA", "varB", "varC"), sep = ".")
    O1 <- O1[order(O1$id, O1$time), ]
)
```
```r
system.time(
    O2 <- merged.stack(mybigdf, id.vars="id",
        var.stubs=c("varA", "varB", "varC"), sep = ".")
)
```
```r
system.time(
    O3 <- Stacked(mybigdf, id.vars="id",
        var.stubs=c("varA", "varB", "varC"), sep = ".")
)
```
charMat

Create a Binary Matrix from a List of Character Values

Description

Create a binary matrix from a list of character values

Usage

charMat(listOfValues, fill = NA, mode = "binary")

Arguments

listOfValues  A list of input values to be inserted in a matrix.
fill          The initializing fill value for the empty matrix.
mode          Either "binary" or "value". Defaults to "binary".

Details

This is primarily a helper function for the concat.split function when creating the "expanded" structure. The input is anticipated to be a list of values obtained using strsplit.

Value

A matrix.

Author(s)

Ananda Mahto

See Also

strsplit, numMat
**Examples**

```r
invec <- c("rock,electro","electro","rock,jazz")
A <- strsplit(invec, ",")
splitstackshape:::charMat(A)
splitstackshape:::charMat(A, @)
splitstackshape:::charMat(A, mode = "value")
```

---

**concat.split**  
*Split Concatenated Cells in a Dataset*

**Description**

The `concat.split` function takes a column with multiple values, splits the values into a list or into separate columns, and returns a new data.frame or data.table.

**Usage**

```r
concat.split(data, split.col, sep = ",", structure = "compact", mode = NULL, type = NULL, drop = FALSE, fixed = FALSE, fill = NA, ...)
```

**Arguments**

- `data`: The source data.frame or data.table.
- `split.col`: The variable that needs to be split; can be specified either by the column number or the variable name.
- `sep`: The character separating each value (defaults to ", ").
- `structure`: Can be either "compact", "expanded", or "list". Defaults to "compact". See Details.
- `mode`: Can be either "binary" or "value" (where "binary" is default and it recodes values to 1 or NA, like Boolean data, but without assuming 0 when data is not available). This setting only applies when `structure = "expanded"`; a warning message will be issued if used with other structures.
- `type`: Can be either "numeric" or "character" (where "numeric" is default). This setting only applies when `structure = "expanded"`; a warning message will be issued if used with other structures.
- `drop`: Logical (whether to remove the original variable from the output or not). Defaults to FALSE.
- `fixed`: Is the input for the `sep` value fixed, or a regular expression? See Details.
- `fill`: The "fill" value for missing values when `structure = "expanded"`. Defaults to NA.
- `...`: Additional arguments to `cSplit`. 
Details

structure

• "compact" creates as many columns as the maximum length of the resulting split. This is the most useful general-case application of this function.
• When the input is numeric, "expanded" creates as many columns as the maximum value of the input data. This is most useful when converting to mode = "binary".
• "list" creates a single new column that is structurally a list within a data.frame or data.table.

fixed

• When structure = "expanded" or structure = "list", it is possible to supply a regular expression containing the characters to split on. For example, to split on "", ", ;", or "|", you can set sep = "", |
| " or sep = "[|", and fixed = FALSE to split on any of those characters.

Note

This is more of a "legacy" or "convenience" wrapper function encompassing the features available in the separated functions of cSplit, concat.split.compact, concat.split.list, and concat.split.expanded.

Author(s)

Ananda Mahto

See Also

cSplit, concat.split.compact, concat.split.expanded, concat.split.list, concat.split.multiple

Examples

## Load some data
temp <- head(concat.test)

# Split up the second column, selecting by column number
concat.split(temp, 2)

# ... or by name, and drop the offensive first column
concat.split(temp, "Likes", drop = TRUE)

# The "Hates" column uses a different separator
concat.split(temp, "Hates", sep = ";", drop = TRUE)

## Not run:
# You'll get a warning here, when trying to retain the original values
concat.split(temp, 2, mode = "value", drop = TRUE)

## End(Not run)
# Try again. Notice the differing number of resulting columns
concat.split(temp, 2, structure = "expanded",
  mode = "value", type = "numeric", drop = TRUE)

# Let's try splitting some strings... Same syntax
concat.split(temp, 3, drop = TRUE)

# Strings can also be split to binary representations
concat.split(temp, 3, structure = "expanded",
  type = "character", fill = 0, drop = TRUE)

# Split up the "Likes column" into a list variable; retain original column
head(concat.split(concat.test, 2, structure = "list", drop = FALSE))

# View the structure of the output to verify
# that the new column is a list; note the
# difference between "Likes" and "Likes_list".
str(concat.split(temp, 2, structure = "list", drop = FALSE))

---

**concat.split.compact**  
*Split Concatenated Cells into a Condensed Format*

**Description**

The default splitting method for `concat.split`. Formerly based on `read.concat` but presently a simple wrapper for `cSplit`.

**Usage**

```r
concat.split.compact(data, split.col, sep = ",", drop = FALSE,
                     fixed = TRUE, ...)
```

**Arguments**

- `data`  
The source data.frame or data.table
- `split.col`  
The variable that needs to be split (either name or index position).
- `sep`  
The character separating each value.
- `drop`  
Logical. Should the original variable be dropped? Defaults to FALSE.
- `fixed`  
Logical. Should the split character be treated as a fixed pattern (TRUE) or a regular expression (FALSE)? Defaults to TRUE.
- `...`  
optional arguments to pass to cSplit.

**Value**

A data.table.
Note

This function no longer does anything different from `cSplit`. It is recommended that you transition your code to the `cSplit` function instead.

Author(s)

Ananda Mahto

See Also

`read.concat`, `cSplit`

Examples

```r
temp <- head(concat.test)
concat.split.compact(temp, "Likes")
concat.split.compact(temp, 4, ";")

## Extra arguments to cSplit
concat.split.compact(temp, "siblings", drop = TRUE, stripWhite = TRUE)
```

---

**concat.split.expanded**  
*Split Concatenated Values into their Corresponding Column Position*

Description

"Expand" concatenated numeric or character values to their relevant position in a `data.frame` or `data.table` or create a binary representation of such data.

Usage

```r
cSplit_e(data, split.col, sep = ",", mode = NULL, type = "numeric", drop = FALSE, fixed = TRUE, fill = NA)
```

Arguments

- `data`: The source `data.frame` or `data.table`.
- `split.col`: The variable that needs to be split (either name or index position).
- `sep`: The character separating each value. Can also be a regular expression.
- `mode`: Can be either "binary" (where presence of a number in a given column is converted to "1") or "value" (where the value is retained and not recoded to "1"). Defaults to "binary".
- `type`: Can be either "numeric" (where the items being split are integers) or "character" (where the items being split are character strings). Defaults to "numeric".
**concat.split.list**

- **drop** Logical. Should the original variable be dropped? Defaults to FALSE.
- **fixed** Used for *strsplit* for allowing regular expressions to be used.
- **fill** Desired "fill" value. Defaults to NA.

**Value**

A data.frame or a data.table depending on the source input.

**Author(s)**

Ananda Mahto

**See Also**

`concat.split`, `concat.split.list`, `concat.split.compact`, `concat.split.multiple`, `numMat`, `charMat`

**Examples**

```r
temp <- head(concat.test)
cSplit_e(temp, "Likes")
cSplit_e(temp, 4, ";", fill = NA)

## The old function name still works
csplit.split.expanded(temp, "Likes")
csplit.split.expanded(temp, 4, ";", fill = NA)
csplit.split.expanded(temp, 4, ";", mode = "value", drop = TRUE)
csplit.split.expanded(temp, "Siblings", type = "character", drop = TRUE)
```

**concat.split.list** *Split Concatenated Cells into a List Format*

**Description**

Takes a column in a data.frame or data.table with multiple values, splits the values into a list, and returns a new data.frame or data.table.

**Usage**

```r
csplit.l(data, split.col, sep = ",", drop = FALSE, fixed = FALSE)
```
Arguments

- **data**: The source data.frame or data.table.
- **split.col**: The variable that needs to be split (either name or index position).
- **sep**: The character separating each value. Can also be a regular expression.
- **drop**: Logical. Should the original variable be dropped? Defaults to FALSE.
- **fixed**: Used for `strsplit` for allowing regular expressions to be used.

Value

A data.frame or data.table with the concatenated column split and added as a list.

Author(s)

Ananda Mahto

See Also

`concat.split`, `concat.split.compact`, `concat.split.expanded`, `concat.split.multiple`

Examples

```r
temp <- head(concat.test)
str(cSplit_l(temp, "Likes"))
cSplit_l(temp, 4, ";")

## The old function name still works
str(concat.split.list(temp, "Likes"))
concat.split.list(temp, 4, ";")
concat.split.list(temp, 4, ";", drop = TRUE)
```

---

**concat.split.multiple**  
*Split Concatenated Cells and Optionally Reshape the Output*

Description

This is a wrapper for the `cSplit` function to maintain backwards compatibility with earlier versions of the "splitstackshape" package. It allows the user to split multiple columns at once and optionally convert the results into a "long" format.

Usage

```r
concat.split.multiple(data, split.cols, seps = ",", direction = "wide", ...)
```
**concat.test**

**Arguments**

data The source data frame or data.table.
split.cols A vector of columns that need to be split.
seps A vector of the separator character used in each column. If all columns use the same character, you can enter that single character.
direction The desired form of the resulting data.frame or data.table, either 'wide' or 'long'. Defaults to 'wide'.
...
Other arguments to `cSplit`.

**Value**

A data.table.

**Author(s)**

Ananda Mahto

**See Also**

`cSplit`, for which this is simply a wrapper, and `concat.split`, `concat.split.compact`, `concat.split.expanded`, `concat.split.multiple`, `Reshape`

**Examples**

```r
temp <- head(concat.test)
concat.split.multiple(temp, split.cols = c("Likes", "Hates", "Siblings"),
                     seps = c("", ";", ";", ";"))
concat.split.multiple(temp, split.cols = c("Likes", "Siblings"),
                     seps = ",", direction = "long")
```

**Example Dataset with Concatenated Cells**

This is a sample dataset to demonstrate the different features of the `concat.split` family of functions.

**Format**

A data.frame in which many columns contain concatenated cells
cSplit

Split Concatenated Values into Separate Values

Description

The cSplit function is designed to quickly and conveniently split concatenated data into separate values.

Usage

\[
c\text{Split}(\text{indt}, \text{splitCols}, \text{sep} = ",", \text{direction} = \text{"wide"}, \text{fixed} = \text{TRUE}, \\
\text{drop} = \text{TRUE}, \text{stripWhite} = \text{TRUE}, \text{makeEqual} = \text{NULL}, \text{type.convert} = \text{TRUE})
\]

Arguments

- \text{indt} \hspace{1cm} \text{The input data.frame or data.table.}
- \text{splitCols} \hspace{1cm} \text{The column or columns that need to be split.}
- \text{sep} \hspace{1cm} \text{The values that serve as a delimiter within each column. This can be a single value if all columns have the same delimiter, or a vector of values in the same order as the delimiters in each of the splitCols.}
- \text{direction} \hspace{1cm} \text{The desired direction of the results, either "wide" or "long".}
- \text{fixed} \hspace{1cm} \text{Logical. Should the split character be treated as a fixed pattern (TRUE) or a regular expression (FALSE)? Defaults to TRUE.}
- \text{drop} \hspace{1cm} \text{Logical. Should the original concatenated column be dropped? Defaults to TRUE.}
- \text{stripWhite} \hspace{1cm} \text{Logical. If there is whitespace around the delimiter in the concatenated columns, should it be stripped prior to splitting? Defaults to TRUE.}
- \text{makeEqual} \hspace{1cm} \text{Logical. Should all groups be made to be the same length? Defaults to FALSE.}
- \text{type.convert} \hspace{1cm} \text{Logical. Should type.convert be used to convert the result of each column? This would add a little to the execution time.}

Value

A data.table with the values split into new columns or rows.

Note

The cSplit function replaces most of the earlier concat.split* functions. The earlier functions remain for compatibility purposes, but now they are essentially wrappers for the cSplit function. If you know that all values in the column would have the same number of values per row after being split, you should use the cSplit.f function instead, which uses fread instead of strsplit and is generally faster.
cSplit_f

Split Concatenated Cells in a data.frame or a data.table

Description
A variation of the concat.split family of functions designed for large rectangular datasets. This function makes use of fread from the "data.table" package for very speedy splitting of concatenated columns of data.

Usage
\[
\text{cSplit}_f(\text{indt}, \text{splitCols}, \text{sep}, \text{drop} = \text{TRUE}, \text{dotsub} = \text{"|"}, \\
\text{stripWhite} = \text{FALSE})
\]

Arguments
- **indt**: The input data.frame or data.table.
- **splitCols**: The columns that need to be split up.

Examples
```r
## Sample data
temp <- head(concat.test)

## Split the "Likes" column
cSplit(temp, "Likes")

## Split the "Likes" and "Hates" columns --
## they have different delimiters...
cSplit(temp, c("Likes", "Hates"), c(","", ":"))

## Split "Siblings" into a long form...
cSplit(temp, "Siblings", ",", direction = "long")

## Split "Siblings" into a long form, removing extra whitespace
## direction = "long", stripWhite = TRUE)
cSplit(temp, "Siblings", ",", direction = "long", stripWhite = TRUE)

## Split a vector
y <- c("a_b_c", "a_b", "c_a_b")
cSplit(as.data.table(y), "y", ")
```

See Also
concat.split, cSplit_f

Author(s)
Ananda Mahto
sep

The character or characters that serve as delimiters within the columns that need to be split up. If different columns use different delimiters, enter the delimiters as a character vector.

drop

Logical. Should the original columns be dropped? Defaults to TRUE.

dotsub

The character that should be substituted as a delimiter if sep = ".". fread does not seem to work nicely with sep = ".", so it needs to be substituted. By default, this function will substitute "." with "|".

stripwhite

Logical. Should whitespace be stripped before writing to the temporary file? Defaults to FALSE.

Details

While the general concat.split functions (cSplit in particular) are able to handle "unbalanced" datasets (for example, where the number of fields in a given column might differ from row to row) because of the nature of fread from the 'data.table' package, this function does not support such data types.

Value

A data.table.

Author(s)

Ananda Mahto. Thanks also to Arun Srinivasan for helping to refine this function.

References

http://stackoverflow.com/a/19231054/1270695

Examples

```r
## Sample data. Change `n` to larger values to test on larger data
set.seed(1)
n <- 10
mydf <- data.frame(id = sequence(n))
mydf <- within(mydf, {
  v3 <- do.call(paste, c(data.frame(matrix(sample(letters, n*4, TRUE), ncol = 4), sep = "_"))
  v2 <- do.call(paste, c(data.frame(matrix(sample(LETTERS, n*3, TRUE), ncol = 3), sep = "."))
  v1 <- do.call(paste, c(data.frame(matrix(sample(10, n*2, TRUE), ncol = 2), sep = "-"))
})
mydf

cSplit_f(mydf, splitCols = c("v1", "v2", "v3"), sep = c("-", ".", "_"))
```
**expandRows**  
*Expand the Rows of a Dataset*

**Description**

Expands (replicates) the rows of a data.frame or a data.table, either by a fixed number, a specified vector, or a value contained in one of the columns in the source data.frame or a data.table.

**Usage**

```
expandRows(dataset, count, count.is.col = TRUE, drop = TRUE)
```

**Arguments**

- `dataset` The input data.frame or data.table.
- `count` The numeric vector of counts OR the column from the dataset that contains the count data. If `count` is a single digit, it is assumed that all rows should be repeated by this amount.
- `count.is.col` Logical. Is the count value a column from the input dataset? Defaults to `TRUE`.
- `drop` Logical. If `count.is.col = TRUE`, should the "count" column be dropped from the result? Defaults to `TRUE`.

**Value**

A data.frame or a data.table, depending on the input.

**Author(s)**

Ananda Mahto

**References**

[http://stackoverflow.com/a/19519828/1270695](http://stackoverflow.com/a/19519828/1270695)

**Examples**

```r
mydf <- data.frame(x = c("a", "b", "q"),  
y = c("c", "d", "r"),  
  count = c(2, 5, 3))
library(data.table)
DT <- as.data.table(mydf)
mydf
expandRows(mydf, "count")
expandRows(DT, "count", drop = FALSE)
expandRows(mydf, count = 3) ## This takes values from the third column!
expandRows(mydf, count = 3, count.is.col = FALSE)
expandRows(mydf, count = c(1, 5, 9), count.is.col = FALSE)
expandRows(DT, count = c(1, 5, 9), count.is.col = FALSE)
```
FacetsToChars

Convert All Factor Columns to Character Columns

Description

Sometimes, we forget to use the `stringsAsFactors` argument when using `read.table` and related functions. By default, R converts character columns to factors. Instead of re-reading the data, the FacetsToChars function will identify which columns are currently factors, and convert them all to characters.

Usage

```
FacetsToChars(mydf)
```

Arguments

- `mydf` The name of your data frame

Author(s)

Ananda Mahto

See Also

- `read.table`

Examples

```r
## Some example data
dat <- data.frame(title = c("title1", "title2", "title3"),
                  author = c("author1", "author2", "author3"),
                  customerID = c(1, 2, 1))

str(dat) # current structure
dat2 <- splitstackshape:::FacetsToChars(dat)
str(dat2) # Your new object
str(dat) # Original object is unaffected
```
Add an "id" Variable to a Dataset

Description

Many functions will not work properly if there are duplicated ID variables in a dataset. This function is a convenience function for .N from the "data.table" package to create an ".id" variable that when used in conjunction with the existing ID variables, should be unique.

Usage

getanID(data, id.vars = NULL)

Arguments

data
The input data.frame or data.table.

to be treated as ID variables. Defaults to NULL, at which point all variables are used to create the new ID variable.

Value

The input dataset (as a data.table) if ID variables are unique, or the input dataset with a new column named ".id".

Author(s)

Ananda Mahto

Examples

```r
mydf <- data.frame(IDA = c("a", "a", "a", "b", "b"),
                   IDB = c(1, 1, 1, 1, 1), values = 1:5)
mydf
gatanID(mydf, "IDA")

mydf <- data.frame(IDA = c("a", "a", "a", "b", "b"),
                   IDB = c(1, 2, 1, 1, 2), values = 1:5)
mydf
gatanID(mydf, 1:2)
```
### listCol_l

**Unlist a Column Stored as a List**

**Description**

Unlists a column stored as a list into a long form.

**Usage**

```
listCol_l(inDT, listcol, drop = TRUE)
```

**Arguments**

- `inDT`: The input dataset.
- `listcol`: The name of the column stored as a list.
- `drop`: Logical. Should the original column be dropped? Defaults to TRUE.

**Value**

A data.table.

**Author(s)**

Ananda Mahto

**See Also**

`listCol_w` to flatten a list column into a "wide" format.

**Examples**

```r
dat <- data.frame(A = 1:3, B = list(c(1, 2), c(1, 3, 5), c(4)))
listCol_l(dat, "B")
```

### listCol_w

**Flatten a Column Stored as a List**

**Description**

Flattens a column stored as a list into a wide form.

**Usage**

```
listCol_w(inDT, listcol, drop = TRUE, fill = NA_character_)
```
merged.stack

Arguments

- `input` The input dataset.
- `listcol` The name of the column stored as a list.
- `drop` Logical. Should the original column be dropped? Defaults to TRUE.
- `fill` The desired fill value. Defaults to `NA_character_`.

Value

A `data.table`.

Author(s)

Ananda Mahto

See Also

- `listCol_l` to unlist a list column into a "long" format.

Examples

dat <- data.frame(A = 1:3, B = I(list(c(1, 2), c(1, 3, 5), c(4))))
listCol_w(dat, "B")

merged.stack

Take a List of Stacked data.tables and Merge Them

Description

A wrapper around the `Stacked` function to merge the resulting list into a single `data.table`.

Usage

```
merged.stack(data, id.vars = NULL, var.stubs, sep, keep.all = TRUE, ...)
```

Arguments

- `data` The input `data.frame`.
- `id.vars` The columns to be used as "ID" variables. Defaults to `NULL`, at which point, all names which are not identified as variable groups are used as the identifiers.
- `var.stubs` The prefixes of the variable groups.
- `sep` The character that separates the "variable name" from the "times" in the source `data.frame`. Alternatively, can be set to "var.stubs" (in quotes) if you do not have a value for `sep`.
- `keep.all` Logical. Should all the variables in the source `data.frame` be kept (`keep.all` = TRUE) or only those which comprise the `id.vars` and split data from the `var.stubs` (`keep.all` = FALSE).
Other arguments to be passed on to `Stacked` (for example, keep.rownames to retain the rownames of the input dataset, or atStart, in case sep = "var.stubs" is specified).

Value

A merged data.table.

Note

The keyed argument to `Stacked` has been hard-coded to TRUE to make merge work.

Author(s)

Ananda Mahto

See Also

`Stacked`, `Reshape`

Examples

```r
set.seed(1)
mydf <- data.frame(id_1 = 1:6, id_2 = c("A", "B"),
                   varA.1 = sample(letters, 6),
                   varA.2 = sample(letters, 6),
                   varA.3 = sample(letters, 6),
                   varB.2 = sample(10, 6),
                   varB.3 = sample(10, 6),
                   varC.3 = rnorm(6))
mydf
merged.stack(mydf, var.stubs = c("varA", "varB", "varC"), sep = ".")
```
NoSep

Arguments

  data  The input data.frame.
  invec The names you want.

Value

  A character vector of the desired names.

Author(s)

  Ananda Mahto

Examples

  mydf <- data.frame(a = 1:2, b = 3:4, c = 5:6)
  splitstackshape:::Names(mydf, c("a", "c"))
  splitstackshape:::Names(mydf, c(1, 3))

Description

  Used to split strings like "Abc8" into "Abc" and "8".

Usage

  NoSep(data, charfirst = TRUE)

Arguments

  data  The vector of strings to be split.
  charfirst  Is the string constructed with characters at the start or numbers? Defaults to TRUE.

Value

  A data.frame with two columns, .var and .time_1.

Note

  This is a helper function for the Stacked and Reshape functions.

Author(s)

  Ananda Mahto
numMat

Create a Numeric Matrix from a List of Values

Description

Create a numeric matrix from a list of values

Usage

numMat(listOfValues, fill = NA, mode = "binary")

Arguments

- **listOfValues**: A list of input values to be inserted in a matrix.
- **fill**: The initializing fill value for the empty matrix.
- **mode**: Either "binary" or "value". Defaults to "binary".

Details

This is primarily a helper function for the `concat.split` function when creating the "expanded" structure. The input is anticipated to be a list of values obtained using `strsplit`.

Value

A matrix.

Author(s)

Ananda Mahto

See Also

`strsplit`, `charMat`
Examples

```r
invec <- c("1,2,4,5,6", "1,2,4,5,6", "1,2,4,5,6", 
               "1,2,4,5,6", "-1,1,2,5,6", "1,2,5,6")
A <- strsplit(invec, ",")
splitstackshape:::numMat(A)
splitstackshape:::numMat(A, fill = 0)
splitstackshape:::numMat(A, mode = "value")
```

othernames

Extract All Names From a Dataset Other Than the Ones Listed

Description

A convenience function for `setdiff(names(data), -some_vector_of_names-)`.

Usage

```r
othernames(data, toremove)
```

Arguments

- `data` The input data.frame.
- `toremove` The names you want to exclude.

Value

A character vector of the remaining names.

Author(s)

Ananda Mahto

See Also

`setdiff`

Examples

```r
mydf <- data.frame(a = 1:2, b = 3:4, c = 5:6)
splitstackshape:::othernames(mydf, "a")
```
read.concat  Read Concatenated Character Vectors Into a data.frame

Description

Originally a helper function for the `concat.split.compact` function. This function has now been effectively replaced by `cSplit`.

Usage

`read.concat(data, col.prefix, sep, ...)`

Arguments

- `data` The input data.
- `col.prefix` The desired column prefix for the output data.frame.
- `sep` The character that acts as a delimiter.
- `...` Other arguments to pass to `read.table`.

Value

A data.frame

Author(s)

Ananda Mahto

See Also

`read.table`

Examples

```r
vec <- c("a,b", "c,d,e", "f, g", "h, i, j,k")
splitstackshape:::read.concat(vec, "var", ",")

## More than 5 lines the same
## 'read.table' would fail with this
vec <- c("12,51,34,17", "84,28,17,10", "11,43,28,15", "80,26,17,91", "10,41,25,13", "97,35,23,12,13")
splitstackshape:::read.concat(vec, "var", ",")
```
Reshape

Reshape Wide Data Into a Semi-long Form

Description

The `reshape` function in base R is very handy when you want a semi-long (or semi-wide) data frame. However, base R’s `reshape` has problems with "unbalanced" panel data, for instance data where one variable was measured at three points in time, and another only twice.

Usage

```r
Reshape(data, id.vars = NULL, var.stubs, sep = ".", rm.rownames = TRUE, ...)
```

Arguments

- `data` The source data frame.
- `id.vars` The variables that serve as unique identifiers. Defaults to `NULL`, at which point, all names which are not identified as variable groups are used as the identifiers.
- `var.stubs` The prefixes of the variable groups.
- `sep` The character that separates the "variable name" from the "times" in the wide data frame.
- `rm.rownames` Logical. `reshape` creates some long distracting rownames that do not seem to serve much purpose. This argument is set to `TRUE` to remove the rownames by default.
- `...` Further arguments to `NoSep` in case the separator is of a different form.

Details

This function was written to overcome that limitation of dealing with unbalanced data, but is also appropriate for basic wide-to-long reshaping tasks.

Related functions like `stack` in base R and `melt` in "reshape2" are also very handy when you want a "long" reshaping of data, but they result in a very long structuring of your data, not the "semi-wide" format that `reshape` produces.

Value

A "long" data frame of the reshaped data that retains the attributes added by base R’s `reshape` function.

Author(s)

Ananda Mahto
See Also

`Stacked, stack, reshape, melt`

Examples

```r
set.seed(1)
mydf <- data.frame(id_1 = 1:6, id_2 = c("A", "B"), varA.1 = sample(letters, 6),
                      varA.2 = sample(letters, 6), varA.3 = sample(letters, 6),
                      varB.2 = sample(10, 6), varB.3 = sample(10, 6),
                      varC.3 = rnorm(6))

mydf

## Note that these data are unbalanced
## reshape() will not work
## Not run:
reshape(mydf, direction = "long", idvar=1:2, varying=3:ncol(mydf))

## End(Not run)
## The Reshape() function can handle such scenarios
Reshape(mydf, id.vars = c("id_1", "id_2"),
        var.stubs = c("varA", "varB", "varC"))
```

Description

A function to conveniently stack groups of wide columns into a long form which can then be merged together.

Usage

```r
Stacked(data, id.vars = NULL, var.stubs, sep, keep.all = TRUE,
         keyed = TRUE, keep.rownames = FALSE, ...)
```

Arguments

- `data`: The source data.frame.
- `id.vars`: The variables that serve as unique identifiers. Defaults to NULL, at which point, all names which are not identified as variable groups are used as the identifiers.
- `var.stubs`: The prefixes of the variable groups.
- `sep`: The character that separates the "variable name" from the "times" in the wide data.frame. Alternatively, can be set to "var.stubs" (in quotes) if you do not have a value for sep.
Stacked

keep.all Logical. Should all the variables from the source data.frame be kept (keep.all = TRUE) or should the resulting data.table comprise only columns for the id.vars, var.stubs, and "times" (keep.all = FALSE). Other variables are recycled to appropriate length. For this to work, both id.vars and var.stubs must be specified.

keyed Logical. Should the Stacked function automatically set the key for the resulting data.tables. If TRUE (default) the key is set to the id.vars and the "time" variables that are created by Stacked.

keep.rownames Logical. Should rownames be kept when converting the input to a data.table? Defaults to FALSE.

... Other arguments to be passed on when sep = "var.stubs" (specifically, atStart: A logical argument to indicate whether the stubs come at the start or at the end of the variable names).

Value

A list of data.tables with one data.table for each "var.stub". The key is set to the id.vars and .time_# vars.

Note

This is the function internally called by merged.stack.

Author(s)

Ananda Mahto

See Also

stack, melt from "reshape2".

Examples

```r
set.seed(1)
mydf <- data.frame(id_1 = 1:6, id_2 = c("A", "B"),
    varA.1 = sample(letters, 6),
    varA.2 = sample(letters, 6),
    varA.3 = sample(letters, 6),
    varB.2 = sample(10, 6),
    varB.3 = sample(10, 6),
    varC.3 = rnorm(6))
mydf
Stacked(data = mydf, var.stubs = c("varA", "varB", "varC"), sep = ".")
```
Take a Stratified Sample From a Dataset

Description

The `stratified` function samples from a data.frame or a data.table in which one or more columns can be used as a "stratification" or "grouping" variable. The result is a new data.table with the specified number of samples from each group.

Usage

```r
stratified(indt, group, size, select = NULL, replace = FALSE,
    keep.rownames = FALSE, bothSets = FALSE, ...)  
```

Arguments

- `indt` The input data.frame or data.table.
- `group` The column or columns that should be used to create the groups. Can be a character vector of column names (recommended) or a numeric vector of column positions. Generally, if you are using more than one variable to create your "strata", you should list them in the order of slowest varying to quickest varying. This can be a vector of names or column indexes.
- `size` The desired sample size.
  - If `size` is a value between 0 and 1 expressed as a decimal, `size` is set to be proportional to the number of observations per group.
  - If `size` is a single positive integer, it will be assumed that you want the same number of samples from each group.
  - If `size` is a named vector, the function will check to see whether the length of the vector matches the number of groups and that the names match the group names.
- `select` A named list containing levels from the "group" variables in which you are interested. The list names must be present as variable names for the input dataset.
- `replace` Logical. Should sampling be with replacement? Defaults to `FALSE`.
- `keep.rownames` Logical. If the input is a data.frame or a matrix, as.data.table would normally drop the rownames. If `TRUE`, the rownames would be retained in a column named `rn`. Defaults to `FALSE`.
- `bothSets` Logical. Should both the sampled and non-sampled sets be returned as a list? Defaults to `FALSE`.
- `...` Optional arguments to `sample`.

Value

If `bothSets = FALSE`, a list of two data.tables; otherwise, a data.table.
Note

Slightly different sizes than requested: Because of how computers deal with floating-point arithmetic, and because R uses a "round to even" approach, the size per strata that results when specifying a proportionate sample may be slightly higher or lower per strata than you might have expected.

Author(s)

Ananda Mahto

See Also

strata from the "strata" package; sample_n and sample_frac from "dplyr".

Examples

# Generate a sample data.frame to play with
defset.seed(1)

dat1 <- data.frame(ID = 1:100,
                     A = sample(c("AA", "BB", "CC", "DD", "EE"), 100, replace = TRUE),
                     B = rnorm(100), C = abs(round(rnorm(100), digits=1)),
                     D = sample(c("CA", "NY", "TX"), 100, replace = TRUE),
                     E = sample(c("M", "F"), 100, replace = TRUE))

# Let's take a 10% sample from all -A- groups in dat1
stratified(dat1, "A", .1)

# Let's take a 10% sample from only "AA" and "BB" groups from -A- in dat1
stratified(dat1, "A", .1, select = list(A = c("AA", "BB")))

# Let's take 5 samples from all -D- groups in dat1,
# specified by column number
stratified(dat1, group = 5, size = 5)

# Use a two-column strata: -E- and -D-
# -E- varies more slowly, so it is better to put that first
stratified(dat1, c("E", "D"), size = .15)

# Use a two-column strata (-E- and -D-) but only interested in
# cases where -E- == "M"
stratified(dat1, c("E", "D"), .15, select = list(E = "M"))

## As above, but where -E- == "M" and -D- == "CA" or "TX"
stratified(dat1, c("E", "D"), .15,
           select = list(E = "M", D = c("CA", "TX")))

# Use a three-column strata: -E-, -D-, and -A-
s.out <- stratified(dat1, c("E", "D", "A"), size = 2)
Index

*Topic **datasets**
  concat.test, 11

*Topic **package**
  splitstackshape-package, 2

  charMat, 4, 9, 22
  concat.split, 2, 4, 5, 7, 9–11, 13, 22
  concat.split.compact, 6, 7, 9–11, 24
  concat.split.expanded, 6, 8, 10, 11
  concat.split.list, 6, 9, 9
  concat.split.multiple, 6, 9, 10, 11
  concat.test, 11
  concatenated(concat.test), 11
  cSplit, 5–8, 10, 11, 12, 14, 24
  cSplit_e (concat.split.expanded), 8
  cSplit_f, 12, 13, 13
  cSplit_l (concat.split.list), 9

  data.frame, 6, 28
  data.table, 12, 27, 28
  dcast, 2

  expandRows, 15

  FacsToChars, 16, 16
  fread, 12, 13

  getanID, 17

  key, 27

  list, 6
  listCol_1, 18, 19
  listCol_w, 18, 18

  melt, 2, 25–27
  merge, 19, 26
  merged.stack, 19, 27

  Names, 20
  NoSep, 21, 25

  numMat, 4, 9, 22

  othernames, 23

  read.concat, 7, 8, 24
  read.table, 16
  Reshape, 11, 20, 21, 25
  reshape, 2, 25, 26

  sample, 28
  sample_frac, 29
  sample_n, 29
  setdiff, 23
  splitstackshape
    (splitstackshape-package), 2
  splitstackshape-package, 2
  stack, 25–27
  Stacked, 19–21, 26, 26
  strata, 29
  stratified, 28, 28
  strsplit, 4, 10, 12, 22

  type.convert, 12