

The reshape Package

February 8, 2009

Type Package

Title Flexibly reshape data.

Version 0.8.2

Date 2008-11-04

Author Hadley Wickham <h.wickham@gmail.com>

Maintainer Hadley Wickham <h.wickham@gmail.com>

Description Reshape lets you flexibly restructure and aggregate data using just two functions: melt and cast.

URL <http://had.co.nz/reshape>

Depends R (>= 2.6.1), plyr

License MIT

LazyData true

R topics documented:

add.all.combinations	3
add.missing.levels	4
all.vars.character	4
as.data.frame.cast_df	5
as.data.frame.cast_matrix	5
as.matrix.cast_df	6
as.matrix.cast_matrix	7
cast	7
cast_matrix	9
cast_parse_formula	10
check_formula	11
clean.vars	11
colsplit	12
combine_factor	12

compute.margins	13
condense.df	14
condense	14
dim_names	15
expand.grid.df	15
expand	16
French fries	16
funstofun	17
guess_value	18
margin.vars	18
melt	19
melt.array	19
melt.cast_df	20
melt.cast_matrix	21
melt_check	21
melt.data.frame	22
melt.default	23
melt.list	24
merge_all	24
merge_recurse	25
namerows	26
nested.by	26
nulldefault	27
prettyprint	27
rdimnames	28
recast	28
rename	29
rescaler	30
rescaler.data.frame	31
rescaler.default	31
rescaler.matrix	32
reshape1	32
round_any	34
Smiths	35
sort_df	35
sparseby	36
stamp	37
str.cast_matrix	38
strip.dups	38
tidystamp	39
Tips	39
uniquedefault	40
untable	40
updatelist	41

`add.all.combinations`*Add all combinations*

Description

Add all combinations of the given rows and columns to the data frames.

Usage

```
add.all.combinations(data, vars = list(NULL), fill=NA)
```

Arguments

<code>data</code>	<code>data.frame</code>
<code>vars</code>	variables (list of character vectors)
<code>fill</code>	value to fill structural missings with

Details

This function is used to ensure that we have a matrix of the appropriate dimensionality with no missing cells.

Author(s)

Hadley Wickham <h.wickham@gmail.com>

Examples

```
rdunif <-  
function(n=20, min=0, max=10) floor(runif(n,min, max))  
df <- data.frame(a = rdunif(), b = rdunif(), c = rdunif(), result=1:20)  
add.all.combinations(df)  
add.all.combinations(df, list("a", "b"))  
add.all.combinations(df, list("a", "b"), fill=0)  
add.all.combinations(df, list(c("a", "b")))  
add.all.combinations(df, list("a", "b", "c"))  
add.all.combinations(df, list(c("a", "b"), "c"))  
add.all.combinations(df, list(c("a", "b", "c")))
```

`add.missing.levels` *Add in any missing values*

Description

@keyword internal

Usage

```
add.missing.levels(data, vars=NULL, fill=NA)
```

Arguments

data

vars

fill

Author(s)

Hadley Wickham <h.wickham@gmail.com>

`all.vars.character` *Get all variables*

Description

All variables in character string of formula.

Usage

```
all.vars.character(formula, blank.char = ".")
```

Arguments

formula

blank.char

Details

Removes .

Author(s)

Hadley Wickham <h.wickham@gmail.com>

Examples

```
all.vars.character("a + b")
all.vars.character("a + b | c")
all.vars.character("a + b")
all.vars.character(". ~ a + b")
all.vars.character("a ~ b | c + d + e")
```

```
as.data.frame.cast_df
  as.data.frame.cast_df
```

Description

Convert cast data.frame into a matrix

Usage

```
as.data.frame.cast_df(x, ...)
```

Arguments

x
...

Details

Strips off cast related attributes so data frame becomes a normal data frame

Author(s)

Hadley Wickham <h.wickham@gmail.com>

```
as.data.frame.cast_matrix
  as.data.frame.cast_matrix
```

Description

Convert cast matrix into a data frame

Usage

```
as.data.frame.cast_matrix(x, row.names, optional, ...)
```

Arguments

x
row.names
optional
...

Details

Converts a matrix produced by cast into a data frame with appropriate id columns.

Author(s)

Hadley Wickham <h.wickham@gmail.com>

as.matrix.cast_df *as.matrix.cast_df*

Description

Convert cast data.frame into a matrix

Usage

```
as.matrix.cast_df(x, ...)
```

Arguments

x
...

Details

Converts a data frame produced by cast into a matrix with appropriate dimnames.

Author(s)

Hadley Wickham <h.wickham@gmail.com>

```
as.matrix.cast_matrix
      as.matrix.cast_matrix
```

Description

Convert cast matrix into a matrix

Usage

```
as.matrix.cast_matrix(x, ...)
```

Arguments

x
...

Details

Strips off cast related attributes so matrix becomes a normal matrix

Author(s)

Hadley Wickham <h.wickham@gmail.com>

```
cast          Cast function
```

Description

Cast a molten data frame into the reshaped or aggregated form you want

Usage

```
cast(data, formula = ... ~ variable, fun.aggregate=NULL, ..., margins=FALSE, subset)
```

Arguments

data	molten data frame, see melt
formula	casting formula, see details for specifics
fun.aggregate	aggregation function
...	further arguments are passed to aggregating function
margins	vector of variable names (can include "grand_col" and "grand_row") to compute margins for, or TRUE to computer all margins

```

subset      logical vector to subset data set with before reshaping
df          argument used internally
fill       value with which to fill in structural missings, defaults to value from applying
           fun.aggregate to 0 length vector

add.missing
value

```

Details

Along with `melt` and `recast`, this is the only function you should ever need to use. Once you have melted your data, `cast` will arrange it into the form you desire based on the specification given by formula.

The cast formula has the following format: `x_variable + x_2 ~ y_variable + y_2 ~ z_variable ~ ... | list_variable + ...`. The order of the variables makes a difference. The first varies slowest, and the last fastest. There are a couple of special variables: `"..."` represents all other variables not used in the formula and `"."` represents no variable, so you can do `formula=var1 ~ .`

Creating high-D arrays is simple, and allows a class of transformations that are hard without `apply` and `sweep`

If the combination of variables you supply does not uniquely identify one row in the original data set, you will need to supply an aggregating function, `fun.aggregate`. This function should take a vector of numbers and return a summary statistic(s). It must return the same number of arguments regardless of the length of the input vector. If it returns multiple value you can use `"result_variable"` to control where they appear. By default they will appear as the last column variable.

The `margins` argument should be passed a vector of variable names, eg. `c("month", "day")`. It will silently drop any variables that can not be margined over. You can also use `"grand_col"` and `"grand_row"` to get grand row and column margins respectively.

Subset takes a logical vector that will be evaluated in the context of `data`, so you can do something like `subset = variable=="length"`

All the actual reshaping is done by `reshape1`, see its documentation for details of the implementation

Author(s)

Hadley Wickham <h.wickham@gmail.com>

See Also

`reshape1`, <http://had.co.nz/reshape/>

Examples

```

#Air quality example
names(airquality) <- tolower(names(airquality))
aqm <- melt(airquality, id=c("month", "day"), na.rm=TRUE)

cast(aqm, day ~ month ~ variable)

```

```

cast(aqm, month ~ variable, mean)
cast(aqm, month ~ . | variable, mean)
cast(aqm, month ~ variable, mean, margins=c("grand_row", "grand_col"))
cast(aqm, day ~ month, mean, subset=variable=="ozone")
cast(aqm, month ~ variable, range)
cast(aqm, month ~ variable + result_variable, range)
cast(aqm, variable ~ month ~ result_variable, range)

#Chick weight example
names(ChickWeight) <- tolower(names(ChickWeight))
chick_m <- melt(ChickWeight, id=2:4, na.rm=TRUE)

cast(chick_m, time ~ variable, mean) # average effect of time
cast(chick_m, diet ~ variable, mean) # average effect of diet
cast(chick_m, diet ~ time ~ variable, mean) # average effect of diet & time

# How many chicks at each time? - checking for balance
cast(chick_m, time ~ diet, length)
cast(chick_m, chick ~ time, mean)
cast(chick_m, chick ~ time, mean, subset=time < 10 & chick < 20)

cast(chick_m, diet + chick ~ time)
cast(chick_m, chick ~ time ~ diet)
cast(chick_m, diet + chick ~ time, mean, margins="diet")

#Tips example
cast(melt(tips), sex ~ smoker, mean, subset=variable=="total_bill")
cast(melt(tips), sex ~ smoker | variable, mean)

ff_d <- melt(french_fries, id=1:4, na.rm=TRUE)
cast(ff_d, subject ~ time, length)
cast(ff_d, subject ~ time, length, fill=0)
cast(ff_d, subject ~ time, function(x) 30 - length(x))
cast(ff_d, subject ~ time, function(x) 30 - length(x), fill=30)
cast(ff_d, variable ~ ., c(min, max))
cast(ff_d, variable ~ ., function(x) quantile(x,c(0.25,0.5)))
cast(ff_d, treatment ~ variable, mean, margins=c("grand_col", "grand_row"))
cast(ff_d, treatment + subject ~ variable, mean, margins="treatment")
lattice::xyplot(`1` ~ `2` | variable, cast(ff_d, ... ~ rep), aspect="iso")

```

cast_matrix

Cast matrix.

Description

Create a new cast matrix

Usage

```
cast_matrix(m, dimnames)
```

Arguments

m matrix to turn into cast matrix
dimnames list of dimension names (as data.frames), row, col, ...

Details

For internal use only

Value

object of type `cast_matrix`

Author(s)

Hadley Wickham <h.wickham@gmail.com>

cast_parse_formula *Cast parse formula*

Description

Parse formula for casting

Usage

```
cast_parse_formula(formula = "... ~ variable", varnames)
```

Arguments

formula
varnames

Details

@value row character vector of row names @value col character vector of column names @value
aggregate boolean whether aggregation will occur @keyword internal

Value

row character vector of row names

col character vector of column names

aggregate boolean whether aggregation will occur

Author(s)

Hadley Wickham <h.wickham@gmail.com>

Examples

```
cast_parse_formula("a + ...", letters[1:6])
cast_parse_formula("a | ...", letters[1:6])
cast_parse_formula("a + b ~ c ~ . | ...", letters[1:6])
```

check_formula	<i>Check formula</i>
---------------	----------------------

Description

Checks that formula is a valid reshaping formula.

Usage

```
check_formula(formula, varnames)
```

Arguments

formula	formula to check
varnames	vector of variable names

Author(s)

Hadley Wickham <h.wickham@gmail.com>

clean.vars	<i>Clean variables.</i>
------------	-------------------------

Description

Clean variable list for reshaping.

Usage

```
clean.vars(vars)
```

Arguments

vars	vector of variable names
------	--------------------------

Value

Vector of "real" variable names (excluding result_variable etc.)

Author(s)

Hadley Wickham <h.wickham@gmail.com>

`colsplit`*Split a vector into multiple columns*

Description

This function can be used to split up a column that has been pasted together.

Usage

```
colsplit(x, split="", names)
```

Arguments

<code>x</code>	character vector or factor to split up
<code>split</code>	regular expression to split on
<code>names</code>	names for output columns

Author(s)

Hadley Wickham <h.wickham@gmail.com>

`combine_factor`*Combine factor levels*

Description

Convenience function to make it easy to combine multiple levels

Usage

```
combine_factor(fac, variable=levels(fac), other.label="Other")
```

Arguments

<code>fac</code>	factor variable
<code>variable</code>	either a vector of . See examples for more details.
<code>other.label</code>	label for other level

Author(s)

Hadley Wickham <h.wickham@gmail.com>

Examples

```
df <- data.frame(a = LETTERS[sample(5, 15, replace=TRUE)], y = rnorm(15))
combine_factor(df$a, c(1,2,2,1,2))
combine_factor(df$a, c(1:4, 1))
(f <- reorder(df$a, df$y))
percent <- tapply(abs(df$y), df$a, sum)
combine_factor(f, c(order(percent)[1:3]))
```

compute.margins *Compute margins*

Description

Compute marginal values.

Usage

```
compute.margins(data, margins, vars, fun.aggregate, ..., df=FALSE)
```

Arguments

data	data frame
margins	margins to compute
vars	all id variables
fun.aggregate	aggregation function
...	other argument passed to aggregation function
df	

Author(s)

Hadley Wickham <h.wickham@gmail.com>

<code>condense.df</code>	<i>Condense a data frame</i>
--------------------------	------------------------------

Description

Condense

Usage

```
condense.df(data, variables, fun, ...)
```

Arguments

<code>data</code>	data frame
<code>variables</code>	character vector of variables to condense over
<code>fun</code>	function to condense with
<code>...</code>	arguments passed to condensing function

Author(s)

Hadley Wickham <h.wickham@gmail.com>

<code>condense</code>	<i>Condense</i>
-----------------------	-----------------

Description

Condense a data frame.

Usage

```
condense(data, variables, fun, ...)
```

Arguments

<code>data</code>	data frame
<code>variables</code>	variables to condense over
<code>fun</code>	aggregating function, may multiple values
<code>...</code>	further arguments passed on to aggregating function

Details

Works very much like `by`, but keeps data in original data frame format. Results column is a list, so that each cell may contain an object or a vector etc. Assumes data is in molten format. Aggregating function must return the same number of arguments for all input.

Author(s)

Hadley Wickham <h.wickham@gmail.com>

dim_names *Dimension names*

Description

Convenience method for extracting row and column names

Usage

```
dim_names(data, vars)
```

Arguments

data	data frame
vars	variables to use

Author(s)

Hadley Wickham <h.wickham@gmail.com>

expand.grid.df *Expand grid*

Description

Expand grid of data frames

Usage

```
expand.grid.df(..., unique=TRUE)
```

Arguments

...	list of data frames (first varies fastest)
unique	only use unique rows?

Details

Creates new data frame containing all combination of rows from data.frames in ...

Author(s)

Hadley Wickham <h.wickham@gmail.com>

Examples

```

expand.grid.df(data.frame(a=1,b=1:2))
expand.grid.df(data.frame(a=1,b=1:2), data.frame())
expand.grid.df(data.frame(a=1,b=1:2), data.frame(c=1:2, d=1:2))
expand.grid.df(data.frame(a=1,b=1:2), data.frame(c=1:2, d=1:2), data.frame(e=c("a","b")))

```

expand

Expand

Description

Expand out condensed data frame.

Usage

```
expand(data)
```

Arguments

data condensed data frame

Details

If aggregating function supplied to condense returns multiple values, this function "melts" it again, creating a new column called result_variable.

If the aggregating function is a named vector, then those names will be used, otherwise will be number X1, X2, ..., Xn etc.

Author(s)

Hadley Wickham <h.wickham@gmail.com>

French fries

Sensory data from a french fries experiment

Description

This data was collected from a sensory experiment conducted at Iowa State University in 2004. The investigators were interested in the effect of using three different fryer oils had on the taste of the fries.

Variables:

time in weeks from start of study.

treatment (type of oil),

subject,
replicate,
potato-y flavour,
buttery flavour,
grassy flavour,
rancid flavour,
painty flavour

Usage

```
data(french_fries)
```

Format

A data frame with 696 rows and 9 variables

funstofun

Aggregate multiple functions into a single function

Description

Combine multiple functions to a single function returning a named vector of outputs

Usage

```
funstofun(...)
```

Arguments

... functions to combine

Details

Each function should produce a single number as output

Author(s)

Hadley Wickham <h.wickham@gmail.com>

Examples

```
funstofun(min, max)(1:10)  
funstofun(length, mean, var)(rnorm(100))
```

guess_value	<i>Guess value</i>
-------------	--------------------

Description

Guess name of value column

Usage

```
guess_value(df)
```

Arguments

df	Data frame to guess value column from
----	---------------------------------------

Details

Strategy:

1. Is value or (all) column present? If so, use that
2. Otherwise, guess that last column is the value column

Author(s)

Hadley Wickham <h.wickham@gmail.com>

margin.vars	<i>Margin variables</i>
-------------	-------------------------

Description

Works out list of variables to margin over to get desired margins.

Usage

```
margin.vars(vars = list(NULL, NULL), margins = NULL)
```

Arguments

vars	column variables
margins	row variables
	vector of variable names to margin over.

Details

Variables that can't be margined over are dropped silently.

Author(s)

Hadley Wickham <h.wickham@gmail.com>

melt

Melt

Description

Melt an object into a form suitable for easy casting.

Usage

```
melt(data, ...)
```

Arguments

data	Data set to melt
...	Other arguments passed to the specific melt method

Details

This the generic melt function. See the following functions for specific details for different data structures:

- `melt.data.frame` for data.frames
- `melt.array` for arrays, matrices and tables
- `melt.list` for lists

Author(s)

Hadley Wickham <h.wickham@gmail.com>

melt.array

Melt an array

Description

This function melts a high-dimensional array into a form that you can use `cast` with.

Usage

```
melt.array(data, varnames = names(dimnames(data)), ...)
```

Arguments

data array to melt
varnames variable names to use in molten data.frame
...

Details

This code is conceptually similar to [as.data.frame.table](#)

Author(s)

Hadley Wickham <h.wickham@gmail.com>

Examples

```
a <- array(1:24, c(2,3,4))  
melt(a)  
melt(a, varnames=c("X", "Y", "Z"))  
dimnames(a) <- lapply(dim(a), function(x) LETTERS[1:x])  
melt(a)  
melt(a, varnames=c("X", "Y", "Z"))  
dimnames(a)[1] <- list(NULL)  
melt(a)
```

melt.cast_df *Melt cast data.frames*

Description

Melt the results of a cast

Usage

```
melt.cast_df(data, drop.margins=TRUE, ...)
```

Arguments

data
drop.margins
...

Details

This can be useful when performing complex aggregations - melting the result of a cast will do it's best to figure out the correct variables to use as id and measured.

Author(s)

Hadley Wickham <h.wickham@gmail.com>

melt.cast_matrix *Melt cast matrices*

Description

Melt the results of a cast

Usage

```
melt.cast_matrix(data, ...)
```

Arguments

data
...

Details

Converts to a data frame and then uses [melt.cast_df](#)

Author(s)

Hadley Wickham <h.wickham@gmail.com>

melt_check *Melt check.*

Description

Check that input variables to melt are appropriate.

Usage

```
melt_check(data, id.vars, measure.vars)
```

Arguments

data data frame
id.vars Vector of identifying variable names or indexes
measure.vars Vector of Measured variable names or indexes

Details

If id.vars or measure.vars are missing, melt_check will do its best to impute them. If you only supply one of id.vars and measure.vars, melt will assume the remainder of the variables in the data set belong to the other. If you supply neither, melt will assume character and factor variables are id variables, and all other are measured.

Value

id list id variable names

measure list of measured variable names

Author(s)

Hadley Wickham <h.wickham@gmail.com>

melt.data.frame *Melt a data frame*

Description

Melt a data frame into form suitable for easy casting.

Usage

```
melt.data.frame(data, id.vars, measure.vars, variable_name = "variable", na.rm = !p
```

Arguments

data	Data set to melt
id.vars	Id variables. If blank, will use all non measure.vars variables. Can be integer (variable position) or string (variable name)
measure.vars	Measured variables. If blank, will use all non id.vars variables. Can be integer (variable position) or string (variable name)
variable_name	Name of the variable that will store the names of the original variables
na.rm	Should NAs be removed from the data set?
preserve.na	Old argument name, now deprecated
...	

Details

You need to tell melt which of your variables are id variables, and which are measured variables. If you only supply one of `id.vars` and `measure.vars`, melt will assume the remainder of the variables in the data set belong to the other. If you supply neither, melt will assume factor and character variables are id variables, and all others are measured.

Value

molten data

Author(s)

Hadley Wickham <h.wickham@gmail.com>

See Also

<http://had.co.nz/reshape/>

Examples

```
head(melt(tips))
names(airquality) <- tolower(names(airquality))
melt(airquality, id=c("month", "day"))
names(ChickWeight) <- tolower(names(ChickWeight))
melt(ChickWeight, id=2:4)
```

melt.default

Default melt function

Description

For vectors, make a column of a data frame

Usage

```
melt.default(data, ...)
```

Arguments

data

...

Author(s)

Hadley Wickham <h.wickham@gmail.com>

`melt.list`*Melt a list*

Description

Melting a list recursively melts each component of the list and joins the results together

Usage

```
melt.list(data, ..., level=1)
```

Arguments

```
data  
...  
level
```

Author(s)

Hadley Wickham <h.wickham@gmail.com>

Examples

```
a <- as.list(1:4)  
melt(a)  
names(a) <- letters[1:4]  
melt(a)  
attr(a, "varname") <- "ID"  
melt(a)  
a <- list(matrix(1:4, ncol=2), matrix(1:6, ncol=2))  
melt(a)  
a <- list(matrix(1:4, ncol=2), array(1:27, c(3,3,3)))  
melt(a)  
melt(list(1:5, matrix(1:4, ncol=2)))  
melt(list(list(1:3), 1, list(as.list(3:4), as.list(1:2))))
```

`merge_all`*Merge all*

Description

Merge together a series of data.frames

Usage

```
merge_all(dfs, ...)
```

Arguments

dfs list of data frames to merge
...

Details

Order of data frames should be from most complete to least complete

Author(s)

Hadley Wickham <h.wickham@gmail.com>

See Also

[merge_recurse](#)

merge_recurse *Merge recursively*

Description

Recursively merge data frames

Usage

```
merge_recurse(dfs, ...)
```

Arguments

dfs list of data frames to merge
...

Author(s)

Hadley Wickham <h.wickham@gmail.com>

See Also

[merge_all](#)

namerows	<i>Name rows</i>
----------	------------------

Description

Add variable to data frame containing rownames

Usage

```
namerows(df, col.name = "id")
```

Arguments

df	data frame
col.name	name of new column containing rownames

Details

This is useful when the thing that you want to melt by is the rownames of the data frame, not an explicit variable

Author(s)

Hadley Wickham <h.wickham@gmail.com>

nested.by	<i>Nested.by function</i>
-----------	---------------------------

Description

Nest series of by statements returning nested list

Usage

```
nested.by(data, INDICES, FUN, ...)
```

Arguments

data
INDICES
FUN
...

Details

Work horse for producing cast lists.

Author(s)

Hadley Wickham <h.wickham@gmail.com>

nulldefault *Null default*

Description

Use default value when null

Usage

```
nulldefault(x, default)
```

Arguments

x
default

Details

Handy method when argument defaults aren't good enough.

Author(s)

Hadley Wickham <h.wickham@gmail.com>

prettyprint *Pretty print*

Description

Print reshaped data frame

Usage

```
prettyprint(x, digits=getOption("digits"), ..., colnames=TRUE)
```

Arguments

x
digits
...
colnames

Details

This will always work on the direct output from `cast`, but may not if you have manipulated (e.g. subsetted) the results.

Author(s)

Hadley Wickham <h.wickham@gmail.com>

`rdimnames`*Dimension names*

Description

These methods provide easy access to the special dimension names

Usage

```
rdimnames(x)
```

Arguments

`x`

Details

Reshape stores dimension names in a slightly different format to base R, to allow for (e.g.) multiple levels of column header. These accessor functions allow you to get and set them.

Author(s)

Hadley Wickham <h.wickham@gmail.com>

`recast`*Recast*

Description

[melt](#) and [cast](#) data in a single step

Usage

```
recast(data, formula, ..., id.var, measure.var)
```

Arguments

<code>data</code>	Data set to melt
<code>formula</code>	Casting formula, see cast for specifics
<code>...</code>	Other arguments passed to cast
<code>id.var</code>	Identifying variables. If blank, will use all non <code>measure.var</code> variables
<code>measure.var</code>	Measured variables. If blank, will use all non <code>id.var</code> variables

Details

This conveniently wraps melting and casting a data frame into one step.

Author(s)

Hadley Wickham <h.wickham@gmail.com>

See Also

<http://had.co.nz/reshape/>

Examples

```
recast(french_fries, time ~ variable, id.var=1:4)
```

rename	<i>Rename</i>
--------	---------------

Description

Rename an object

Usage

```
rename(x, replace)
```

Arguments

<code>x</code>	object to be renamed
<code>replace</code>	named vector specifying new names

Details

The `rename` function provide an easy way to rename the columns of a `data.frame` or the items in a list.

Author(s)

Hadley Wickham <h.wickham@gmail.com>

Examples

```
rename(mtcars, c(wt = "weight", cyl = "cylinders"))
a <- list(a = 1, b = 2, c = 3)
rename(a, c(b = "a", c = "b", a="c"))
```

rescaler

Rescaler

Description

Convenient methods for rescaling data

Usage

```
rescaler(x, type="sd", ...)
```

Arguments

x	object to rescale
type	type of rescaling to use (see description for details)
...	other options (only passed to rank)

Details

Provides methods for vectors, matrices and data.frames

Currently, five rescaling options are implemented:

- I: do nothing
- range: scale to [0, 1]
- rank: convert values to ranks
- robust: robust version of sd, subtract median and divide by median absolute deviation
- sd: subtract mean and divide by standard deviation

Author(s)

Hadley Wickham <h.wickham@gmail.com>

See Also

[rescaler.default](#)

```
rescaler.data.frame
```

Rescale a data frame

Description

Rescales data frame by columns

Usage

```
rescaler.data.frame(x, type="sd", ...)
```

Arguments

x	data.frame to rescale
type	type of rescaling to apply
...	other arguments passed to rescaler

Author(s)

Hadley Wickham <h.wickham@gmail.com>

```
rescaler.default
```

Default rescaler

Description

See [rescaler](#) for details

Usage

```
rescaler.default(x, type="sd", ...)
```

Arguments

x	vector to rescale
type	type of rescaling to apply
...	other arguments passed to rescaler

Author(s)

Hadley Wickham <h.wickham@gmail.com>

```
rescaler.matrix      Rescale a matrix
```

Description

Rescales matrix by columns

Usage

```
rescaler.matrix(x, type="sd", ...)
```

Arguments

x	matrix to rescale
type	type of rescaling to apply
...	other arguments passed to rescaler

Author(s)

Hadley Wickham <h.wickham@gmail.com>

```
reshape1      Casting workhorse.
```

Description

Takes data frame and variable list and casts data.

Usage

```
reshape1(data, vars = list(NULL, NULL), fun.aggregate=NULL, margins, df=FALSE, fill)
```

Arguments

data	data frame
vars	variables to appear in columns
fun.aggregate	variables to appear in rows
margins	aggregation function
df	should the aggregating function be supplied with the entire data frame, or just the relevant entries from the values column
fill	vector of variable names (can include "grand_col" and "grand_row") to compute margins for, or TRUE to computer all margins
add.missing	value with which to fill in structural missings
...	further arguments are passed to aggregating function

Author(s)

Hadley Wickham <h.wickham@gmail.com>

See Also

[cast](#)

Examples

```

ffm <- melt(french_fries, id=1:4, na.rm = TRUE)
# Casting lists -----
cast(ffm, treatment ~ rep | variable, mean)
cast(ffm, treatment ~ rep | subject, mean)
cast(ffm, treatment ~ rep | time, mean)
cast(ffm, treatment ~ rep | time + variable, mean)
names(airquality) <- tolower(names(airquality))
aqm <- melt(airquality, id=c("month", "day"), preserve=FALSE)
#Basic call
reshape1(aqm, list("month", NULL), mean)
reshape1(aqm, list("month", "variable"), mean)
reshape1(aqm, list("day", "month"), mean)

#Explore margins -----
reshape1(aqm, list("month", NULL), mean, "month")
reshape1(aqm, list("month", NULL) , mean, "grand_col")
reshape1(aqm, list("month", NULL) , mean, "grand_row")

reshape1(aqm, list(c("month", "day"), NULL), mean, "month")
reshape1(aqm, list(c("month"), "variable"), mean, "month")
reshape1(aqm, list(c("variable"), "month"), mean, "month")
reshape1(aqm, list(c("month"), "variable"), mean, c("month", "variable"))

reshape1(aqm, list(c("month"), "variable"), mean, c("grand_row"))
reshape1(aqm, list(c("month"), "variable"), mean, c("grand_col"))
reshape1(aqm, list(c("month"), "variable"), mean, c("grand_row", "grand_col"))

reshape1(aqm, list(c("variable", "day"), "month"), mean, c("variable"))
reshape1(aqm, list(c("variable", "day"), "month"), mean, c("variable", "grand_row"))
reshape1(aqm, list(c("month", "day"), "variable"), mean, "month")

# Multiple fnction returns -----
reshape1(aqm, list(c("month", "result_variable"), NULL), range)
reshape1(aqm, list(c("month"), "result_variable") , range)
reshape1(aqm, list(c("result_variable", "month"), NULL), range)

reshape1(aqm, list(c("month", "result_variable"), "variable"), range, "month")
reshape1(aqm, list(c("month", "result_variable"), "variable"), range, "variable")
reshape1(aqm, list(c("month", "result_variable"), "variable"), range, c("variable", "month"))
reshape1(aqm, list(c("month", "result_variable"), "variable"), range, c("grand_col"))
reshape1(aqm, list(c("month", "result_variable"), "variable"), range, c("grand_row"))

reshape1(aqm, list(c("month"), c("variable")), function(x) diff(range(x)))

```

round_any	<i>Round any</i>
-----------	------------------

Description

Round to multiple of any number

Usage

```
round_any(x, accuracy, f=round)
```

Arguments

x	numeric vector to round
accuracy	number to round to
f	function to use for round (eg. <code>floor</code>)

Details

Useful when you want to round a number to arbitrary precision

Author(s)

Hadley Wickham <h.wickham@gmail.com>

Examples

```
round_any(135, 10)
round_any(135, 100)
round_any(135, 25)
round_any(135, 10, floor)
round_any(135, 100, floor)
round_any(135, 25, floor)
round_any(135, 10, ceiling)
round_any(135, 100, ceiling)
round_any(135, 25, ceiling)
```

Smiths	<i>Demo data describing the Smiths</i>
--------	--

Description

A small demo dataset describing John and Mary Smith. Used in the introductory vignette.

Usage

```
data(smiths)
```

Format

A data frame with 2 rows and 5 variables

sort_df	<i>Sort data frame</i>
---------	------------------------

Description

Convenience method for sorting a data frame using the given variables.

Usage

```
sort_df(data, vars=names(data))
```

Arguments

data	data frame to sort
vars	variables to use for sorting

Details

Simple wrapper around order

Author(s)

Hadley Wickham <h.wickham@gmail.com>

`sparseby`*Apply a Function to a Data Frame split by levels of indices*

Description

Function `sparseby` is a modified version of `by` for `tapply` applied to data frames. It always returns a new data frame rather than a multi-way array.

Usage

```
sparseby(data, INDICES = list(), FUN, ..., GROUPNAMES = TRUE)
```

Arguments

<code>data</code>	an R object, normally a data frame, possibly a matrix.
<code>INDICES</code>	a variable or list of variables indicating the subgroups of <code>data</code>
<code>FUN</code>	a function to be applied to data frame subsets of <code>data</code> .
<code>...</code>	further arguments to <code>FUN</code> .
<code>GROUPNAMES</code>	a logical variable indicating whether the group names should be bound to the result

Details

A data frame or matrix is split by row into data frames or matrices respectively subsetted by the values of one or more factors, and function `FUN` is applied to each subset in turn.

`sparseby` is much faster and more memory efficient than `by` or `tapply` in the situation where the combinations of `INDICES` present in the data form a sparse subset of all possible combinations.

Value

A data frame or matrix containing the results of `FUN` applied to each subgroup of the matrix. The result depends on what is returned from `FUN`:

If `FUN` returns `NULL` on any subsets, those are dropped.

If it returns a single value or a vector of values, the length must be consistent across all subgroups. These will be returned as values in rows of the resulting data frame or matrix.

If it returns data frames or matrices, they must all have the same number of columns, and they will be bound with `rbind` into a single data frame or matrix.

Names for the columns will be taken from the names in the list of `INDICES` or from the results of `FUN`, as appropriate.

Author(s)

Duncan Murdoch

See Also[tapply](#), [by](#)**Examples**

```
x <- data.frame(index=c(rep(1,4),rep(2,3)),value=c(1:7))
x
sparseby(x,x$index,nrow)

# The version below works entirely in matrices
x <- as.matrix(x)
sparseby(x,list(group = x["index"]), function(subset) c(mean=mean(subset[,2])))
```

stamp

*Stamp***Description**

Stamp is like reshape but the "stamping" function is passed the entire data frame, instead of just a few variables.

Usage

```
stamp(data, formula = . ~ ., fun.aggregate, ..., margins=NULL, subset=TRUE, add.mis
```

Arguments

data	data.frame (no molten)
formula	formula that describes arrangement of result, columns ~ rows, see reshape for more information
fun.aggregate	aggregation function to use, should take a data frame as the first argument
...	arguments passed to the aggregation function
margins	margins to compute (character vector, or TRUE for all margins), can contain <code>grand_row</code> or <code>grand_col</code> to include grand row or column margins respectively.
subset	logical vector by which to subset the data frame, evaluated in the context of the data frame so you can
add.missing	

Details

It is very similar to the [by](#) function except in the form of the output which is arranged using the formula as in [reshape](#)

Note that it's very easy to create objects that R can't print with this function. You will probably want to save the results to a variable and then use `extract` the results. See the examples.

Author(s)

Hadley Wickham <h.wickham@gmail.com>

`str.cast_matrix` *Print cast objects*

Description

Printing methods

Usage

```
str.cast_matrix(object, ...)
```

Arguments

object
...

Details

Used for printing.

Author(s)

Hadley Wickham <h.wickham@gmail.com>

`strip.dups` *Strip duplicates.*

Description

Strips out duplicates from data.frame and replace them with blanks.

Usage

```
strip.dups(df)
```

Arguments

df data.frame to modify

Value

character matrix

Author(s)

Hadley Wickham <h.wickham@gmail.com>

tidystamp

Tidy up stamped data set

Description

@keyword internal

Usage

tidystamp(x)

Arguments

x

Author(s)

Hadley Wickham <h.wickham@gmail.com>

Tips

Tipping data

Description

One waiter recorded information about each tip he received over a period of a few months working in one restaurant. He collected several variables:

1. tip in dollars,
2. bill in dollars,
3. sex of the bill payer,
4. whether there were smokers in the party,
5. day of the week,
6. time of day,
7. size of the party.

In all he recorded 244 tips. The data was reported in a collection of case studies for business statistics (Bryant & Smith 1995).

Usage

data(tips)

Format

A data frame with 244 rows and 7 variables

References

Bryant, P. G. and Smith, M (1995) *Practical Data Analysis: Case Studies in Business Statistics*. Homewood, IL: Richard D. Irwin Publishing:

uniquedefault	<i>Unique default</i>
---------------	-----------------------

Description

Convenience function for setting default if not unique

Usage

```
uniquedefault(values, default)
```

Arguments

values	vector of values
default	default to use if values not uniquez

Details

Used by ggplot2

Author(s)

Hadley Wickham <h.wickham@gmail.com>

untable	<i>Untable a dataset</i>
---------	--------------------------

Description

Inverse of table

Usage

```
untable(df, num)
```

Arguments

df matrix or data.frame to untable
num vector of counts (of same length as df)

Details

Given a tabulated dataset (or matrix) this will untabulate it by repeating each row by the number of times it was repeated

Author(s)

Hadley Wickham <h.wickham@gmail.com>

updatelist *Update list*

Description

Update a list, but don't create new entries

Usage

```
updatelist(x, y)
```

Arguments

x list to be updated
y list with updated values

Details

Don't know what this is used for!

Author(s)

Hadley Wickham <h.wickham@gmail.com>

Index

*Topic **category**

sparseby, 35

*Topic **datasets**

French fries, 15

Smiths, 34

Tips, 38

*Topic **internal**

add.all.combinations, 1

add.missing.levels, 2

all.vars.character, 3

as.data.frame.cast_df, 3

as.data.frame.cast_matrix, 4

as.matrix.cast_df, 5

as.matrix.cast_matrix, 5

cast_matrix, 8

cast_parse_formula, 9

check_formula, 10

clean.vars, 10

compute.margins, 12

condense, 13

dim_names, 14

expand, 15

guess_value, 17

margin.vars, 17

melt.cast_df, 19

melt.cast_matrix, 20

melt.default, 22

melt.list, 23

melt_check, 20

merge_recurse, 24

nested.by, 25

nulldefault, 26

prettyprint, 26

rdimnames, 27

rescaler.data.frame, 30

rescaler.default, 30

rescaler.matrix, 31

reshape1, 31

round_any, 33

stamp, 36

str.cast_matrix, 37

strip.dups, 37

tidystamp, 38

updatelist, 40

*Topic **iteration**

sparseby, 35

*Topic **manip**

cast, 6

colsplit, 11

combine_factor, 11

condense, 13

condense.df, 12

expand, 15

expand.grid.df, 14

funstofun, 16

melt, 18

melt.array, 18

melt.data.frame, 21

merge_all, 23

namerows, 25

recast, 27

rename, 28

rescaler, 29

sort_df, 34

uniquedefault, 39

untable, 39

add.all.combinations, 1

add.missing.levels, 2

all.vars.character, 3

apply, 6

as.data.frame.cast_df, 3

as.data.frame.cast_matrix, 4

as.data.frame.table, 19

as.matrix.cast_df, 5

as.matrix.cast_matrix, 5

by, 35, 36

cast, 6, 18, 27, 28, 32
 cast_matrix, 8, 8
 cast_parse_formula, 9
 check_formula, 10
 clean.vars, 10
 colsplit, 11
 combine_factor, 11
 compute.margins, 12
 condense, 13
 condense.df, 12

 dim_names, 14

 expand, 15
 expand.grid.df, 14

 floor, 33
 French fries, 15
 french_fries (*French fries*), 15
 funstofun, 16

 guess_value, 17

 margin.vars, 17
 melt, 6, 18, 27
 melt.array, 18, 18
 melt.cast_df, 19, 20
 melt.cast_matrix, 20
 melt.data.frame, 18, 21
 melt.default, 22
 melt.list, 18, 23
 melt.matrix (*melt.array*), 18
 melt.table (*melt.array*), 18
 melt_check, 20
 merge_all, 23, 24
 merge_recurse, 24, 24

 namerows, 25
 nested.by, 25
 nulldefault, 26

 prettyprint, 26
 print.cast_df (*str.cast_matrix*),
 37
 print.cast_matrix
 (*str.cast_matrix*), 37

 rank, 29
 rbind, 35
 rcolnames (*rdimnames*), 27

 rcolnames<- (*rdimnames*), 27
 rdimnames, 27
 rdimnames<- (*rdimnames*), 27
 recast, 6, 27
 rename, 28
 rescaler, 29, 30
 rescaler.data.frame, 30
 rescaler.default, 29, 30
 rescaler.matrix, 31
 reshape, 36
 reshape1, 7, 31
 round_any, 33
 rrownames (*rdimnames*), 27
 rrownames<- (*rdimnames*), 27

 Smiths, 34
 smiths (*Smiths*), 34
 sort_df, 34
 sparseby, 35
 stamp, 36
 str.cast_df (*str.cast_matrix*), 37
 str.cast_matrix, 37
 strip.dups, 37
 sweep, 6

 tapply, 35, 36
 tidystamp, 38
 Tips, 38
 tips (*Tips*), 38

 unquedefault, 39
 untable, 39
 updatelist, 40