

# Package ‘reshape’

January 2, 2012

**Type** Package

**Title** Flexibly reshape data.

**Version** 0.8.4

**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

**Repository** CRAN

**Date/Publication** 2011-01-28 20:02:12

## R topics documented:

cast . . . . .	2
colsplit . . . . .	4
combine_factor . . . . .	5
condense.df . . . . .	5
expand.grid.df . . . . .	6
French fries . . . . .	7
funstofun . . . . .	7
melt . . . . .	8
melt.array . . . . .	9
melt.data.frame . . . . .	9
merge_all . . . . .	11
namerows . . . . .	11

recast . . . . .	12
rename . . . . .	13
rescaler . . . . .	14
Smiths . . . . .	15
sort_df . . . . .	15
sparseby . . . . .	16
stamp . . . . .	17
Tips . . . . .	18
uniquedefault . . . . .	18
untable . . . . .	19
<b>Index</b>	<b>20</b>

---

cast	<i>Cast function</i>
------	----------------------

---

## 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=TRUE, df=FALSE, fill)
```

## Arguments

data	molten data frame, see <a href="#">melt</a>
formula	casting formula, see details for specifics
fun.aggregate	aggregation function
add.missing	fill in missing combinations?
value	name of value column
...	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

## 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")

```

---

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

x	character vector or factor to split up
split	regular expression to split on
names	names for output columns

## Author(s)

Hadley Wickham <h.wickham@gmail.com>

---

combine_factor	<i>Combine factor levels</i>
----------------	------------------------------

---

**Description**

Convenience function to make it easy to combine multiple levels

**Usage**

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

**Arguments**

fac	factor variable
variable	either a vector of . See examples for more details.
other.label	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]))
```

---

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

---

**Description**

Condense

**Usage**

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

**Arguments**

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

**Author(s)**

Hadley Wickham <h.wickham@gmail.com>

---

expand.grid.df	<i>Expand grid</i>
----------------	--------------------

---

**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")))
```

---

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))
```

---

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	<i>Melt an array</i>
------------	----------------------

---

**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**

<code>data</code>	array to melt
<code>varnames</code>	variable names to use in molten data.frame
<code>...</code>	other arguments ignored

**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.data.frame	<i>Melt a data frame</i>
-----------------	--------------------------

---

**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 = !preserve.na, preserv
```

**Arguments**

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

**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)
```

---

merge_all	<i>Merge all</i>
-----------	------------------

---

**Description**

Merge together a series of data.frames

**Usage**

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

**Arguments**

dfs	list of data frames to merge
...	other arguments passed on 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](#)

---

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>

---

recast

*Recast*

---

**Description**

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

**Usage**

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

**Arguments**

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

x	object to be renamed
replace	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"))

# Example supplied by Timothy Bates
names <- c("john", "tim", "andy")
ages <- c(50, 46, 25)
mydata <- data.frame(names,ages)
names(mydata) #-> "name", "ages"

# lets change "ages" to singular.
# nb: The operation is not done in place, so you need to set your
# data to that returned from rename

mydata <- rename(mydata, c(ages="age"))
names(mydata) #-> "name", "age"
```

---

`rescaler`*Rescaler*

---

### Description

Convenient methods for rescaling data

### Usage

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

### Arguments

<code>x</code>	object to rescale
<code>type</code>	type of rescaling to use (see description for details)
<code>...</code>	other options (only passed to <a href="#">rank</a> )

### 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](mailto:h.wickham@gmail.com)>

### See Also

[rescaler.default](#)

---

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 data
<code>FUN</code>	a function to be applied to data frame subsets of data.
<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.missing=FALSE)
```

**Arguments**

data	data.frame (no molten)
formula	formula that describes arrangement of result, columns ~ rows, see <a href="#">reshape</a> 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 grand_row or grand_col 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	fill in missing combinations?

**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>

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:

- tip in dollars,
- bill in dollars,
- sex of the bill payer,
- whether there were smokers in the party,
- day of the week,
- time of day,
- 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*Unique default*

---

**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                      *Untable a dataset*

---

**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>

# Index

- \*Topic **category**
  - [sparseby](#), [16](#)
- \*Topic **datasets**
  - [French fries](#), [7](#)
  - [Smiths](#), [15](#)
  - [Tips](#), [18](#)
- \*Topic **iteration**
  - [sparseby](#), [16](#)
- \*Topic **manip**
  - [cast](#), [2](#)
  - [colsplit](#), [4](#)
  - [combine\\_factor](#), [5](#)
  - [condense.df](#), [5](#)
  - [expand.grid.df](#), [6](#)
  - [funstofun](#), [7](#)
  - [melt](#), [8](#)
  - [melt.array](#), [9](#)
  - [melt.data.frame](#), [9](#)
  - [merge\\_all](#), [11](#)
  - [namerows](#), [11](#)
  - [recast](#), [12](#)
  - [rename](#), [13](#)
  - [rescaler](#), [14](#)
  - [sort\\_df](#), [15](#)
  - [stamp](#), [17](#)
  - [uniquedefault](#), [18](#)
  - [untable](#), [19](#)
- [apply](#), [3](#)
- [as.data.frame.table](#), [9](#)
- [by](#), [16](#), [17](#)
- [cast](#), [2](#), [9](#), [12](#)
- [colsplit](#), [4](#)
- [combine\\_factor](#), [5](#)
- [condense.df](#), [5](#)
- [expand.grid.df](#), [6](#)
- [French fries](#), [7](#)
- [french\\_fries \(French fries\)](#), [7](#)
- [funstofun](#), [7](#)
- [melt](#), [2](#), [3](#), [8](#), [12](#)
- [melt.array](#), [8](#), [9](#)
- [melt.data.frame](#), [8](#), [9](#)
- [melt.list](#), [8](#)
- [melt.matrix \(melt.array\)](#), [9](#)
- [melt.table \(melt.array\)](#), [9](#)
- [merge\\_all](#), [11](#)
- [merge\\_recurse](#), [11](#)
- [namerows](#), [11](#)
- [rank](#), [14](#)
- [rbind](#), [16](#)
- [recast](#), [3](#), [12](#)
- [rename](#), [13](#)
- [rescaler](#), [14](#)
- [rescaler.default](#), [14](#)
- [reshape](#), [17](#)
- [reshape1](#), [3](#)
- [Smiths](#), [15](#)
- [smiths \(Smiths\)](#), [15](#)
- [sort\\_df](#), [15](#)
- [sparseby](#), [16](#)
- [stamp](#), [17](#)
- [sweep](#), [3](#)
- [tapply](#), [16](#), [17](#)
- [Tips](#), [18](#)
- [tips \(Tips\)](#), [18](#)
- [uniquedefault](#), [18](#)
- [untable](#), [19](#)