Package 'multitable'

July 2, 2014

Type Package

Title Simultaneous manipulation of multiple arrays of data, with data.list objects

Version 1.6

Date 2013-12-12

Author Steve C Walker

Maintainer Steve C Walker <steve.walker@utoronto.ca>

Description Data frames are integral to R. They provide a standard format for passing data to model-fitting and plotting functions, and this standard makes it easier for experienced users to learn new functions that accept data as a single data frame. Still, many data sets do not easily fit into a single data frame; data sets in ecology with a so-called fourth-corner problem provide important examples. Manipulating such inherently multiple-table data using several data frames can result in long and difficult-to-read work-flows. This package provides new data storage objects called data.list objects, which extend the data.frame concept to explicitly multiple-table settings. Like data frames, data lists are lists of variables stored as vectors; what is new is that these vectors have dimension attributes that make accessing and manipulating them easier. As data.list objects can be coerced to data.frame objects, they can be used with all R functions that accept an object that can be coerced to a data.frame.

Suggests MASS, lattice, testthat, arm, ggplot2, rbenchmark, scales,vegan

License GPL-2

URL http://multitable.r-forge.r-project.org/

LazyLoad yes

BuildVignettes true

NeedsCompilation no

Repository CRAN

Date/Publication 2013-12-12 19:05:18

R topics documented:

`multitable-package` *Simultaneous manipulation of multiple arrays of data, with data.list objects*

Description

Data frames are integral to R. They provide a standard format for passing data to model-fitting and plotting functions, and this standard makes it easier for experienced users to learn new functions that accept data as a single data frame. Still, many data sets do not easily fit into a single data frame; data sets in ecology with a so-called fourth-corner problem provide important examples. Manipulating such inherently multiple-table data using several data frames can result in long and difficult-to-read workflows. We introduce the R **multitable** package to provide new data storage objects called `data.list` objects, which extend the `data.frame` concept to explicitly multiple-table settings. Like data frames, data lists are lists of variables stored as vectors; what is new is that these vectors have dimension attributes that make accessing and manipulating them easier. As

data.list objects can be coerced to data.frame objects, they can be used with all R functions that accept an object that is coercible to a data.frame.

Details

Package:	multitable
Type:	Package
Version:	1.5
Date:	2012-11-09
Suggests:	MASS, lattice, testthat, arm, ggplot2, rbenchmark, scales, vegan
License:	GPL-2
URL:	http://multitable.r-forge.r-project.org/
LazyLoad:	yes

Note

Using **multitable** DOES NOT REQUIRE ANY OTHER PACKAGES, other than those that typically come with R. The suggested packages are only for examples in the help files and vignette, and for package development (i.e. **testthat**).

Author(s)

Maintainer: Steve Walker <steve.walker@utoronto.ca>

References

Steven C Walker, Guillaume Guenard, Peter Solymos, Pierre Legendre (2012). Multiple-Table Data in R with the multitable Package. Journal of Statistical Software, 51(8), 1-38. URL http://www.jstatsoft.org/v51/i08/

Examples

```
#######################################################
# The package vignette (Walker et al. 2012 JSS) is a
# useful place to start
#######################################################
vignette("multitable")


#######################################################
# The structure of data lists
#######################################################

# load the example data set in data list form.
data(fake.community)
fake.community
```

```
# print a summary of the relational structure
# of the data set.
summary(fake.community)

#####################################################
# Subscripting data lists
#####################################################

# extract two years of data.
fake.community[, c("2008", "2009"), ]

# extraction using both numerical and character
# vectors.
fake.community[1:3, "1537", 1]

# subscripting data lists is designed to be as
# intuitive as possible to R users.  above are
# the examples covered in the manuscript, but
# see the help file for more examples and
# explanation.
?`[.data.list`

#####################################################
# Transforming variables in data lists
#####################################################

# transformation occurs much like it would with
# data frames.
fake.community$abundance <- log1p(fake.community$abundance)
fake.community$temperature[, "1537"] <- c(5, 10, 30, 20, -80, -10)
fake.community$precipitation[, "1537"] <- c(5, 50, 75, 50, 2, 7)
fake.community$body.size["moss"] <- 1
fake.community

#####################################################
# Simple analysis functions
#####################################################

# we can pass data lists to lm just as we would pass
# data frames.
lm(abundance ~ body.size*temperature, data = fake.community)
lm(abundance ~ homeotherm*temperature, data = fake.community)

# this works for any function that tries to coerce
# data to a data frame, such as the robust linear
# model function from MASS.
library("MASS")
rlm(abundance ~ body.size*temperature, data = fake.community)

#####################################################
# Coercing data lists to data frames
#####################################################
```

```
# data frames are easily coerced to data frames via
# the as.data.frame method for data.list objects.
fake.community.df <- as.data.frame(fake.community)
fake.community.df[, -6]

# therefore, data list objects can easily be passed to
# any R function accepting a data frame, after they
# have been converted to a data frame.
library(lattice)
xyplot(abundance ~ temperature | body.size, data = fake.community.df)

# for further information about coercing in multitable:
?as.data.list

#######################################################
# How data lists are made
#######################################################

# here are three example objects to be combined into
# a data list.
abundance <- data.frame(
sites=c(
"midlatitude", "subtropical", "tropical", "equatorial",
"arctic", "midlatitude", "tropical", "equatorial",
"subtropical"
),
species=c(rep("capybara", 4),rep("moss", 4), "vampire"),
abundance=c(4, 10, 8, 7, 5, 6, 9, 3, 1)
)
environment <- data.frame(
sites=c(
"subarctic", "midlatitude", "subtropical",
"tropical", "equatorial"
),
temperature=c(0, 10, 20, 50, 30),
precipitation=c(40, 20, 100, 150, 200)
)
trait <- data.frame(
species=c("capybara", "moss", "vampire"),
body.size=c(140, 5, 190),
metabolic.rate=c(20, 5, 0)
)
abundance
environment
trait

# we use the dlcast function to combine them.
# the dimids argument tells dlcast what dimensions
# (or columns as they are in 'long' format) are
# shared among tables.  the fill argument tells
# dlcast how to fill in any structural missing
# values.
dl <- dlcast(list(abundance, environment, trait),
```

```
dimids=c("sites", "species"),
fill=c(0, NA, NA)
)
dl

# for other ways to create data list objects, see:
?data.list
?as.data.list
?read.multitable
?variable
```

`aperm.data.list` *Data list transposition*

Description

Transpose a data list by permuting its dimensions.

Usage

```
## S3 method for class 'data.list'
aperm(a, perm, ...)

## S3 method for class 'data.list'
t(x)
```

Arguments

`a`	a data list to be transposed.
`x`	a data list to be transposed.
`perm`	the subscript permutation vector (see `aperm`).
`...`	not currently used.

Value

A transposed version of the data list, `a`, with the replication dimensions permuted as indicated by `perm`.

Note

Does not allow `resize = FALSE`, as can be done in the default method (`aperm`).

Currently `aperm.data.list` is implemented by converting the data list to a list, transposing the benchmark variable (see `data.list` for a description of the benchmark concept), and creating a new data list object. This works because the `as.data.list` function automatically transposes the other variables such that their dimensions are in the same order as the benchmark. However, this implementation is perhaps not as efficient as it could be and may be changed in the future.

`t.data.list` is equivalent to `aperm.data.list` with the `perm` argument `missing`, which results in a data list for which the order of subscripts is reversed (as in the default method, `aperm`).

See Also

aperm, t

---

as.data.list *Coercion to and from data lists*

Description

These functions coerce lists and data frame objects to data lists and vice versa.

Usage

```
as.data.list(x, ...)

is.data.list(x)

is.4th.corner(x)

data.list.mold(x)

## Default S3 method:
as.data.list(x, dimids, match.dimids,
check = TRUE, drop = TRUE,...)

## S3 method for class 'data.list'
as.data.frame(x, row.names = NULL,
optional = FALSE, scheme = "repeat", mold, ...)

## S3 method for class 'data.list'
as.matrix(x, ...)

## S3 method for class 'data.list'
as.list(x, drop.attr = TRUE,
factorsTOstrings = FALSE, ...)
```

Arguments

x	An object. Either a data.list, data.frame, or list.
mold	A list typically obtained as output from data.list.mold. If missing, then mold is automatically created. See details.
dimids	See data.list for an explanation.
match.dimids	See data.list for an explanation.
check	See data.list for an explanation.
drop	If TRUE, single dimension data lists are coerced to data frames (i.e. their replication dimensions are 'dropped').

row.names	Passed to the default method after some processing.
optional	Passed to the default method after some processing.
scheme	Type of coercion. Currently, only "repeat" is allowed but others may follow.
drop.attr	Should the attributes of the data list be dropped?
factorsTOstrings	Should factors be converted to character strings?
...	Additional arguments (passed to the default method in as.data.frame.data.list).

Details

Ultimately, all functions that create data.list objects (i.e. data.list; dlcast; read.multitable; variable; variableGroup) are wrappers for as.data.list. From a practical perspective the as.data.list function is very similar to data.list, with the following principal differences:

- The as.data.list function requires that multiple objects be organised in a list (x) whereas data.list uses a ... argument.
- Like data.frame, data.list automatically names variables as character versions of the names passed to ..., whereas variables must be named explicitly with as.data.list.

The is.data.list function tests if an object inherits from class data.list.

The is.4th.corner function tests if an object is both a data list and has all the properties of a data set with a fourth corner problem.

By default, as.data.frame.data.list works in two steps. First, a mold of the coercion process is created with data.list.mold. The mold is a list of vectors, each with length equal to the number of rows in the coerced data frame. Each vector is associated with a particular variable in x and contains indices that point to the elements of that variable. Second, the mold is 'filled with' (i.e. used to subscript) x thereby producing the coerced data frame.

If data lists are to be iteratively coerced to data frames (e.g. repeated random subscripting and coercion), then speed can be enhanced by creating a mold explicitly with data.list.mold and then passing it to as.data.frame.data.list through the mold argument. Because the mold will be the same for any data list of the same 'shape', the mold only needs to be created once before the iterations thereby eliminating the need to iteratively create the mold.

The as.matrix method for data lists returns either a single matrix (if all variables are numeric or all variables are either factors or character); otherwise, two matrices are returned, one with the numeric and one with the factor variables respectively. The matrices are returned by first coercing to data frames and then to matrices. This method was created as a result of a suggestion by Philip Dixon.

The as.list.data.list function is equivalent to unclass if drop.attr = FALSE, otherwise (i.e. drop.attr = TRUE) the "match.dimids", "bm", and "repdim" attributes are dropped from x and the "subsetdim" attributes are dropped from each variable in x.

Value

The coerced object.

See Also

as.data.frame; as.list; as.matrix; variablize; data.list; dlcast; read.multitable.

Examples

```
data(fake.community)
fake.community

## testing for data list
is.data.list(fake.community) # TRUE
is.data.list(fake.community$abundance) # FALSE
is.array(fake.community$abundance) # TRUE

## this example illustrates the main difference between
## \code{data.list} and \code{as.data.list}. compare with
## the similar example in \code{\link{data.list}}. notice
## the naming problems with the latter method.
a1 <- matrix(runif(50), 10, 5)
a2 <- matrix(runif(50), 10, 5)
a3 <- matrix(runif(50), 10, 5)
a <- list(a1, a2, a3)
b <- runif(10)
c <- letters[1:5]
data.list(a, b, c)
as.data.list(list(a, b, c))

## coercing to a data frame with explicit mold creation
fake.mold <- data.list.mold(fake.community)
fake.mold
as.data.frame(fake.community)
as.data.frame(fake.community, mold = fake.mold)

## we can coerce a similar data list faster using this
## \code{fake.mold} object
fc.rnd <- fake.community
fc.rnd$abundance <- fc.rnd$abundance[sample(6), , ]
as.data.frame(fc.rnd,mold = fake.mold)

## unless drop=FALSE, simple data lists are immediately
## coerced into data frames -- simple data don't require
## data lists
x <- runif(10)
as.data.list(x)
as.data.list(x, drop = FALSE)

## different ways to coerce a data list to a list.
## each way can produce slightly different results.
as.list(fake.community)
as.list(fake.community, drop.attr = FALSE)
unclass(fake.community)
lapply(fake.community, I)

is.4th.corner(fake.community) ## FALSE
dl <- dropdl(fake.community[,1,])
is.4th.corner(dl) ## TRUE
is.4th.corner(dl[1:3]) ## FALSE
```

bci *Barro Colorado Island Data List*

Description

Abundances and functional traits of tree species on Barro Colorado Island.

Usage

```
data(bci)
```

Note

This data list object is mainly for illustration purposes and should probably not be used in serious scientific studies, because only species found in both the traits data set and the abundance data set are included.

Source

Traits are from: Wright et. al (2010) Functional traits and the growth-mortality trade-off in tropical trees. Ecology. 91: 3664-3674.

Abundances are from: Condit et. al (2002) Beta-diversity in tropical forest trees. Science 295: 666-669.

bm *Data list benchmark*

Description

Extract the 'benchmark' variable from a data list.

Usage

```
bm(x)
```

Arguments

x	A data list.

Details

When data.list, as.data.list, dlcast, or read.multitable is used to create a new data list object, one of the variables in this object is designated to be the 'benchmark' variable. The first variable passed to data.list that is replicated along every dimension is designated the benchmark. The "bm" attribute in a data.list object gives the subscript of the benchmark variable. Thus, the bm function is a single line: x[[attr(x, "bm")]].

Although any variable that is replicated along every dimension can serve as the benchmark, it is recommended that the benchmark be a response variable if such a variable exists. Therefore, such a response variable should be the first variable passed to data.list.

The purpose of the benchmark is to provide a frame of reference for other less-replicated variables in a data list and is a prototype of the replication in a data list as a whole. In this way, one may access the names of the dimensions of the data list simply by accessing the names of the dimensions of the benchmark. Indeed, this is how dimnames.data.list works.

Value

The benchmark variable.

See Also

data.list; as.data.list

Examples

```
data(fake.community)
bm(fake.community)
```

---

data.list *Create a data list*

Description

Function for creating a data.list object from vectors, matrices, arrays, data frames, lists, and information on how these objects are related.

Usage

```
data.list(..., dimids, match.dimids, check = TRUE,
drop = TRUE, unique = TRUE)
```

Arguments

...	A comma-separated collection of vectors, matrices, arrays, data frames, and lists, containing the variables that will comprise the resulting data list.

`dimids` An optional vector of character strings giving identifiers for the replication dimensions of the data list. If `missing`, these identifiers are either set to `c("D1","D2",...)` or determined from information passed to the `match.dimids` argument if it is not `missing`.

`match.dimids` A (possibly named) list of character vectors, each associated with the elements of `...`, giving identifiers for the replication dimensions of these elements. The only replication dimension of a data frame is along the rows, as columns of data frames are taken to be variables. Data frames are split into their component vectors in the returned data list. If missing, it is determined automatically if possible. See details on how automatic dimension matching is done.

`check` If TRUE, the structure of the created data list is checked for consistency.

`drop` If TRUE, single dimension data lists are coerced to data frames (i.e. their replication dimensions are 'dropped').

`unique` If TRUE, variable names are forced to be unique via `make.names`.

Details

This function creates data lists, which are multiple-table extensions of `data.frame`s. With the `data.frame` function, a collection of vectors (of identical length) containing data are combined into a single object that can be passed to model-fitting and plotting functions. In contrast, the `data.list` function allows not just vectors of the same length in the collection but matrices and arrays with possibly different dimensions.

The `data.list` function creates objects of class `data.list`, which are collections of variables (i.e. vectors, matrices, and arrays). These variables are related because they share dimensions of replication. For example, a `matrix`-valued variable might share its first dimension with the only dimension of a `vector`-valued variable. See `vignette("multitable")` for more information on the structure of data lists.

The `...` argument accepts a collection of vectors, matrices, arrays, data frames, and lists to be converted to a data list. These different types of objects are used by `data.list` in different ways:

`vector` Becomes a variable in the resulting data list with a single dimension of replication. In particular, a `vector` without a dimension attribute is converted to a one-dimensional `array`.

`matrix` Becomes a variable in the resulting data list with two dimensions of replication.

`array` Becomes a variable in the resulting data list with the same number of dimensions as the array itself.

`data.frame` Each column becomes a variable with a single dimension.

`list` Each element becomes a variable. It is required that each element be either a `vector`, `matrix`, or `array`, and that they all have the same value for their dimension attributes.

The pattern of dimension sharing between the variables is either determined automatically (if `match.dimids` is missing) or supplied by passing a list via the `match.dimids` argument. Automatic dimension matching proceeds in two steps. First, `data.list` tries to deduce the pattern of dimension matching through the names of the dimensions of the objects passed to `...`. Different names are used for the different types of objects:

`vector` The `names` attribute.

`matrix` **or** `array` The `dimnames` attribute.

`data.frame` The `row.names` attribute.

`list` Either the `names` or `dimnames` attribute, depending on which one its elements posses.

For example, if the `names` attribute of a `vector` is identical to the first element in the `dimnames` attribute of an `array`, then the single dimension of this `vector` is matched with the first dimension of this `array`. Dimension matching by naming is the recommended method, because it requires thought about the relationships between the variables and therefore ensures that the structure of the data are well-understood.

If dimension matching via the names of the dimensions fails, then `data.list` tries to infer the pattern of matching by the sizes of the dimensions of its variables. This method will fail if (1) any object has two or more dimensions of the same size AND (2) at least one other object also has at least one other dimension of the same size. In the case of failure, an error message is reported suggesting that either the dimensions of the variables be named or that explicit dimension matching be supplied as a `list` via the `match.dimids` argument.

Each element of the list passed to `match.dimids` (i.e. match dimension identifiers) is associated with one of the objects in the collection (e.g. the first element corresponds to the first object in the collection). In particular, the elements in `match.dimids` specify which dimensions the associated objects are replicated along. Each element in `match.dimids` should consist of a vector of character strings identifying the dimensions in the corresponding object in `....` Dimensions in different objects will be considered shared if they share the same identifier passed to `match.dimids`. The specification of dimension identifiers depends on the associated type of object passed to `...`:

`vector` A single string identifying the only dimension.

`matrix` A length-2 character vector identifying the first and second dimensions.

`array` A length-n character vector identifying the n dimensions.

`data.frame` A single string identifying the dimension associated with the rows. Each column is given the same dimension identifier.

`list` A length-n character vector identifying the n dimensions of the elements of the list. Each element is given the same set of dimension identifiers.

To form a valid data list, at least one of the objects in `...` must be replicated along all dimensions.

During the production of a data list, one variable is singled out as the 'benchmark' variable. See `bm` for further details on the benchmark concept. Note that the dimensions of each variable are permuted such that their order matches that of the benchmark.

Value

A `data.list` object which is a list with one element for each variable passed via `...` (note that each column in a data frame is treated as a separate variable, as is each element in a list). Each variable is given a "subsetdim" attribute, which is a logical vector with each element corresponding to one of the dimensions in the benchmark variable. TRUE elements specify that this variable is replicated along the corresponding dimension, and FALSE indicates otherwise. The `data.list` object itself also contains the following attributes:

`names` Names of the variables

match.dimids	A list of vectors giving the names of the dimensions of replication for each variable (one vector per variable).
bm	The index of the benchmark variable (see bm)
repdim	The replication dimensions (equal to the dimension attribute of the benchmark variable)

Note

The data.list function is largely a wrapper for as.data.list.default that lets objects be combined into a data list via a ... argument, as is done in data.frame.

See Also

as.data.frame.data.list for coercing to a data frame, Extract.data.list for subscripting the multiple tables in a data list simultaneously, and dim.data.list, dimnames.data.list, nvar, varnames, and print.data.list for other methods for data.list objects. If your data are originally in (database-like) 'long' format data frames, then use dlcast for creating data lists. If your data are originally in text files, use read.multitable.

Examples

```
## Automatic dimension matching by the sizes of dimensions.
## Note that this example would not work if all of the 10's were
## changed to 5's.  This example also illustrates how to pass
## several variables through one list, as long as each variable
## shares the same dimensions (in this case 10-by-5 matrices).
## We also see here how \code{data.list} automatically converts
## character vectors to factors.
a1 <- matrix(runif(50), 10, 5)
a2 <- matrix(runif(50), 10, 5)
a3 <- matrix(runif(50), 10, 5)
a <- list(a1, a2, a3)
b <- runif(10)
c <- letters[1:5]
data.list(a, b, c)

## Here we illustrate the use of dimension matching by
## dimension naming.
a <- lapply(a, `dimnames<-`, list(letters[1:10], LETTERS[1:5]))
names(b) <- letters[1:10]
names(c) <- LETTERS[1:5]
data.list(a, b, c)

## If we want to name the dimension identifiers themselves
## we can use \code{dimids}.
data.list(a, b, c, dimids = c("small letters", "large letters"))

## Or we could explicitly specify the pattern of dimension
## sharing using \code{match.dimids}.
md <- list(
c("small letters", "large letters"),
```

```
"small letters",
"large letters")
data.list(a, b, c, match.dimids = md)
```

`Data_list_arithmetic` *Data list arithmetic*

Description

Operators for adding (subtracting) the variables in one data list to (from) the variables in another data list.

Usage

```
## S3 method for class 'data.list'
Ops(e1, e2)
```

Arguments

`e1`	A data list
`e2`	Another data list

Value

Yet another data list

See Also

Useful with `variable` and `variableGroup`.

Examples

```
variable(matrix(runif(15), 5, 3), c("n", "m"), "A") +
variable(letters[1:3], "m", "B")

data(fake.community)
fake.community + variable(c(10, 0, 3), "species", "cuteness")
```

`dim.data.list` *Replication dimensions of a data list*

Description

Retrieve or set the replication dimensions of a data list object.

Usage

```
repdim(x)

## S3 method for class 'data.list'
dim(x)

## S3 replacement method for class 'data.list'
dim(x) <- value
```

Arguments

`x`	A data list.
`value`	Any value for `value` will result in an error (see details).

Details

For extraction, `repdim` and `dim.data.list` are identical.

Currently, `dim<-.data.list` simply returns an error because in the vast majority of cases it will not make sense to set the replication dimensions in this way.

Value

The `repdim` attribute of `x` (see `data.list` for information on this attribute).

See Also

`dim`; `dimnames.data.list`.

Examples

```
data(fake.community)
dim(fake.community)
repdim(fake.community)
```

dimnames.data.list *Dimnames of a data list object*

Description

Retrieve or set the dimnames of a data list object.

Usage

```
## S3 method for class 'data.list'
dimnames(x)
## S3 replacement method for class 'data.list'
dimnames(x) <- value
```

Arguments

x	A data.list object.
value	A list with each element containing a character vector with the names of the dimensions of the data list, x.

See Also

dimnames for the default method and dim.data.list to retrieve and set the dimensions of a data list.

Examples

```
data(fake.community)
fake.community

dimnames(fake.community)
dimnames(fake.community)[[1]] <- letters[1:6]
dimnames(fake.community)
dimnames(fake.community) <- list(
letters[7:12], letters[13:15], letters[16:18])
dimnames(fake.community)
```

dims_to_vars *Dimensions of replication to variables*

Description

Creates a new data list from an old one that includes categorical variables indicating replicates along specified dimensions of replication.

Usage

```
dims_to_vars(dl, dimids)
```

Arguments

dl	A data list
dimids	A character vector containing the identifier(s) for the dimension(s) that are to be used to create the new variable(s). If missing, all of the dimensions are used.

Value

A data list.

Examples

```
data(fake.community)

fc0 <- fake.community
fc1 <- dims_to_vars(fake.community)
fc2 <- dims_to_vars(fake.community, "years")

summary(fc0)
summary(fc1)
summary(fc2)
```

---

dlapply *Apply Functions Over Data List Margins*

Description

Returns a data list (or list of data lists) obtained by attempting to apply a function to margins of each variable in a data list.

Usage

```
dlapply(X, MARGIN, FUN, ...)

sdlapply(X, MARGIN, FUN, simplify = TRUE, ...)

variable_margins(X, MARGIN)
```

Arguments

X	A data list
MARGIN	See apply.
FUN	See apply.
simplify	Should the result be simplified to a data frame?
...	Optional arguments to FUN.

Details

Essentially, dlapply works by passing each variable in X to apply. For each variable, the dimensions identified by MARGIN are only used if that variable is replicated along that dimension.

Because data lists may contain very different variables, it is not guaranteed that FUN will generate anything useful for each variable in X. If an error occurs during a call to apply, that variable is simply removed from the output with a message. Variables may also be removed (also with a message) because they are not replicated along any of the dimensions specified by MARGIN.

FUN may induce a new dimension to the data list because its return value has length greater than one. In this case, the new dimension is called as.character(substitute(FUN)). However, if the length of this new dimension differs among variables, then a data list cannot be constructed and instead a list of the variables (in data list form) is returned (with a message).

The variable_margins function returns a list with one vector for each variable, giving the value for MARGIN when passing X to apply. This is helpful when figuring out why dlcast results in an error.

Value

A data list or list of data lists.

Examples

```
data(fake.community)
dlapply(fake.community, 1, quantile, na.rm = TRUE)
dlapply(fake.community, c(1,3), quantile, na.rm = TRUE)
```

---

dlcast *Cast data lists*

Description

Cast a list of molten (i.e. long-format) (i.e. database-like) data frames into a data list.

Usage

```
dlcast(x, dimids, fill, placeholders, ...)
```

Arguments

x	A list of long-format data frames
dimids	See data.list. If missing, a reasonable guess is made, which is the column names shared by all data frames in x.
fill	A vector the same length as x, giving the value to use for structural missing values for each of the data frames in x.
placeholders	Character vector giving elements of the dimids columns that are to be omitted.
...	Additional arguments to pass to as.data.list and data.list.

Details

This is one of several ways to create `data.list` objects. It is useful when the data are stored as several long-format `data.frame` objects. For the purposes of `dlcast`, a long-format data frame is a data frame with one named column for each variable, one named column for each dimension of replication (called `idvars` in the `reshape2` package), and one row for each (possibly multivariate) sample. Several data frames will allow variables to differ in their dimensions of replication. In order to combine several data frames into a valid data list, at least one data frame must contain all of the dimensions of replication in all of the data frames. Each data frame column that corresponds to a dimension of replication must be coercible to `factor`.

Value

A data list.

Note

This function is inspired by the `acast` function in the `reshape2` package by Hadley Wickham. Levi Waldron was a great help during the writing of `dlcast`. The `placeholders` argument arose out of a suggestion of a reviewer.

See Also

For other ways to create `data.list` objects: `data.list`; `as.data.list`; `read.multitable`; `variable`. For an approximate inverse of `dlcast`, see `dlmelt`.

Examples

```
## a fictitious data set with a fourth-corner problem

abundance <- data.frame(
sites=c(
"midlatitude", "subtropical", "tropical", "equatorial",
"arctic", "midlatitude", "tropical", "equatorial",
"subtropical"
),
species=c(rep("capybara", 4), rep("moss", 4), "vampire"),
abundance=c(4, 10, 8, 7, 5, 6, 9, 3, 1)
)

environment <- data.frame(
sites=c(
"arctic", "subarctic", "midlatitude", "subtropical",
"tropical", "equatorial"
),
temperature=c(-30, 0, 10, 20, 50, 30),
precipitation=c(20, 40, 20, 100, 150, 200)
)

trait <- data.frame(
species=c("capybara", "moss", "vampire"),
body.size=c(140, 5, 190),
```

```
metabolic.rate=c(20, 5, 0)
)

dlcast(list(abundance, environment, trait),
dimids=c("sites", "species"),
fill=c(0, NA, NA)
)
```

`dlmelt` *Melt data lists*

Description

Melt a data list into a list of molten (i.e. long-format) (i.e. database-like) data frames.

Usage

```
dlmelt(x)
```

Arguments

`x`	A data list

Value

A data list.

Note

This function is inspired by the `melt` function in the `reshape2` package by Hadley Wickham.

See Also

For an approximate inverse of `dlmelt`, see `dlcast`.

Examples

```
data(fake.community)
dlmelt(fake.community)
```

dropdl *Drop Data List Dimensions*

Description

Delete the dimensions of a data list.

Usage

```
dropdl(x)

dropl(x)
```

Arguments

x	A data list (for dropdl) or a list (for dropl).

Value

For dropdl, a data list without the dimensions with only one level, and without any of the variables that are only replicated along dimensions with only one level.

For dropl, a list without null elements.

See Also

drop, data.list, list

Extract.data.list *Extract or replace parts of a data list*

Description

Operators acting on data list objects to extract or replace parts.

Usage

```
## S3 method for class 'data.list'
x[..., drop = TRUE, vextract = TRUE]
## S3 replacement method for class 'data.list'
x$i <- value
## S3 replacement method for class 'data.list'
x[[i, match.dimids, shape, drop = TRUE]] <- value
```

Arguments

x	A data.list object.
i	Subscript or variable name.
value	Replacement value.
...	Vectors to subscript the data list.
drop	If TRUE, single dimension data lists that result from subscripting are coerced to data frames (i.e. their replication dimensions are 'dropped').
vextract	If TRUE, variable extraction is done whenever only a single argument is passed to .... If FALSE, variable extraction is not allowed. Sometimes it is useful to set vextract = FALSE whenever single-dimension data lists are expecting dimension-wise subscripting.
match.dimids	A character vector supplying the dimids (ids of the replication dimensions) that value is replicated along, iff i is a variable name that does not yet exist in the data list, x. Ignored if shape is provided.
shape	A single character string giving the name of an already existing variable in x, whose dimensions will be used for the new variable, i.

Details

The [.data.list method of extraction can be used in two main ways: (1) to extract parts of the replication dimensions of all of the variables in the data list (modeled after the method for arrays and matrices, Extract), and (2) to extract some of the variables while keeping their replication dimensions unaltered (modeled after the method for lists, Extract).

In array-like extraction, each replication dimension of the data list is subscripted as an element in a comma-separated list passed to .... In intuitive terms, all variables in the data list will be subsequently subscripted along the appropriate replication dimensions; in technical terms, there is a two-step process for each variable where (1) the subscripts for the dimensions that this variable is not replicated along are deleted and then (2) the remaining subscripts are used to index that variable. This kind of subscripting is one of the major advantages of the data list concept; because the vectors, matrices, and arrays in a data list are related via dimension sharing, they can all be subscripted in a single line. To emphasize an important point, the main difference between subscripting arrays and array-like subscripting of data lists is that the latter does not allow any dimensions to be either reduced to length zero or be dropped. In general, the differences between these two types of subscripting are (compare with sections 3.4.1 and 3.4.2 in the R language definition),

Integer The special zero index is not allowed with data lists. This is because it creates dimensions with zero length, which leads to non-sensical data frames that are generated by as.data.frame.data.list. Similarly, any subscripting with negative integers that results in some zero-length dimensions, also causes an error.

Other numeric Identical to array-like subscripting, with truncation towards zero to integers.

Logical Identical to array-like subscripting. Note that logical vectors longer than the dimension to be subscripted result in an error, unless there is only a single dimension in the data list – in which case, extra NA elements are added as needed (i.e. just as with subscripting of vectors and one-dimensional arrays).

Character Identical to array-like subscripting.

Factor Factor subscripting is not allowed with data lists.

Empty Unlike array-like subscripting, `x[]` does not drop any attributes but rather simply returns `x`.

NULL NULL subscripting is not allowed with data lists, for the same reason that zero subscripting is not allowed.

Matrix Matrices are currently not valid subscripting objects but this could change in the future.

In list-like extraction, only a single subscript vector is used. A data list with the extracted variables is returned whenever a single subscript vector is used and the input data list has more than one dimension of replication. By default, a one-dimensional data list is also extracted in a list-like manner whenever a single subscript vector is used; however, this behaviour can be changed by setting `vextract = FALSE`, which leads to (one-dimensional-)array-like subscripting. Whenever the resulting data list is composed of variables that are replicated along fewer dimensions than the original data list, those dimensions are dropped without exception. If the resulting data list is only replicated along a single dimension, it is coerced to a data frame by default unless `drop = FALSE`. If no variables remain in the resulting data list, then an error is thrown.

The `$<-.data.list` and `[[<-.data.list` methods of replacement work very much like the default methods `$<-` and `[[<-` for lists, except that they ensure that a valid data list object is returned. One technical difference between the default and `data.list` methods for `$<-` is that new variables cannot be added by passing a new name to `i`. However new variables can be added to an existing data list using `[[<-.data.list`. Adding new variables in this way is only slightly more complicated than adding new variables to a data frame; when adding to a data list, one must specify the dimensions of replication of the new variable in one of two ways. Perhaps the easiest way is via the `shape` argument, which expects the name of an existing variable; the dimensions of the new variable are then assumed to be identical to the dimensions of the old variable specified via `shape`, and if this is not the case an error is thrown. The other way to specify dimensions is via the `match.dimids` argument, which accepts a character vector supplying the ids of the replication dimensions of the new variable. Note that `match.dimids` is ignored if `shape` is also provided. Setting `drop = FALSE` in `` `[[<-`.data.list `` ensures that the output will be a data list (i.e. the replication dimensions will not be dropped to create a data frame even if the resulting data list has only a single dimension). To make use of the default replacement methods for lists, simply `unclass` the data list object as in `unclass(x)[...]`, `unclass(x)$i <- value`, and `unclass(x)[[i]] <- value`.

See Also

`data.list` for creating new data lists and `as.data.frame.data.list` for coercing data lists to data frames.

Examples

```
data(fake.community)
fake.community

## array-like extraction
fake.community[1:3, , ]
fake.community["arctic", c("2009", "2008"), c(TRUE, TRUE, FALSE)]

## list-like extraction
fake.community["abundance"]
fake.community["body.size"]
```

```
fake.community["body.size", drop = FALSE]
fake.community[c("abundance", "body.size")]
fake.community[1:3]

## assignment
fake.community$abundance  <- log1p(fake.community$abundance)
fake.community[[6]] <- NULL
fake.community$body.size <- fake.community$precipitation <- NULL
fake.community

## adding a new variable to an existing data list
fake.community[["newvar", match.dimids = "years"]] <- letters[1:3]
fake.community[["reallynewvar", shape = "metabolic.rate"]] <- runif(3)
fake.community[["sites", match.dimids = "sites"]] <-
dimnames(fake.community)[[1]]
fake.community
```

`fake.community` *Fake community data*

Description

A simple and silly data set for illustrating multiple-table data organization and manipulation.

Usage

```
data(fake.community)
```

Format

A `data.list` object with six fictitious variables (abundance, precipitation, body size, metabolic rate, and homeotherm) replicated along three dimensions (sites, years, species):

	abundance	temperature	precipitation	body.size	metabolic.rate	homeotherm
sites	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE
years	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE
species	TRUE	FALSE	FALSE	TRUE	TRUE	TRUE

Source

My brain.

Examples

```
data(fake.community)
str(fake.community)
summary(fake.community)
```

higgins *Stream fish*

Description

Real community data.

Usage

```
data(higgins)
```

Source

C. L. Higgins (2009) Spatiotemporal variation in functional and taxonomic organization of stream-fish assemblages in central Texas. Aquatic Ecology. 43: 1133-1141.

merge.data.list *Merge Two Data Lists*

Description

Merge two data lists, such that one of the data lists is replicated along all of the dimensions of the other.

Usage

```
## S3 method for class 'data.list'
merge(x, y, ...)
```

Arguments

x	A data list
y	Another data list
...	Not currently used

Details

This `merge.data.list` function is a simple wrapper to data list addition, `Ops.data.list`

Value

Yet another data list

nvar *Number of variables in a data list*

Description

Extract the number of variables in a data list object.

Usage

```
nvar(dl)
```

Arguments

dl	A data.list object.

Details

Implemented as simply `length(dl)`.

Value

The number of variables in `dl`.

Examples

```
data(fake.community)
nvar(fake.community) # 6
```

print.data.list *Printing data lists*

Description

A print method for data lists.

Usage

```
## S3 method for class 'data.list'
print(x, ...)
```

Arguments

x	A data.list object.
...	Not used.

Details

Prints each variable in x proceeded by the underlined name of that variable and followed by the names of the dimensions that that variable is replicated along. A special notation is used whereby dimension names are separated by two vertical bars, ||. After all the variables are printed, the repdim attribute of x is also printed.

See Also

print

---

read.multitable *Multiple table data input*

Description

These functions are used to read several text files and create a data list from them.

Usage

```
read.multitable(files, dimids, fill = rep(NA, length(files)), ...)

read.multicsv(files, dimids, fill = rep(NA, length(files)), ...)

read.multidelim(files, dimids, fill = rep(NA, length(files)), ...)

read.fourthcorner(community, environment, traits, dimids=c("sites", "species"),
community.name = "abundance", ...)

multifile.choose(n)

read.matrix(...)
```

Arguments

files	A character vector with the names of the files containing the tables (possibly created with multifile.choose).
community	A character string of the name of the file containing the community data of a fourth-corner problem.
environment	A character string of the name of the file containing the environment data of a fourth-corner problem.
traits	A character string of the name of the file containing the trait data of a fourth-corner problem.
dimids	A vector with the names of the columns associated with replication dimensions. For read.fourthcorner this is simply the names of the replication dimensions.
fill	See dlcast.

community.name	Character string of the name of community matrix.
n	Number of files to choose.
...	Additional arguments to pass to read.table. The most important such argument is sep, which specifies how entries are separated in the data files. Note that headers MUST be present in order to distinguish columns for replication dimensions from columns for variables, and therefore header is set to TRUE and cannot be changed. Exception: header may be missing in read.fourthcorner.

Details

The read.multitable function is a multiple-table version of read.table. It is largely a wrapper for dlcast, so the tables that are read need to produce 'long' format data frames. The implementation of read.multitable is very simple: (1) repeatedly call read.table to load in the files with names in files and then (2) call dlcast to combine these tables into a data list. Therefore, the dimids and fill arguments are simply passed to dlcast.

The read.multicsv and read.multidelim simply wrap read.multitable with the appropriate value for the sep argument (compare with read.csv for example).

The read.fourthcorner function reads in three files (with names passed to community, environment, and traits), and creates a data list out of the resulting files. The community table is coerced to a matrix so that the rows and columns are treated as two different dimensions of replication, whereas the other two tables are left as data frames. This function is useful because many community ecologists will store their data in this way.

The multifile.choose function allows the interactive selection of the names of n files. Compare with file.choose.

The read.matrix function is identical to read.table but returns a matrix instead of a data frame. The definition of this function is simply a call to read.table wrapped in a call to as.matrix.

Value

A data.list object.

Examples

```
abundance.file <- tempfile()
environment.file <- tempfile()
trait.file <- tempfile()

abundance <- data.frame(
sites=c(
"midlatitude","subtropical","tropical","equatorial",
"arctic","midlatitude","tropical","equatorial",
"subtropical"
),
species=c(rep("capybara",4),rep("moss",4),"vampire"),
abundance=c(4,10,8,7,5,6,9,3,1)
)

environment <- data.frame(
sites=c(
```

```
"arctic","subarctic","midlatitude","subtropical",
"tropical","equatorial"
),
temperature=c(-30,0,10,20,50,30),
precipitation=c(20,40,20,100,150,200)
)

trait <- data.frame(
species=c("capybara","moss","vampire"),
body.size=c(140,5,190),
metabolic.rate=c(20,5,0)
)

write.table(abundance,abundance.file,sep=",")
write.table(environment,environment.file,sep=",")
write.table(trait,trait.file,sep=",")

files <- c(abundance.file,environment.file,trait.file)
dimids <- c("sites","species")
read.multicsv(files,dimids,fill=c(0,NA,NA))

## Modifications necessary to use the read.fourthcorner function
abundance <- data.frame(
capybara = c(4,10,8,7,0,0),
moss = c(6,0,9,3,5,0),
vampire = c(0,1,0,0,0,0),
row.names = c(
"arctic","subarctic","midlatitude","subtropical",
"tropical","equatorial"
)
)
write.table(abundance,abundance.file,sep=",")
read.fourthcorner(abundance.file,environment.file,trait.file,sep=",")
```

`simple.scale` *'Simple' Scaling and Centering*

Description

Simpler version of the `scale` function, which works better for variables in data lists.

Usage

```
simple.scale(x, center = TRUE, scale = TRUE, simplify = TRUE)
```

Arguments

`x`	an R object with numeric data
`center`	should mean of `x` be subtracted from `x`?

scale	should x (centered or otherwise) be divided by its (population) standard deviation?
simplify	should x be first passed through simplify2array?

Details

simple.scale simplifies scale in three ways:

- Treats matrix-like objects as a single variable, rather than several variables (i.e. one per column as in scale).
- Uses the population version of the standard deviation, rather than the sample version.
- Preserves dimensions of arrays of any number of dimensions, rather than collapse to a single column matrix for arrays with three or more dimensions.
- Doesn't bother to check if x is numeric.

For further information, just type simple.scale to see the source (it really is a very simple function).

Value

A scaled version of x (possibly simplified to an array, if simplify = TRUE).

See Also

scale

---

summary.data.list *Information about data list objects*

Description

Summarize and compactly display the structure of a data list object.

Usage

```
## S3 method for class 'data.list'
summary(object, ...)

## S3 method for class 'data.list'
str(object, give.attr = FALSE, hide.internals = TRUE, ...)
```

Arguments

object	A data list object
give.attr	See documentation for str. Ignored if hide.internals is FALSE.
hide.internals	If FALSE, object is unclassed first to reveal further internal details.
...	Potential further arguments (see str)

Details

The summary.data.list function returns a logical matrix indicating which dimensions (rows) are associated with which variables (columns) of the data list object.

Currently str.data.list is identical to str except that give.attr defaults to FALSE, because data lists are 'cluttered' with attributes that should not be part of the str summary. Also, setting the hide.internals argument to FALSE reveals more information. Perhaps in the future, a special str method will be written for data lists.

Note

The hide.internals argument was suggested by a reviewer of a JSS manuscript.

See Also

summary, str

Examples

```
data(fake.community)
summary(fake.community)
str(fake.community)
```

---

variable *Create simple data lists*

Description

These functions create data lists with a single variable (variable) or several variables with identical dimensions of replication (variableGroup).

Usage

```
variable(x, dimids, name)
variableGroup(x, dimids)
```

Arguments

x	For variable, the variable (vector, matrix, or array); for variableGroup, a data.frame or list with the variables in the group.
dimids	Character vector of identifiers for the dimensions of replication of the variable(s).
name	Character string giving the name of the variable

Value

A data.list object.

See Also

Designed to be used in variable arithmetic (see Ops.data.list).

To create more complex data lists see: data.list; as.data.list; dlcast; read.multitable

Examples

```
##
variable(runif(5), "a dimension", "a variable")

## stringing variables together using the + operator
variable(matrix(runif(15), 5, 3), c("n","m"), "A") +
variable(letters[1:3], "m", "B") +
variable(runif(5), "n", "C") +
variable(array(runif(15*4), c(3,5,4)), c("m","n","p"), "D") +
variableGroup(data.frame(a = runif(5), b = runif(5)), "n") +
variableGroup(list(
c = matrix(runif(20), 4, 5),
d = matrix(runif(20), 4, 5)
), c("p","n"))
```

---

variablize *Variablization*

Description

Convert all but the first dimension of replication of an array or data list into variables. Currently an experimental function.

Usage

```
variablize(x, ...)

## Default S3 method:
variablize(x, ...)

## S3 method for class 'factor'
variablize(x, ...)

## S3 method for class 'data.list'
variablize(x, ...)
```

Arguments

x	A data list
...	Not currently used

Value

A data frame with the first dimension of x representing the observations (rows) and all other dimensions of x treated as variables (columns). If x is a data.list then any variable not replicated along the first dimension is removed.

Note

To use a dimension other than the first as the rows of the data frame, transpose the array (aperm) or data list (aperm.data.list) first.

See Also

aperm.data.list,data.list

Examples

```
data(fake.community)
variablize(fake.community)
variablize(t(fake.community))
```

---

varnames *Data list variable names*

Description

Retrieve and set variable names in data list objects.

Usage

```
varnames(dl)
varnames(dl) <- value
```

Arguments

dl	A data.list object.
value	A character vector with the names of the data list.

Details

Currently, varnames(dl) is implemented as names(dl), and therefore the two are identical.

See Also

names for a similar function and nvar for the number of variables in a data.list object.

`with.data.list` *Evaluate an Expression in a Data (List) Environment*

Description

Evaluate an R expression in an environment constructed from a data list, possibly modifying the original data.

Usage

```
## S3 method for class 'data.list'
with(data, expr, func = "I", ...)
## S3 method for class 'data.list'
within(data, expr, ...)
```

Arguments

`data`	A data list.
`expr`	Expression to evaluate.
`func`	A character string giving the name of a function to process `data` before evaluating `expr`. Defaults to "`I`" which just uses `data` unaltered. Another useful choice is "`as.data.frame`".
`...`	Arguments to pass to `func`. Not used with `within.data.list`.

Value

For with, the value of the evaluated `expr`. For `within`, the modified `data.list`.

See Also

`with`

Index