Package 'mosaic'

August 29, 2016

Type Package

Title Project MOSAIC Statistics and Mathematics Teaching Utilities

Version 0.14.4

Date 2016-07-29

Depends R (>= 3.0.0), dplyr, lattice (>= 0.20-21), ggplot2,
mosaicData, Matrix

Imports lazyeval (>= 0.2.0), MASS, grid, tidyr, readr, methods, utils,
splines, latticeExtra, ggdendro, gridExtra

Suggests lubridate, fastR, magrittr, NHANES, RCurl, sp, maptools, vcd,
testthat, knitr, tools, parallel, mapproj, rgl, rmarkdown

Enhances manipulate

VignetteBuilder knitr

Author Randall Pruim <rpruim@calvin.edu>, Daniel T. Kaplan
<kaplan@macalester.edu>, Nicholas J. Horton <nhorton@amherst.edu>

Maintainer Randall Pruim <rpruim@calvin.edu>

Description Data sets and utilities from Project MOSAIC (http://mosaic-web.org) used
to teach mathematics, statistics, computation and modeling. Funded by the
NSF, Project MOSAIC is a community of educators working to tie together
aspects of quantitative work that students in science, technology,
engineering and mathematics will need in their professional lives, but
which are usually taught in isolation, if at all.

License GPL (>= 2)

LazyLoad yes

LazyData yes

URL https://github.com/ProjectMOSAIC/mosaic

BugReports https://github.com/ProjectMOSAIC/mosaic/issues

RoxygenNote 5.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2016-07-29 19:07:11

R **topics documented:**

| mosaic-package | *mosaic: the Project MOSAIC package* |

Description

mosaic

Details

Data sets and utilities from Project MOSAIC (mosaic-web.org) used to teach mathematics, statistics, computation and modeling. Funded by the NSF, Project MOSAIC is a community of educators working to tie together aspects of quantitative work that students in science, technology, engineering and mathematics will need in their professional lives, but which are usually taught in isolation, if at all.

Author(s)

Randall Pruim (<rpruim@calvin.edu>), Daniel Kaplan (<kaplan@macalester.edu>), Nicholas Horton (<nhorton@smith.edu>)

References

<http://mosaic-web.org>

| .polyExp | *Takes a call and returns its polynomial coefficients* |

Description

Takes a call and returns its polynomial coefficients

Takes a call and returns its polynomial coefficients as numerics.

Method for putting a polynomial together given the coefficients and power from .polyExp()

Usage

```
.polyExp(tree, .x., params, iterate = 1)

.polyExp.num(tree, .x.)

.makePoly(form, poly)
```

Arguments

`tree`	A call that will be parsed and simplified recursively
`.x.`	the variable name with respect to which the polynomial should be most simplified
`params`	All names of free variables. If there are no free variables, the value should be ""
`iterate`	The number of times the call is nested. Default and proper value when called from the outside is 1
`form`	original formula - provides information on which variable the polynomial was reduced with respect to.
`poly`	output of .polyExp()

Details

Will work on any call as long as it can be reduced to a polynomial with respect the the variable and each of the parameters. Operates recursively, reducing each of the coefficients with respect to the extra parameters in turn. Calls .polyExp.num when all remaining coefficients are numeric to reduce the expression more fully.

works with the same structure as .polyExp() but will return only if all coefficients reduce to numeric values.

Value

A list containing a list, `coeffs`, of coefficients ordered high to low (i.e. the list (2,3,4) would correspond to the polynomial 2*x^2+3*x+4) and value, pow, indicating the order of the polynomial. If the expression is not a polynomial, this method returns an empty list or an error.

A list containing a list, `coeffs`, of coefficients ordered high to low (i.e. the list (2,3,4) would correspond to the polynomial 2*x^2+3*x+4) and value, pow, indicating the order of the polynomial. If the expression is not a polynomial, this method returns an empty list or an error.

A formula whose left hand side is a polynomial that fits the description given with the input poly.

---

adapt_seq   *Adaptively generate sequences in an interval*

Description

`adapt_seq` is similar to `seq` except that instead of selecting points equally spaced along an interval, it selects points such that the values of a function applied at those points are (very) roughly equally spaced. This can be useful for sampling a function in such a way that it can be plotted more smoothly, for example.

Usage

```
adapt_seq(from, to, length.out = 200, f = function(x, ...) {      1 },
  args = list(), quiet = FALSE)
```

Arguments

`from`	start of interval
`to`	end of interval
`length.out`	desired length of sequence
`f`	a function
`args`	arguments passed to `f`
`quiet`	suppress warnings about NaNs, etc.

Value

a numerical vector

Examples

```
adapt_seq(0, pi, 25, sin)
```

---

aggregatingFunction1   *1-ary Aggregating functions*

Description

aggregatinFuntion1 creates statistical summaries of one numerical vector that are formula aware.

Usage

```
aggregatingFunction1(fun, output.multiple = FALSE, envir = parent.frame(),
  na.rm = getOption("na.rm", FALSE), style = c("formula1st", "formula",
  "flexible"))
```

Arguments

`fun`	a function that takes a numeric vector and computes a summary statistic, returning a numeric vector.
`output.multiple`	
	a boolean indicating whether `fun` returns multiple values
`envir`	an environment in which evaluation takes place.
`na.rm`	the default value for na.rm in the resulting function.
`style`	one of `"formula1st"`, `"formula2nd"` or `"flexible"`. In the first two cases, the first argument must be a formula or evaluate to an object. In the latter case, bare names will be converted into formulas.

Details

The logic of the resulting function is this: 1) If the first argument is a formula, use that formula and `data` to create the necessary call(s) to `fun`; (2) Else simply pass everything to `fun` for evaluation.

Value

a function that generalizes fun to handle a formula/data frame interface.

Note

Earlier versions of this function supported a "bare name + data frame" interface. This functionality has been removed since it was (a) ambiguous in some cases, (b) unnecessary, and (c) difficult to maintain.

Examples

```
if (require(mosaicData)) {
  foo <- aggregatingFunction1(base::mean)
  foo( ~ length, data=KidsFeet)
  base::mean(KidsFeet$length)
  foo(length ~ sex, data=KidsFeet)
}
```

---

aggregatingFunction1or2

1- or 2-ary aggregating functions

Description

aggregatingFunction1or2() creates statistical summaries for functions like [var](http://)() that can have either 1 or 2 numeric vector inputs.

Usage

```
aggregatingFunction1or2(fun, output.multiple = FALSE,
  na.rm = getOption("na.rm", FALSE))
```

Arguments

fun	a function that takes 1 or 2 numeric vectors and computes a summary statistic, returning a numeric vector of length 1.
output.multiple	
	a boolean indicating whether fun returns multiple values
na.rm	the default value for na.rm in the resulting function.

Details

This was designed primarily to support var which can be used to compute either the variance of one variable or the covariance of two variables. The logic of the resulting function is this: 1) If the first two arguments are both formulas, then those formulas are evaluated (with data) to compute the covariance; (2) If the first argument is a formula, and the second is NULL, then the formula and data are used to create the necessary call(s) to fun; (3) Else everything is simply passed to fun for evaluation.

Note

Earlier versions of this function supported a "bare name + data frame" interface. This functionality has been removed since it was (a) ambiguous in some cases, (b) unnecessary, and (c) difficult to maintain.

---

aggregatingFunction2   *2-ary aggregating functions*

Description

aggregatinFuntion2 creates statistical summaries of two numerical vectors that are formula aware.

Usage

```
aggregatingFunction2(fun)
```

Arguments

fun   a function that takes two numeric vectors and computes a summary statistic, returning a numeric vector of length 1.

Details

This was designed to support functions like cov() which can be used to compute numerical summaries from two numeric vectors. The logic of the resulting function is this: 1) If the first two arguments are both formulas, then those formulas are evaluated (with data) to compute the covariance; (2) If the first argument is a formula, and the second is NULL, then the left and ride sides of the formula and data are used to create the vectors passed to fun; (3) Else everything is simply passed to fun for evaluation.

Value

a function that generalizes fun to handle a formula/data frame interface.

Note

Earlier versions of this function supported a "bare name + data frame" interface. This functionality has been removed since it was (a) ambiguous in some cases, (b) unnecessary, and (c) difficult to maintain.

Examples

```
if(require(mosaicData)) {
  foo <- aggregatingFunction2(stats::cor)
  foo(length ~ width, data=KidsFeet)
   stats::cor(KidsFeet$length, KidsFeet$width)
}
```

---

as.xtabs   *Convert objects to xtabs format*

Description

Convert a data frame or a matrix into an xtabs object.

Usage

```
as.xtabs(x, ...)

## S3 method for class 'data.frame'
as.xtabs(x, rowvar = NULL, colvar = NULL, labels = 1,
  ...)

## S3 method for class 'matrix'
as.xtabs(x, rowvar = NULL, colvar = NULL, ...)
```

Arguments

x	object (typically a data frame) to be converted to xtabs format
...	additional arguments to be passed to or from methods.
rowvar	name of the row variable as character string
colvar	name of the column variable as character string
labels	column of data frame that contains the labels of the row variable.

Details

The intended use is to convert a two-way contingency table stored in a data frame or a matrix into an xtabs object.

Value

An xtabs object.

Examples

```
# example from example(fisher.test)
df <- data.frame( X=c('Tea','Milk'), Tea=c(3,1), Milk=c(1,3) )
xt <- as.xtabs(df, rowvar="Guess", colvar="Truth"); xt
if (require(vcd)) { mosaic(xt) }
```

---

ashplot   *Average Shifted Histograms*

Description

An ASH plot is the average over all histograms of a fixed bin width.

Mainly a utility for the **lattice** and **ggplot2** plotting functions, ash_points() returns the points to be plotted.

Usage

```
ashplot(x, ..., width = NULL, adjust = NULL, panel = panel.ashplot,
  prepanel = prepanel.default.ashplot)

prepanel.default.ashplot(x, darg, groups = NULL, subscripts = TRUE, ...)

panel.ashplot(x, darg = list(), plot.points = FALSE, ref = FALSE,
  groups = NULL, jitter.amount = 0.01 * diff(current.panel.limits()$ylim),
  type = "p", ..., identifier = "ash")

ash_points(x, binwidth = NULL, adjust = 1)

StatAsh

stat_ash(mapping = NULL, data = NULL, geom = "line",
  position = "identity", na.rm = FALSE, show.legend = NA,
  inherit.aes = TRUE, binwidth = NULL, adjust = 1, ...)

geom_ash(mapping = NULL, data = NULL, stat = "ash",
  position = "identity", na.rm = FALSE, show.legend = NA,
  inherit.aes = TRUE, binwidth = NULL, adjust = 1, ...)
```

Arguments

x	a formula or numeric vector
...	additional arguments passed to panel and prepanel functions or data, a data frame in which to find the variales used for the plot.
width	the historam bin width.
adjust	a numeric adjustment to width. Primarily useful when width is not specified. Increasing adjust makes the plot smoother.
panel	a panel funtion
prepanel	a prepanel function
darg	a list of arguments for the function computing the ASH.
groups	as in other lattice plots

subscripts	as in other lattice prepanel functions
plot.points	One of TRUE, FALSE, "jitter", or "rug"
ref	a logical indicating whether a reference line should be displayed
jitter.amount	when plot.points="jitter", the value to use as the amount argument to [jitter]().
type	type argument used to plot points, if requested. This is not expected to be useful, it is available mostly to protect a type argument, if specified, from affecting the display of the ASH.
identifier	A character string that is prepended to the names of i grobs that are created by this panel function.
binwidth	the width of the histogram bins. If NULL (the default) the binwidth will be chosen so that approximately 10 bins cover the data. adjust can be used to to increase or decrease binwidth.
mapping	set of aesthetic mappings created by [aes]() or [aes_]().
data	a data frame
geom	a geom to use for this layer, as a string.
position	position adjustment, either as a string or the result of a call to a position adjustment function.
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. [borders]().
stat	a statistical transformation to use on the data for this layer, as a string.

Format

An object of class StatAsh (inherits from Stat, ggproto) of length 3.

Value

a data frame containing x and y coordinates of the resulting ASH plot.

Examples

```
ashplot( ~age | substance, groups = sex, data = HELPrct)
ggplot(faithful, aes(x = eruptions)) +
  geom_histogram(aes(y = ..density..),
    fill = "lightskyblue", colour = "gray50", alpha = 0.2) +
  geom_ash(colour = "red") +
  geom_ash(colour = "forestgreen", adjust = 2) +
  geom_ash(colour = "navy", adjust = 1/2) +
  theme_minimal()
```

---

bargraph *Create bar graphs from raw data*

Description

[barchart](#) from the `lattice` package makes bar graphs from pre-tabulated data. Raw data can be tabulated using [xtabs](#), but the syntax is unusual compared to the other lattice plotting functions. bargraph provides an interface that is consistent with the other `lattice` functions.

Usage

```
bargraph(x, data = parent.frame(), groups = NULL, horizontal = FALSE,
  origin = 0, ylab = ifelse(horizontal, "", type),
  xlab = ifelse(horizontal, type, ""), type = c("count", "frequency",
  "proportion", "percent"), ...)
```

Arguments

x	a formula describing the plot
data	a data frame in which the formula x is evaluated
groups	a variable or expression used for grouping. See [barchart](#).
horizontal	a logical indicating whether bars should be horizontal
origin	beginning point for bars. For the default behavior used by [barchart](#) set `origin` to NULL, but 0 is often a better default. If 0 is not good, perhaps you should use a different kind of plot as the results may be misleading.
ylab	a character vector of length one used for the y-axis label
xlab	a character vector of length one used for the x-axis label
type	one of "frequency", "count", "percent", or "proportion" indicating what type of scale to use. Unique prefixes are sufficient.
...	additional arguments passed to [barchart](#)

Details

bargraph(formula, data=data, ...) works by creating a new data frame from xtabs(formula, data=data) and then calling [barchart](#) using modified version of the formula and this new data frame as inputs. This has implications on, for example, conditional plots where one desires to condition on some expression that will be evaluated in `data`. This typically does not work becuase the required variables do not exist in the output of xtabs. One solution is to first add a new variable to `data` first and then to condition using this new variable. See the examples.

Value

a trellis object describing the plot

See Also

barchart

Examples

```
if (require(mosaicData)) {
data(HELPrct)
bargraph( ~ substance, data = HELPrct)
bargraph( ~ substance, data = HELPrct, horizontal = TRUE)
bargraph( ~ substance | sex, groups = homeless, auto.key = TRUE, data = HELPrct)
bargraph( ~ substance, groups = homeless, auto.key=TRUE,
            data = HELPrct %>% filter(sex == "male"))
HELPrct2 <- mutate(HELPrct, older = age > 40)
bargraph( ~ substance | older, data = HELPrct2)
bargraph( ~ rbinom(1000, 10, 0.5))
}
```

---

binom.test   *Exact Tests for Proportions*

Description

The binom.test function performs an exact test of a simple null hypothesis about the probability of success in a Bernoulli experiment from summarized data or from raw data. The mosaic binom.test provides wrapper functions around the function of the same name in **stats**. These wrappers provide an extended interface (including formulas).

Usage

```
binom.test(x, n = NULL, p = 0.5, alternative = c("two.sided", "less",
  "greater"), conf.level = 0.95, ci.method = c("Clopper-Pearson",
  "binom.test", "Score", "Wilson", "prop.test", "Wald", "Agresti-Coull",
  "Plus4"), data = NULL, success = NULL, ...)
```

Arguments

x	count of successes, length 2 vector of success and failure counts, a formula, or a character, numeric, or factor vector containing raw data.
n	sample size (successes + failures) or a data frame (for the formula interface)
p	probability for null hypothesis
alternative	type of alternative hypothesis
conf.level	confidence level for confidence interval
ci.method	a method to use for computing the confidence interval (case insensitive and may be abbreviated). See details below.
data	a data frame (if missing, n may be a data frame)
success	level of variable to be considered success. All other levels are considered failure.
...	additional arguments (often ignored)

Details

binom.test is a wrapper around binom.test from the base package to simplify its use when the raw data are available, in which case an extended syntax for binom.test is provided. See the examples.

Also, five confidence interval methods are provided:

Clopper-Pearson, binom.test This is the interval produced when using binom.test from the stats package. It guarantees a coverage rate at least as large as the nominal coverage rate, but may produce wider intervals than some of the methods below, which may either under- or over-cover depending on the data.

Score, Wilson, prop.test This is the usual method used by prop.test and is computed by inverting p-values from score tests. It is often atrributed to Edwin Wilson.

Wald This is the interval traditionally taught in entry level statistics courses. It uses the sample proportion to estimate the standard error and uses normal theory to determine how many standard deviations to add and/or substract from the sample proportion to determine an interval.

Agresti-Coull This is the Wald method after setting $n' = n + z^2$ and $p' = (x + z^2/2)/n'$ and using $x' = n'p'$ and n' in place of x and n.

Plus4 This is Wald after adding in two artifical success and two artificial failures. It is nearly the same as the Agresti-Coull method when the confidence level is 95 z^2 is approximately 4 and $z^2/2$ is approximately 2.

Value

an object of class htest

Note

When x is a 0-1 vector, 0 is treated as failure and 1 as success. Similarly, for a logical vector TRUE is treated as success and FALSE as failure.

See Also

prop.test, binom.test

Examples

```
# Several ways to get a confidence interval for the proportion of Old Faithful
# eruptions lasting more than 3 minutes.
data(faithful)
binom.test(faithful$eruptions > 3)
binom.test(97, 272)
binom.test(c(97, 272-97))
faithful$long <- faithful$eruptions > 3
binom.test(faithful$long)
binom.test(resample(1:4, 400), p=.25)
binom.test(~ long, data = faithful)
binom.test(~ long, data = faithful, ci.method = "Wald")
binom.test(~ long, data = faithful, ci.method = "Plus4")
with(faithful, binom.test(~long))
```

```
with(faithful, binom.test(long))
```

Broyden   *Multi-Dimensional Root Finding*

Description

Implementation of Broyden's root finding function to numerically compute the root of a system of nonlinear equations

Usage

```
Broyden(system, vars, x = 0, tol = .Machine$double.eps^0.4,
  maxiters = 10000)
```

Arguments

system	A list of functions
vars	A character string list of variables that appear in the functions
x	A starting vector
tol	The tolerance for the function specifying how precise it will be
maxiters	maximum number of iterations.

---

cdist   *Central portion of a distribution*

Description

This function determines the critial values for isolating a central portion of a distribution with a specified probability. This is designed to work especially well for symmteric distributions, but it can be used with any distribution.

Usage

```
cdist(dist, p, ..., verbose = FALSE)
```

Arguments

dist	a character string naming a distribution family (e.g., "norm"). This wil work for any family for which the usual d/p/q functions exist.
p	the proportion to be in the central region, with equal proportions in either "tail".
...	additional arguments passed to the distribution functions. Typically these specify the parameters of the particular distribution desired. See the examples.
verbose	a logical indicating whether a more verbose output value should be returned.

Value

a pair of numbers indicating the upper and lower bounds, unless verbose is TRUE, in which case a 1-row data frame is returned containing these bounds, the central probability, the tail probabilities, and the name of the distribution.

Note

This function is still experimental and changes the input or output formats are possible in future versions of the package.

Examples

```
cdist( "norm", .95)
cdist( "t", c(.90, .95, .99), df=5)
cdist( "t", c(.90, .95, .99), df=50)
cdist( "t", .95, df=c(3,5,10,20))
cdist( "t", .95, df=c(3,5,10,20), verbose = TRUE )
cdist( "norm", .95, mean=500, sd=100 )
cdist( "chisq", c(.90, .95), df=3 )
# CI
x <- rnorm(23, mean = 10, sd = 2)
cdist("t", p = 0.95, df=22)
mean(x) + cdist("t", p = 0.95, df=22) * sd(x) / sqrt(23)
confint(t.test(x))
cdist("t", p = 0.95, df=22, verbose = TRUE)
```

---

chisq	*Extract Chi-squared statistic*

Description

Extract Chi-squared statistic

Usage

```
chisq(x, ...)

## S3 method for class 'htest'
chisq(x, ...)

## S3 method for class 'table'
chisq(x, correct = FALSE, ...)

## Default S3 method:
chisq(x, correct = FALSE, ...)
```

Arguments

x	An object of class `"htest"` a coming from a Chi-squared test, an object of class `"table"`, or the inputs to [tally](#).
...	additional arguments passed on to `tally` or `chisq.test`.
correct	a logical indicating whether a continuity correction should be applied.

See Also

[stat](#)

Examples

```
if(require(mosaicData)) {
  Mites.table <- tally( ~ outcome + treatment, data=Mites )
  Mites.table
  chisq.test(Mites.table)
  chisq(Mites.table)
  chisq(chisq.test(Mites.table))
  ## Randomization test.  Increase replications to decrease Monte Carlo error.
  do(3) * chisq( tally( ~ outcome + shuffle(treatment),  data=Mites ) )
  Mites.rand <- do(1000) * chisq( tally( ~ outcome + shuffle(treatment),  data=Mites ) )
  tally( ~(X.squared >= chisq(Mites.table)), data=Mites.rand, format="proportion")
}
```

CIAdata   *Return a dataset based on the CIA World Factbook*

Description

This function can be used in two different ways. Without an argument, it returns a reference table that includes information about all the CIA World Factbook tables that are available through this function. Note the `Name` column that indicates a unique name for each available dataset. If this name is passed as an argument to the function, the function will return the corresponding dataset.

Usage

```
CIAdata(name = NULL)
```

Arguments

name	An optional parameter specifying the name of the desired dataset. If multiple names are given, a merge will be attempted on the individual data sets.

Examples

```
head(CIAdata())
Population <- CIAdata("pop")
nrow(Population)
head(Population)

PopArea <- CIAdata(c("pop","area")) %>% mutate(density = pop / area)
nrow(PopArea)
head(PopArea)
PopArea %>%
  filter(!is.na(density)) %>%
  arrange(density) %>%
  tail
```

CIsim   *Compute confidence intervals from (multiple) simulated data sets*

Description

This function automates the calculation of coverage rates for exploring the robustness of confidence interval methods.

Usage

```
CIsim(n, samples = 100, rdist = rnorm, args = list(), plot = if (samples
  <= 200) "draw" else "none", estimand = 0, conf.level = 0.95,
  method = t.test, method.args = list(), interval = function(x) {
  do.call(method, c(list(x, conf.level = conf.level), method.args))$conf.int },
  estimate = function(x) {     do.call(method, c(list(x, conf.level =
  conf.level), method.args))$estimate }, verbose = TRUE)
```

Arguments

n	size of each sample
samples	number of samples to simulate
rdist	function used to draw random samples
args	arguments required by rdist
plot	one of "print", "return", "horizontal", or "none" describing whether a plot should be printed, returned, printed with horizontal intervals, or not generated at all.
estimand	true value of the parameter being estimated
conf.level	confidence level for intervals
method	function used to compute intervals. Standard functions that produce an object of class htest can be used here.
method.args	arguments required by method

interval	a function that computes a confidence interval from data. Function should return a vector of length 2.
estimate	a function that computes an estimate from data
verbose	print summary to screen?

Value

A data frame with variables `lower`, `upper`, `estimate`, `cover` ('Yes' or 'No'), and `sample` is returned invisibly. See the examples for a way to use this to display the intervals graphically.

Examples

```
# 1000 95% intervals using t.test; population is N(0,1)
CIsim(n=10, samples=1000)
# this time population is Exp(1); fewer samples, so we get a plot
CIsim(n=10, samples=100, rdist=rexp, estimand=1)
# Binomial treats 1 like success, 0 like failure
CIsim(n=30, samples=100, rdist=rbinom, args=list(size=1, prob=.7),
       estimand = .7, method = binom.test, method.args=list(ci = "Plus4"))
```

---

coef.function   *Extract coefficients from a function*

Description

coef will extract the coefficients attribute from a function. Functions created by applying link{makeFun} to a model produced by [lm](#), [glm](#), or [nls](#) store the model coefficients there to enable this extraction.

Usage

```
## S3 method for class 'function'
coef(object, ...)
```

Arguments

| object | a function |
| ... | ignored |

Examples

```
if (require(mosaicData)) {
model <- lm( width ~ length, data=KidsFeet)
f <- makeFun( model )
coef(f)
}
```

---

columns	*return a vector of row or column indices*

Description

return a vector of row or column indices

Usage

```
columns(x, default = c())

rows(x, default = c())
```

Arguments

x	an object that may or may not have any rows or columns
default	what to return if there are no rows or columns

Value

if x has rows or columns, a vector of indices, else `default`

Examples

```
columns(iris)
if (require(mosaicData)) {
dim(HELPrct)
columns(HELPrct)
rows(HELPrct)
}
columns(NULL)
columns("this doesn't have columns")
```

---

compareMean	*Compare means between 2 groups*

Description

A function to calculate the difference between the means of a continuous variable for two groups.

Usage

```
compareMean(formula, data = parent.frame(), ...)
```

Arguments

formula	a formula
data	a data frame in which x is evaluated if x is a formula. Note that the default is data=parent.frame(). This makes it convenient to use this function interactively by treating the working envionment as if it were a data frame. But this may not be appropriate for programming uses. When programming, it is best to use an explicit data argument – ideally supplying a data frame that contains the variables mentioned
...	other arguments

Value

the difference in means between the second and first group

Note

This funciton has been deprecated. Use diffmean instead.

See Also

do, compareProportion and shuffle

Examples

```
if (require(mosaicData)) {
  data(HELPrct)
  # calculate the observed difference
  mean(age ~ sex, data=HELPrct)
  obs <- diffmean(age ~ sex, data=HELPrct); obs
  # calculate the permutation distribution
  nulldist <- do(100) * diffmean(age ~ shuffle(sex),
    data=HELPrct)
  histogram(~ diffmean, groups=(diffmean >= obs), nulldist,
    xlab="difference in means")
}
```

---

compareProportion   *Compare proportions between 2 groups*

Description

A function to facilitate 2 group permutation tests for a categorical outcome variable

Usage

```
compareProportion(formula, data = NULL, ...)
```

Arguments

`formula`	a formula
`data`	a data frame in which x is evaluated if x is a formula.
`...`	other arguments

Value

the difference in proportions between the second and first group

Note

This funciton has been deprecated. Use [`diffprop`](#) instead.

Examples

```
if (require(mosaicData)) {
  data(HELPrct)
  # calculate the observed difference
  mean(homeless=="housed" ~ sex, data=HELPrct)
  obs <- diffprop(homeless=="housed" ~ sex, data=HELPrct); obs
  # calculate the permutation distribution
  nulldist <- do(100) * diffprop(homeless=="housed" ~ shuffle(sex), data=HELPrct)
  histogram(~ diffprop, groups=(diffprop>= obs), nulldist,
    xlab="difference in proportions")
}
```

---

confint   *Confidence interval methods for output of resampling*

Description

Methods for `confint` to compute confidence intervals on numerical vectors and numerical components of data frames.

Usage

```
## S3 method for class 'numeric'
confint(object, parm, level = 0.95, ...,
  method = "percentile", margin.of.error = "stderr" %in% method ==
  "stderr")

## S3 method for class 'do.tbl_df'
confint(object, parm, level = 0.95, ...,
  method = "percentile", margin.of.error = "stderr" %in% method,
  df = NULL)

## S3 method for class 'do.data.frame'
```

```
confint(object, parm, level = 0.95, ...,
  method = "percentile", margin.of.error = "stderr" %in% method,
  df = NULL)

## S3 method for class 'data.frame'
confint(object, parm, level = 0.95, ...)

## S3 method for class 'summary.lm'
confint(object, parm, level = 0.95, ...)
```

Arguments

object   and R object

parm   a vector of parameters

level   a confidence level

...   additional arguments

method   a character vector of methods to use for creating confidence intervals. Choices
                  are "percentile" (or "quantile") which is the default, "stderr" (or "se"), "bootstrap-
                  t", and "reverse" (or "basic"))

margin.of.error
                  if true, report intervals as a center and margin of error.

df   degrees for freedom. This is required when object was produced using link{do}
                  when using the standard error to compute the confidence interval since typically
                  this information is not recorded in these objects. The default (Inf) uses a normal
                  critical value rather than a one derived from a t-distribution.

Details

The methods of producing confidence intervals from bootstrap distributions are currently quite
naive. In particular, when using the standard error, assistance may be required with the degrees
of freedom, and it may not be possible to provide a correct value in all situations. None of the
methods include explicit bias correction. Let q_a be the a quantile of the bootstrap distribution, let
t_a, df be the a quantile of the t distribution with df degrees of freedom, let SE_b be the standard
deviation of the bootsrap distribution, and let $\hat{\theta}$ be the estimate computed from the original data.
Then the confidence intervals with confidence level $1 - 2a$ are

quantile (q_a, q_{1-a})

reverse $(2\hat{\theta} - q_{1-a}, 2\hat{\theta} - q_a)$

stderr $(\hat{\theta} - t_{1-a,df} SE_b, \hat{\theta} + t_{1-a,df} SE_b)$. When df is not provided, at attempt is made to determine
an appropriate value, but this should be double checked. In particular, missing data an lead to
unreliable results.

The bootstrap-t confidence interval is computed much like the reverse confidence interval
but the bootstrap t distribution is used in place of a theoretical t distriubiton. This interval
has much better properties than the reverse (or basic) method, which is here for comparison
purposes only and is not recommended.

Value

When applied to a data frame, returns a data frame giving the confidence interval for each variable in the data frame using `t.test` or `binom.test`, unless the data frame was produced using do, in which case it is assumed that each variable contains resampled statistics that serve as an estimated sampling distribution from which a confidence interval can be computed using either a central proportion of this distribution or using the standard error as estimated by the standard deviation of the estimated sampling distribution. For the standard error method, the user must supply the correct degrees of freedom for the t distribution since this information is typically not available in the output of do.

When applied to a numerical vector, returns a vector.

References

Tim C. Hesterberg (2015): What Teachers Should Know about the Bootstrap: Resampling in the Undergraduate Statistics Curriculum, The American Statistician, `http://dx.doi.org/10.1080/00031305.2015.1089789`.

Examples

```
if (require(mosaicData)) {
  bootstrap <- do(500) * diffmean( age ~ sex, data=resample(HELPrct) )
  confint(bootstrap)
  confint(bootstrap, method = "percentile")
  confint(bootstrap, method = "boot")
  confint(bootstrap, method = "se", df=nrow(HELPrct) - 1)
  confint(bootstrap, margin.of.error = FALSE)
  confint(bootstrap, margin.of.error = TRUE, level=0.99, method=c("boot", "se", "perc") )
  bootstrap2 <- do(500)*mean( resample(1:10) )
  confint(bootstrap2)
}
confint( summary(lm(width ~ length * sex, data=KidsFeet)) )
```

---

confint.htest	*Extract summary statistics*

Description

Extract confidence intervals, test statistics or p-values from an `htest` object.

Usage

```
## S3 method for class 'htest'
confint(object, parm, level, ...)

pval(x, ...)

## S3 method for class 'htest'
```

```
pval(x, digits = 4, verbose = FALSE, ...)

stat(x, ...)

## S3 method for class 'htest'
stat(x, ...)
```

Arguments

`object`	a fitted model object or an htest object.
`parm`	a specification of which parameters are to be given confidence intervals, either a vector of numbers or a vector of names. If missing, all parameters are considered.
`level`	the confidence level required.
`x`	An object of class `htest`.
`digits`	number of digits to display in verbose output
`verbose`	a logical
`...`	Additional arguments.

Value

the extracted p-value, confidence interval, or test statistic

Examples

```
confint(t.test(rnorm(100)))
pval(t.test(rnorm(100)))
stat(t.test(rnorm(100)))
confint(var.test(rnorm(10,sd=1), rnorm(20, sd=2)))
pval(var.test(rnorm(10,sd=1), rnorm(20, sd=2)))
if (require(mosaicData)) {
data(HELPrct)
stat(t.test (age ~ shuffle(sex), data=HELPrct))
# Compare to test statistic computed with permuted values of sex.
do(10) * stat(t.test (age ~ shuffle(sex), data=HELPrct))
}
```

---

cross	*Factor cross products*

Description

Construct a product of factors.

Usage

```
cross(..., sep = ":", drop.unused.levels = FALSE)
```

Arguments

sep	separator between levels
drop.unused.levels	
	should levels that do not appear in cross product be dropped?
...	factors to be crossed.

Value

a factor

Examples

```
x <- letters[1:3]
y <- c(1,2,1,1,3,1,3)
cross(x, y)
cross(x, y, drop.unused.levels=TRUE)
```

---

cull_for_do	*Cull objects used with do()*

Description

The do function facilitates easy repliaction for randomization tests and bootstrapping (among other things). Part of what makes this particularly useful is the ability to cull from the objects produced those elements that are useful for subsequent analysis. cull_for_do does this culling. It is generic, and users can add new methods to either change behavoir or to hanlde additional classes of objects.

Usage

```
cull_for_do(object, ...)
```

Arguments

object	an object to be culled
...	additional arguments (currently ignored)

Details

When do(n) * expression is evaluated, expression is evaluated n times to produce a list of n result objects. cull_for_do is then applied to each element of this list to extract from it the information that should be stored. For example, when applied to a object of class "lm", the default cull_for_do extracts the coefficients, coefficient of determinism, an the estimate for the variance, etc.

Examples

```
cull_for_do(lm(length ~ width, data = KidsFeet))
do(1) * lm(length ~ width, data = KidsFeet)
```

| D | *Derivative and Anti-derivative operators* |

Description

Operators for computing derivatives and anti-derivatives as functions.

Usage

```
D(formula, ..., .hstep = NULL, add.h.control = FALSE)

## Default S3 method:
D(formula, ..., .hstep = NULL, add.h.control = FALSE)

## S3 method for class 'formula'
D(formula, ..., .hstep = NULL, add.h.control = FALSE)

antiD(formula, ..., lower.bound = 0, force.numeric = FALSE)

makeAntiDfun(.function, .wrt, from, .tol = .Machine$double.eps^0.25)

numerical_integration(f, wrt, av, args, vi.from, ciName = "C", .tol)
```

Arguments

formula	A formula. The right side of a formula specifies the variable(s) with which to carry out the integration or differentiation. On the left side should be an expression or a function that returns a numerical vector of the same length as its argument. The expression can contain unbound variables. Functions will be differentiated as if the formula f(x) ~ x were specified but with x replaced by the first argument of f.
.hstep	horizontal distance between points used for secant slope calculation in numerical derivatives.
add.h.control	logical indicating whether the returned derivative function should have an additional parameter for setting .hstep. Meaningful only for numerical derivatives.
lower.bound	for numerical integration only, the lower bound used
force.numeric	If TRUE, a numerical integral is performed even when a symbolic integral is available.
.function	function to be integrated
.wrt	character string naming the variable of integration
from	default value for the lower bound of the integral region
.tol	Numerical tolerance. See stats::integrate
f	a function
wrt	character string naming a variable: the var. of integration

av	a list of the arguments passed to the function calling this
args	default values (if any) for parameterss
vi.from	the the lower bound of the interval of integration
ciName	character string giving the name of the symbol for the constant of integration
...	Default values to be given to unbound variables in the expression expr. See examples.#' Note that in creating anti-derivative functions, default values of "from" and "to" can be assigned. They are to be written with the name of the variable as a prefix, e.g. y.from.

Details

D attempts to find a symbolic derivative for simple expressions, but will provide a function that is a numerical derivative if the attempt at symbolic differentiation is unsuccessful. The symbolic derivative can be of any order (although the expression may become unmanageably complex). The numerical derivative is limited to first or second-order partial derivatives (including mixed partials). antiD will attempt simple symbolic integration but if it fails it will return a numerically-based anti-derivative.

antiD returns a function with the same arguments as the expression passed to it. The returned function is the anti-derivative of the expression, e.g., antiD(f(x)~x) -> F(x). To calculate the integral of f(x), use F(to) - F(from).

Value

For derivatives, the return value is a function of the variable(s) of differentiation, as well as any other symbols used in the expression. Thus, D(A*x^2 + B*y ~ x + y) will compute the mixed partial with respect to x then y (that is, $\frac{d^2 f}{dy\,dx}$). The returned value will be a function of x and y, as well as A and B. In evaluating the returned function, it's best to use the named form of arguments, to ensure the order is correct.

a function of the same arguments as the original expression with a constant of integration set to zero by default, named "C", "D", ... depending on the first such letter not otherwise in the argument list.

Note

numerical_integration is not intended for direct use. It packages up the numerical anti-differentiation process so that the contents of functions produced by antiD look nicer to human readers.

Examples

```
D(sin(t) ~ t)
D(A*sin(t) ~ t )
D(A*sin(2*pi*t/P) ~ t, A=2, P=10) # default values for parameters.
f <- D(A*x^3 ~ x + x, A=1) # 2nd order partial -- note, it's a function of x
f(x=2)
f(x=2,A=10) # override default value of parameter A
g <- D(f(x=t, A=1)^2 ~ t)  # note: it's a function of t
g(t=1)
gg <- D(f(x=t, A=B)^2 ~ t, B=10)  # note: it's a function of t and B
gg(t=1)
```

```
gg(t=1, B=100)
f <- makeFun(x^2~x)
D(f(cos(z))~z) #will look in user functions also
antiD( a*x^2 ~ x, a = 3)
antiD( A/x~x ) # This gives a warning about no default value for A
F <- antiD( A*exp(-k*t^2 ) ~ t, A=1, k=0.1)
F(t=Inf)
one = makeFun(1 ~ x + y)
by.x = antiD(one(x=x, y=y) ~ x, y=1)
by.xy = antiD(by.x(x = sqrt(1-y^2), y = y) ~ y)
4 * by.xy(y = 1) # area of quarter circle
```

---

deg2rad   *Convert between degrees and radians*

Description

Facilitates conversion between degrees and radians.

Usage

```
deg2rad(x)

rad2deg(x)
```

Arguments

x   a numeric vector

Value

a numeric vector

See Also

[latlon2xyz](), [googleMap](), and [rgeo]().

Examples

```
deg2rad(180)
rad2deg(2*pi)
```

---

deltaMethod | *deltaMethod has moved to a separate package*

Description

deltaMethod has moved to its own separate package.

Usage

```
deltaMethod(...)
```

Arguments

...	arguments (ignored, since this is defunct)

---

derivedVariable | *Create new variables from logicals*

Description

Utility functions for creating new variables from logicals describing the levels

Usage

```
derivedVariable(..., .ordered = FALSE, .method = c("unique", "first",
  "last"), .debug = c("default", "always", "never"), .sort = c("given",
  "alpha"), .default = NULL, .asFactor = FALSE)

derivedFactor(..., .asFactor = TRUE)
```

Arguments

.ordered	a logical indicating whether the resulting factored should be ordered Ignored if .asFactor is FALSE.
.method	one of "unique", "first", and "last". If "unique", exactly one rule must be TRUE for each position. If "first", the first TRUE rule defines the level. If "last", the last TRUE rule defines the level.
.debug	one of "default", "always", and "never", indicating whehter debugging information should be printed. If "default", debugging information is printed only when multiple rules give conflicting definitions for some positions.
.sort	One of "given" (the default) or "alpha" or a vector of integers the same length as the number of levels indicating the order in which the levels should appear in the resulting factor. Ignored if .asFactor is FALSE.
.default	character vector of length 1 giving name of default level or NULL for no default.
.asFactor	A logical indicating whether the returned value should be a factor.
...	named logical "rules" defining the levels.

Details

Each logical "rule" corresponds to a level in the resulting variable. If .default is defined, an implicit rule is added that is TRUE whenever all other rules are FALSE. When there are multiple TRUE rules for a slot, the first or last such is used or an error is generated, depending on the value of method.

derivedVariable is designed to be used with [transform](#) or [mutate](#) to add new variables to a data frame. derivedFactor() is the same but that the default value for .asFactor is TRUE. See the examples.

Examples

```
Kf <- mutate(KidsFeet, biggerfoot2 = derivedFactor(
                    dom = biggerfoot == domhand,
                    nondom = biggerfoot != domhand)
                    )
tally( ~ biggerfoot + biggerfoot2, data = Kf)
tally( ~ biggerfoot + domhand, data = Kf)

# Three equivalent ways to define a new variable
# Method 1: explicitly define all levels
modHELP <- mutate(HELPrct, drink_status = derivedFactor(
  abstinent = i1 == 0,
  moderate = (i1>0 & i1<=1 & i2<=3 & sex=='female') |
      (i1>0 & i1<=2 & i2<=4 & sex=='male'),
  highrisk = ((i1>1 | i2>3) & sex=='female') |
       ((i1>2 | i2>4) & sex=='male'),
  .ordered = TRUE)
)
tally( ~ drink_status, data = modHELP)

# Method 2: Use .default for last level
modHELP <- mutate(HELPrct, drink_status = derivedFactor(
  abstinent = i1 == 0,
  moderate = (i1<=1 & i2<=3 & sex=='female') |
      (i1<=2 & i2<=4 & sex=='male'),
  .ordered = TRUE,
  .method = "first",
  .default = "highrisk")
)
tally( ~ drink_status, data = modHELP)

# Method 3: use TRUE to catch any fall through slots
modHELP <- mutate(HELPrct, drink_status = derivedFactor(
  abstinent = i1 == 0,
  moderate = (i1<=1 & i2<=3 & sex=='female') |
      (i1<=2 & i2<=4 & sex=='male'),
  highrisk=TRUE,
  .ordered = TRUE,
  .method = "first"
  )
)
```

```
tally( ~ drink_status, data = modHELP)
is.factor(modHELP$drink_status)

modHELP <- mutate(HELPrct, drink_status = derivedVariable(
  abstinent = i1 == 0,
  moderate = (i1<=1 & i2<=3 & sex=='female') |
      (i1<=2 & i2<=4 & sex=='male'),
  highrisk=TRUE,
  .ordered = TRUE,
  .method = "first"
  )
)
is.factor(modHELP$drink_status)
```

---

dfapply   *apply-type function for data frames*

Description

An apply-type function for data frames.

Usage

```
dfapply(data, FUN, select = TRUE, ...)
```

Arguments

data	data frame
FUN	a function to apply to (some) variables in the data frame
select	a logical, character (naming variables), or numeric vector or a function used to select variables to which FUN is applied. If a function, it should take a vector as input and return a single logical. See examples.
...	arguments passed along to FUN

See Also

[apply](#), [sapply](#), [tapply](#), [lapply](#), [inspect](#)

Examples

```
dfapply(iris, favstats, select = is.numeric)
dfapply(iris, favstats, select = c(TRUE, TRUE, FALSE, FALSE, FALSE))
dfapply(iris, favstats, select = c(1,2))
dfapply(iris, favstats, select = c("Sepal.Length", "Petal.Length"))
dfapply(HELPrct, table, select = is.factor)
do.call(rbind, dfapply(HELPrct, favstats, select = is.numeric))
```

---

diffmean   *Difference in means and proportions*

Description

Wrappers around diff(mean(...)) and diff(prop(...)) that facilitate better naming of the result

Usage

```
diffmean(x, ..., data = parent.frame(), only.2 = TRUE)

diffprop(x, ..., data = parent.frame(), only.2 = TRUE)
```

Arguments

x, data, ...   as in [mean](#) or [prop](#)

only.2   a logical indicating whether differences should only be computed between two groups.

Examples

```
if (require(mosaicData)) {
diffprop( homeless ~ sex , data=HELPrct)
do(3) * diffprop( homeless ~ shuffle(sex) , data=HELPrct)
diffmean( age ~ substance, data=HELPrct, only.2=FALSE)
do(3) * diffmean(age ~ shuffle(substance), data=HELPrct, only.2=FALSE)
diffmean( age ~ sex, data=HELPrct)
do(3) * diffmean(age ~ shuffle(sex), data=HELPrct)
}
```

---

do   *Do Things Repeatedly*

Description

do() provides a natural syntax for repetition tuned to assist with replication and resampling methods.

Usage

```
do(object, ...)

## S3 method for class 'numeric'
do(object, ...)

## Default S3 method:
do(object, ...)

Do(n = 1L, cull = NULL, mode = "default", algorithm = 1,
  parallel = TRUE)

## S3 method for class 'repeater'
print(x, ...)

## S4 method for signature 'repeater,ANY'
e1 * e2
```

Arguments

object	an object
...	additional arguments
n	number of times to repeat
cull	function for culling output of objects being repeated. If NULL, a default culling function is used. The default culling function is currently aware of objects of types lme, lm, htest, table, cointoss, and matrix.
mode	target mode for value returned
algorithm	a number usd to select the algorithm used. Currently numbers below 1 use an older algorithm and numbers >=1 use a newer algorithm which is faster in some situations.
parallel	a logical indicating whether parallel computation should be attempted using the **parallel** package (if it is installed and loaded).
x	an object created by do.
e1	an object (in cases documented here, the result of running do)
e2	an object (in cases documented here, an expression to be repeated)

Value

do returns an object of class repeater which is only useful in the context of the operator *. See the examples.

Note

do is a thin wrapper around Do to avoid collision with do from the **dplyr** package.

Author(s)

Daniel Kaplan (<kaplan@macalaster.edu>) and Randall Pruim (<rpruim@calvin.edu>)

See Also

[replicate](#), [set.rseed](#)

Examples

```
do(3) * rnorm(1)
do(3) * "hello"
do(3) * 1:4
do(3) * mean(rnorm(25))
if (require(mosaicData)) {
  do(3) * lm(shuffle(height) ~ sex + mother, Galton)
  do(3) * anova(lm(shuffle(height) ~ sex + mother, Galton))
  do(3) * c(sample.mean = mean(rnorm(25)))
  set.rseed(1234)
  do(3) * tally( ~sex|treat, data=resample(HELPrct))
  set.rseed(1234)  # re-using seed gives same results again
  do(3) * tally( ~sex|treat, data=resample(HELPrct))
}
```

---

docFile   *Return the path to a documentation file in a package*

Description

Return the path to a documentation file in a package

Usage

```
docFile(file, package = "mosaic", character.only = FALSE)
```

Arguments

file	the name of a file
package	the name of a package
character.only	a logical. If TRUE package names must be specified as character, else names will be converted as a convenience as is [library](#) and [library](#).

Value

a character vector specifying the path to the file on the user's system.

Examples

```
MustangPrice <- read.file(docFile("MustangPrice.csv", "mosaic"))
```

---

dotPlot   *Dotplots*

Description

A high level function and panel function for producing a variant of a histogram called a dotplot.

Usage

```
dotPlot(x, breaks, ..., panel = panel.dotPlot)

panel.dotPlot(x, breaks, equal.widths = TRUE, groups = NULL, nint = if
  (is.factor(x)) nlevels(x) else round(1.3 * log2(length(x)) + 4), pch, col,
  lty = trellis.par.get("dot.line")$lty,
  lwd = trellis.par.get("dot.line")$lwd,
  col.line = trellis.par.get("dot.line")$col,
  alpha = trellis.par.get("dot.symbol")$alpha, cex = 1, type = "count",
  ...)
```

Arguments

x	a vector of values or a formula
breaks, equal.widths, groups, pch, col, lty, lwd, col.line, type, alpha	
	as in `histogram`
panel	a panel function
nint	the number of intervals to use
cex	a ratio by which to increase or decrease the dot size
...	additional arguments

Value

a trellis object

See Also

`histogram`

Examples

```
if (require(mosaicData)) {
dotPlot( ~ age, data = HELPrct)
dotPlot( ~ age, nint=42, data = HELPrct)
dotPlot( ~ height | voice.part, data = singer, nint = 17,
          endpoints = c(59.5, 76.5), layout = c(4,2), aspect = 1,
          xlab = "Height (inches)")
}
```

---

dpqrdist   *Distribution wrapper*

Description

Utility function wrapping up the d/p/q/r distribution functions

Usage

```
dpqrdist(dist, type = c("d", "p", "q", "r"), ...)
```

Arguments

dist   a character discription of a distribution, for example "norm", "t", or "chisq"

type   one of "x", "p", "q", or "r"

...   additional arguments passed on to underlying distribution function. Note that
                one of d, p, q, or n must be a named argument in ...

Examples

```
# 3 random draws from N(1,2)
dpqrdist("norm", "r", n=3, mean = 1, sd = 2)
# These should all be the same
dpqrdist("norm", "d", x=0) == dnorm(x=0)
dpqrdist("norm", "p", q=0, mean = 1, sd = 2) == pnorm(q=0, mean = 1, sd = 2)
dpqrdist("norm", "q", p=0.5, mean = 1, sd = 2) == qnorm(p=0.5, mean = 1, sd = 2)
```

---

ediff   *Lagged Differences with equal length*

Description

Often when creating lagged differences, it is awkward that the differences vector is shorter than the
original. ediff pads with NAs to make its output the same length as the input.

Usage

```
ediff(x, lag = 1, differences = 1, pad = c("head", "tail", "symmetric"),
  frontPad, ...)
```

Arguments

x	a numeric vector or a matrix containing the values to be differenced
lag	an integer indicating which lag to use
differences	an integer indicating the order of the difference
pad	one of "head", "tail", or "symmetric". indicating where the NA padding should be added to the result.
frontPad	logical indicating whether padding is on the front (head) or back (tail) end. This exists for backward compatibility. New code should use pad instaed.
...	further arguments to be passed to or from methods

See Also

diff since ediff is a thin wrapper around diff.

Examples

```
ediff(1:10)
ediff(1:10, 2)
ediff(1:10, 2, 2)
x <- cumsum(cumsum(1:10))
ediff(x, lag = 2)
ediff(x, differences = 2)
ediff(x, differences = 2, pad="symmetric")
ediff(.leap.seconds)
if (require(mosaicData)) {
Men <- subset(SwimRecords, sex=="M")
Men <- mutate(Men, change=ediff(time), interval=ediff(year))
head(Men)
}
```

---

evalFormula	*Evaluate a formula*

Description

Evaluate a formula

Usage

```
evalFormula(formula, data = parent.frame(), subset, ops = c("+", "&"))
```

Arguments

formula	a formula (y ~ x | z) to evaluate
data	a data frame or environment in which evaluation occurs
subset	an optional vector describing a subset of the observations to be used. Currently only implemented when data is a data frame.
ops	a vector of operator symbols allowable to separate variables in rhs

Value

a list containing data frames corresponding to the left, right, and condition slots of formula

Examples

```
if (require(mosaicData)) {
data(CPS85)
cps <- CPS85[1:6,]
cps
evalFormula(wage ~ sex & married & age | sector & race, data=cps)
}
```

---

evalSubFormula   *Evaluate a part of a formula*

Description

Evaluate a part of a formula

Usage

```
evalSubFormula(x, data = NULL, ops = c("+", "&"), env = parent.frame())
```

Arguments

x	an object appearing as a subformula (typically a name or a call)
data	a data frame or environment in which things are evaluated
ops	a vector of operators that are not evaluated as operators but instead used to further split x
env	an environment in which to search for objects not in data.

Value

a data frame containing the terms of the evaluated subformula

Examples

```
if (require(mosaicData)) {
data(CPS85)
cps <- CPS85[1:6,]
cps
evalSubFormula( rhs( ~ married & sector), data=cps )
}
```

---

expandFun   *Expand the left-hand side of a formula*

Description

Expands the contents of functions used in a formula.

Usage

```
expandFun(formula, ...)
```

Arguments

formula   A mathematical expression (see examples and [plotFun](#))
...   additional parameters

Value

A list with the new expanded formula and the combined formals

Examples

```
f=makeFun(x^2~x)
expandFun(f(z)~z) #Returns z^2~z
```

---

factorize   *Conditionally convert vectors to factors*

Description

A generic function and several instances for creating factors from other sorts of data. The primary use case is for vectors that contain few unique values and might be better considered as factors. When applied to a data frame, this is applied to each variable in the data frame.

Usage

```
factorize(x, ...)

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

## S3 method for class 'numeric'
factorize(x, max.levels = 5L, ...)

## S3 method for class 'character'
factorize(x, max.levels = 5L, ...)
```

```
## S3 method for class 'data.frame'
factorize(x, max.levels = 5L, ...)

factorise(x, ...)
```

Arguments

x	an object
...	additional arguments (currently ignored)
max.levels	an integer. Only convert if the number of unique values is no more than `max.levels`.

Examples

```
data(KidsFeet, package="mosaicData")
str(KidsFeet)
factorize(KidsFeet$birthyear)
str(factorize(KidsFeet))
# alternative spelling
str(factorise(KidsFeet))
```

---

fav_stats   *Some favorite statistical summaries*

Description

Likely you mean to be using [favstats](#). Each of these computes the mean, standard deviation, quartiles, sample size and number of missing values for a numeric vector, but [favstats](#) can take a formula describing how these summary statistics should be aggregated across various subsets of the data.

Usage

```
fav_stats(x, ..., na.rm = TRUE, type = 7)
```

Arguments

x	numeric vector
...	additional arguments (currently ignored)
na.rm	boolean indicating whether missing data should be ignored
type	an integer between 1 and 9 selecting one of the nine quantile algorithms detailed in the documentation for [quantile](#)

Value

A vector of statistical summaries

Examples

```
fav_stats(1:10)
fav_stats(faithful$eruptions)
favstats(Sepal.Length ~ Species, data=iris)  # Note: this is favstats() rather than fav_stats()
```

---

fetchData	*Defunct functions now in the fetch package*

Description

These functions have been moved to the fetch package.

Usage

```
fetchData(...)

fetchGapminder1(...)

fetchGapminder(...)

fetchGoogle(...)
```

Arguments

...	arguments

---

findZeros	*Find zeros of functions*

Description

Compute numerically zeros of a function or simultaneous zeros of multiple functions.

Solve an equation

Usage

```
findZeros(expr, ..., xlim = c(near - within, near + within), near = 0,
  within = Inf, nearest = 10, npts = 1000, iterate = 1,
  sortBy = c("byx", "byy", "radial"))

## S3 method for class 'formula'
solve(form, ..., near = 0, within = Inf, nearest = 10,
  npts = 1000, iterate = 1, sortBy = c("byx", "byy", "radial"))
```

Arguments

expr	A formula. The right side names the variable with respect to which the zeros should be found. The left side is an expression, e.g. sin(x) ~ x. All free variables (all but the variable on the right side) named in the expression must be assigned a value via ...
xlim	The range of the dependent variable to search for zeros. Inf is a legitimate value, but is interpreted in the numerical sense as the non-Inf largest floating point number. This can also be specified replacing x with the name of the variable. See the examples.
near	a value near which zeros are desired
within	only look for zeros at least this close to near. near and within provide an alternative to using xlim to specify the search space.
nearest	the number of nearest zeros to return. Fewer are returned if fewer are found.
npts	How many sub-intervals to divide the xlim into when looking for candidates for zeros. The default is usually good enough. If Inf is involved, the intervals are logarithmically spaced up to the largest finite floating point number. There is no guarantee that all the roots will be found.
iterate	maximum number of times to iterate the search. Subsequent searches take place with the range of previously found zeros. Choosing a large number here is likely to kill performance without improving results, but a value of 1 (the default) or 2 works well when searching in c(-Inf,Inf) for a modest number of zeros near near.
sortBy	specifies how the zeros found will be sorted. Options are 'byx', 'byy', or 'radial'.
form	Expression to be solved
...	Formulas corresponding to additional functions to use in simultaneous zero finding and/or specific numerical values for the free variables in the expression.

Details

Searches numerically using uniroot.

Uses findZerosMult of findZeros to solve the given expression

Value

A dataframe of zero or more numerical values. Plugging these into the expression on the left side of the formula should result in values near zero.

a dataframe with solutions to the expression.

Author(s)

Daniel Kaplan (<kaplan@macalester.edu>)

Cecylia Bocovich

Examples

```
findZeros( sin(t) ~ t, xlim=c(-10,10) )
# Can use tlim or t.lim instead of xlim if we prefer
findZeros( sin(t) ~ t, tlim=c(-10,10) )
findZeros( sin(theta) ~ theta, near=0, nearest=20)
findZeros( A*sin(2*pi*t/P) ~ t, xlim=c(0,100), P=50, A=2)
# Interval of a normal at half its maximum height.
findZeros( dnorm(x,mean=0,sd=10) - 0.5*dnorm(0,mean=0,sd=10) ~ x )
# A pathological example
# There are no "neareset" zeros for this function.  Each iteration finds new zeros.
f <- function(x) { if (x==0) 0 else sin(1/x) }
findZeros( f(x) ~ x, near=0 )
# Better to look nearer to 0
findZeros( f(x) ~ x, near=0, within=100 )
findZeros( f(x) ~ x, near=0, within=100, iterate=0 )
findZeros( f(x) ~ x, near=0, within=100, iterate=3 )
# Zeros in multiple dimensions (not run: these take a long time)
# findZeros(x^2+y^2+z^2-5~x&y&z, nearest=3000, within = 5)
# findZeros(x*y+z^2~z&y&z, z+y~x&y&z, npts=10)
solve(3*x==3~x)

# plot out sphere (not run)
# sphere = solve(x^2+y^2+z^2==5~x&y&z, within=5, nearest=1000)
# cloud(z~x+y, data=sphere)
```

---

findZerosMult   *Find the zeros of a function of two or more variables*

Description

Compute numerically zeros of a function of two or more variables. All free variables (all but the variable on the right side) named in the expression must be assigned a value via ...

Usage

```
findZerosMult(..., npts = 10, rad = 5, near = 0, sortBy = "byx")
```

Arguments

...	arguments for values NOTE: if the system has more than one equation and the rhs variables do not match up, there will be an error.
npts	number of desired zeros to return
rad	radius around near in which to look for zeros
near	center of search for zeros
sortBy	options for sorting zeros for plotting. Options are 'byx', 'byy' and 'radial'. The default value is 'byx'.

Details

sorts points in the domain according to the sign of the function value at respective points. Use continuity and uniroot to find zeros between points of opposite signs. Returns any number of points which may be sorted and plotted according to x, y, or radial values.

Value

A data frame of numerical values which should all result in a value of zero when input into original function

Author(s)

Cecylia Bocovich

Examples

```
findZerosMult(a*x^2-8~a&x, npts = 50)
findZerosMult(a^2+x^2-8~a&x, npts = 100, sortBy='radial')
## Not run: findZerosMult(a^2+x^2-8~a&x, npts = 1000, sortBy='radial')
```

---

  fitModel   *Fit a nonlinear least squares model*

Description

Allows you to specify a formula with parameters, along with starting guesses for the parameters. Refines those guesses to find the least-squares fit.

Usage

```
fitModel(formula, data = parent.frame(), start = list(), ...)

model(object, ...)

## S3 method for class 'nlsfunction'
model(object, ...)

## S3 method for class 'nlsfunction'
summary(object, ...)

## S3 method for class 'nlsfunction'
coef(object, ...)
```

Arguments

formula	formula specifying the model
data	dataframe containing the data to be used
start	passed as start to nls. If and empty list, a simple starting point is used (thus avoiding the usual warning message).
object	an R object (typically a the result of fitModel)
...	additional arguments passed to nls

Details

Fits a nonlinear least squares model to data. In contrast to linear models, all the parameters (including linear ones) need to be named in the formula. The function returned simply contains the formula together with pre-assigned arguments setting the parameter value. Variables used in the fitting (as opposed to parameters) are unassigned arguments to the returned function.

Value

a function

Note

This doesn't work with categorical explanatory variables. Also, this does not work with synthentic data that fit the model perfectly. See link{nls} for details.

See Also

linearModel, nls

Examples

```
if (require(mosaicData)) {
f <- fitModel(temp ~ A+B*exp(-k*time), data=CoolingWater, start=list(A=50,B=50,k=1/20))
f(time=50)
coef(f)
summary(f)
model(f)
}
```

---

fitSpline   *Fit splines to data*

Description

These functions create mathematical functions from data, using splines.

Usage

```
fitSpline(formula, data = parent.frame(), df = NULL, knots = NULL,
  degree = 3, type = c("natural", "linear", "cubic", "polynomial"), ...)
```

Arguments

formula	a formula. Only one quantity is allowed on the left-hand side, the output quantity
data	a data frame in which formula is evaluated.
df	degrees of freedom (used to determine how many knots should be used)
knots	a vector of knots
degree	parameter for splines when type is "polynomial". 1 is locally linear, 2 is locally quadratic, etc.
type	type of splines to use; one of "linear", "cubic", "natural" (cubic with linear tails, the default), or "polynomial".
...	additional arguments passed to spline basis functions (ns and bs).

Value

a function of the explanatory variable

See Also

bs and ns for the bases used to generate the splines.

Examples

```
f <- fitSpline( weight ~ height, data=women, df=5 )
xyplot( weight ~ height, data=women )
plotFun(f(height) ~ height, add=TRUE)

g <- fitSpline( length ~ width, data = KidsFeet, type='natural', df=5 )
h <- fitSpline( length ~ width, data = KidsFeet, type='linear', df=5 )
xyplot( length ~ width, data = KidsFeet, col='gray70', pch=16)
plotFun(g, add=TRUE, col='navy')
plotFun(h, add=TRUE, col='red')
```

---

formularise	*Convert lazy objects into formulas*

Description

Convert lazy objects into a formula

Usage

```
formularise(lazy_formula, envir = parent.frame())
```

Arguments

`lazy_formula`	an object of class `lazy`
`envir`	an environment that will be come the environment of the returned formula

Details

The expression of the lazy object is evaluated in its environment. If the result is not a formula, then the formula is created with an empty left hand side and the expression on the right hand side.

Value

a formula

Examples

```
formularise(lazyeval::lazy(foo))
formularise(lazyeval::lazy(y ~ x))
bar <- a ~ b
formularise(lazyeval::lazy(bar))
```

---

fortify.hclust   *mosaic tools for clustering*

Description

mosaic tools for clustering

Usage

```
## S3 method for class 'hclust'
fortify(model, data, which = c("segments", "heatmap",
  "leaves", "labels", "data"), k = 1, ...)

## S3 method for class 'hclust'
mplot(object, data, colorize = TRUE, k = 1,
  labels = FALSE, heatmap = 0, enumerate = "white", ...)
```

Arguments

`model`	a model
`data`	a data-like object
`which`	which kind of fortification to compute
`k`	number of clusters
`...`	additional arguments passed on to `link{dendro_data}`
`object`	an object of class `"hclust"`

colorize	whether to show clusters in different colors
labels	a logical indicating whether labels should be used to identify leaves of the tree.
heatmap	the ratio of size of heatmap to size of dendrogram. Use `0` or `FALSE` to omit the heatmap.
enumerate	a color used for numbers within heatmap. Use `"transparent"` to hide.

Examples

```
KidsFeet %>% select(-name, -birthmonth) %>% rescale() -> KidsFeet2
  M <- dist(KidsFeet2)
  Cl <- hclust(M)
  fortify(Cl, k=5) %>% head(3)
  fortify(Cl, which="heatmap", data=KidsFeet2) %>% head(3)
  fortify(Cl, which="data", data=KidsFeet2) %>% head(3)
  fortify(Cl, which="labels") %>% head(3)
  mplot(Cl, data=KidsFeet2, k=4, heatmap=2)
  mplot(Cl, data=KidsFeet2, k=4, heatmap=0.5, enumerate="transparent")
  mplot(Cl, data=KidsFeet2, k=4, heatmap=2, type="triangle")
  mplot(Cl, data=KidsFeet2, k=4, heatmap=0, type="triangle")
```

---

fortify.summary.lm   *Extract data from R objects*

Description

Extract data from R objects

Usage

```
## S3 method for class 'summary.lm'
fortify(model, data = NULL, level = 0.95, ...)

## S3 method for class 'summary.glm'
fortify(model, data = NULL, level = 0.95, ...)

## S3 method for class 'TukeyHSD'
fortify(model, data, order = c("asis", "pval",
  "difference"), ...)
```

Arguments

model	an R object
data	original data set, if needed
level	confidence level
...	additional arguments
order	one of `"pval"`, `"diff"`, or `"asis"` determining the order of the `pair` factor, which determines the order in which the differences are displayed on the plot.

Examples

```
fortify(TukeyHSD(lm(age ~ substance, data=HELPrct)))
```

---

freqpoly	*Turn histograms into frequency polygons*

Description

Turn histograms into frequency polygons

Usage

```
freqpoly(x, plot = TRUE, ...)

hist2freqpolygon(hist)

## S3 method for class 'freqpolygon'
plot(x, freq = equidist, col = graphics::par("fg"),
  lty = NULL, lwd = 1, main = paste("Frequency polygon of", paste(x$xname,
  collapse = "\n")), sub = NULL, xlab = x$xname, ylab, xlim = range(x$x),
  ylim = NULL, axes = TRUE, labels = FALSE, add = FALSE, ann = TRUE,
  ...)
```

Arguments

x	a vector of values for which a frequency polygon is desired.
plot	a logical indicating if a plot should be generated.
...	additional arguments passed on to [hist]().
hist	a histgram object produced by link{hist}().
freq	A logical indicating whether the vertical scale should be frequency (count).
col	A color for the frequency polygon.
lty	An integer indicating the line type.
lwd	An integer indicating the line width.
main	A title for the plot.
sub	A sub-title for the plot.
xlab	Label for the horizontal axis.
ylab	Label for the vertical axis.
xlim	A numeric vector of lenght 2.
ylim	A numeric vector of lenght 2.
axes	A logical indicating whehter axes should be drawn.
labels	A logical indicating whether labels should be printed or a character vector of labels to add.
add	A logical indicating whether the plot should be added to the current plot
ann	A logical indicating whether annotations (titles and axis titles) should be plotted.

Value

An object of class `"freqpoly"` (invisibly). Additionally, if `plot` is TRUE, a plot is generated.

Examples

```
freqpoly(faithful$eruptions)
bks <- c(0, 1, 1.5, 2, 3, 3.5, 4, 4.5, 5, 7)
hist(faithful$eruptions, breaks = bks)
freqpoly(faithful$eruptions, col = rgb(0,0,1,.5), lwd = 5, breaks = bks, add = TRUE)
```

---

freqpolygon   *Frequency Polygons*

Description

Frequency polygons are an alternative to histograms that make it simpler to overlay multiple distributions.

Usage

```
freqpolygon(x, ..., panel = "panel.freqpolygon",
  prepanel = "prepanel.default.freqpolygon")

prepanel.default.freqpolygon(x, darg = list(), plot.points = FALSE,
  ref = FALSE, groups = NULL, subscripts = TRUE, jitter.amount = 0.01 *
  diff(current.panel.limits()$ylim), center = NULL, nint = NULL,
  breaks = NULL, width = darg$width, type = "density", ...)

panel.freqpolygon(x, darg = list(), plot.points = FALSE, ref = FALSE,
  groups = NULL, weights = NULL, jitter.amount = 0.01 *
  diff(current.panel.limits()$ylim), type = "density", breaks = NULL,
  nint = NULL, center = NULL, width = darg$width,
  gcol = trellis.par.get("reference.line")$col,
  glwd = trellis.par.get("reference.line")$lwd, h, v, ...,
  identifier = "freqpoly")
```

Arguments

x	a formula or a numeric vector
panel	a panel function
prepanel	a prepanel function
darg	a list of arguments for the function computing the frequency polygon. This exists primarily for compatibility with `densityplot` and is unlikely to be needed by the end user.

plot.points	one of TRUE, FALSE, "jitter", or "rug" indicating how points are to be displayed
ref	a logical indicating whether a horizontal reference line should be added (roughly equivalent to h=0)
groups, weights, jitter.amount, identifier	
	as in densityplot or histogram
subscripts	as in other lattice prepanel functions
center	center of one of the bins
nint	an approximate number of bins for the frequency polygon
breaks	a vector of breaks for the frequency polygon bins
width	width of the bins
type	one of 'density', 'percent', or 'count'
gcol	color of guidelines
glwd	width of guidelines
h, v	a vector of values for additional horizontal and vertical lines
...	additional arguments passed on to histogram and panel.

Value

a trellis object

Note

This function make use of histogram to determine overall layout. Often this works reasonably well but sometimes it does not. In particular, when groups is used to overlay multiple frequency polygons, there is often too little head room. In the latter cases, it may be necessary to use ylim to determine an approprate viewing rectangle for the plot.

Examples

```
freqpolygon(~age | substance, data=HELPrct, v=35)
freqpolygon(~age, data=HELPrct, labels=TRUE, type='count')
freqpolygon(~age | substance, data=HELPrct, groups=sex)
freqpolygon(~age | substance, data=HELPrct, groups=sex, ylim=c(0,0.11))
## comparison of histogram and frequency polygon
histogram(~eruptions, faithful, type='density', width=.5)
ladd( panel.freqpolygon(faithful$eruptions, width=.5 ))
```

FunctionsFromData   *Create function from data*

Description

These functions create mathematical functions from data, by smoothing, splining, or linear combination (fitting). Each of them takes a formula and a data frame as an argument

Usage

```
spliner(formula, data = NULL, method = "fmm", monotonic = FALSE)

connector(formula, data = NULL, method = "linear")

smoother(formula, data, span = 0.5, degree = 2, ...)

linearModel(formula, data, ...)
```

Arguments

formula	a formula. Only one quantity is allowed on the left-hand side, the output quantity
data	a data frame
method	a method for splining. See [spline](#).
monotonic	a TRUE/FALSE flag specifying whether the spline should respect monotonicity in the data
span	parameter to smoother. How smooth it should be.
degree	parameter to smoother. 1 is locally linear, 2 is locally quadratic.
...	additional arguments to [loess](#) or [lm](#)

Details

These functions use data to create a mathematical, single-valued function of the inputs. All return a function whose arguments are the variables used on the right-hand side of the formula. If the formula involves a transformation, e.g. sqrt(age) or log(income), only the variable itself, e.g. age or income, is an argument to the function.

linearModel takes a linear combination of the vectors specified on the right-hand side. It differs from project in that linearModel returns a function whereas project returns the coefficients. NOTE: An intercept term is not included unless that is explicitly part of the formula with +1. This conflicts with the standard usage of formulas as found in lm. Another option for creating such functions is to combine [lm](#) and [makeFun](#).

spliner and connector currently work for only one input variable.

See Also

[project](#) method for formulas

Examples

```
if (require(mosaicData)) {
data(CPS85)
f <- smoother(wage ~ age, span=.9, data=CPS85)
f(40)
derivf <- D(f(age) ~ age)
derivf(40)
g <- linearModel(log(wage) ~ age + educ + 1, data=CPS85)
g(age=40, educ=12)
# an alternative way to define g (Note: + 1 is the default for lm().)
g2 <- makeFun(lm(log(wage) ~ age + educ, data=CPS85))
g2(age=40, educ=12)
dgdeduc <- D(g(age=age, educ=educ) ~ educ)
dgdeduc(age=40, educ=12)
x<-1:5; y=c(1, 2, 4, 8, 8.2)
f1 <- spliner(y ~ x)
f1(x=8:10)
f2 <- connector(x~y)
}
```

---

getVarFormula   *Extract data from a data frame using a formula interface*

Description

Uses the full model syntax.

Usage

```
getVarFormula(formula, data = parent.frame(), intercept = FALSE)
```

Arguments

formula	a formula. The right-hand side selects variables; the left-hand side, if present, is used to set row names. A . on the right-hand side indicates to use all variables not in the LHS.
data	a data frame
intercept	a logical indicating whether to include the intercept in the model default: FALSE (no intercept)

Examples

```
getVarFormula( ~ wt + mpg, data = mtcars)
```

---

googleMap   *Display a point on earth on a Google Map*

Description

Creates a URL for Google Maps for a particular latitude and longitude position.

Usage

```
googleMap(latitude, longitude, position = NULL, zoom = 12,
  maptype = c("roadmap", "satellite", "terrain", "hybrid"), mark = FALSE,
  radius = 0, browse = TRUE, ...)
```

Arguments

`latitude, longitude`	
	vectors of latitude and longitude values
`position`	a data frame containing latitude and longitude positions
`zoom`	zoom level for initial map (1-20)
`maptype`	one of `'roadmap'`, `'satellite'`, `'terrain'`, and `'hybrid'`
`mark`	a logical indicating whether the location should be marked with a pin
`radius`	a vector of radii of circles centered at position that are displayed on the map
`browse`	a logical indicating whether the URL should be browsed (else only returned as a string)
`...`	additional arguments passed to `browseURL`

Value

a string containing a URL. Optionally, as a side-effect, the URL is visited in a browser

See Also

[deg2rad](), [latlon2xyz]() and [rgeo]().

Examples

```
## Not run:
googleMap(40.7566, -73.9863, radius=1)    # Times Square
googleMap(position=rgeo(2), radius=1)     # 2 random locations

## End(Not run)
```

---

gwm   *Groupwise models Construct a model based on groupwise means or proportions*

Description

Groupwise models

Construct a model based on groupwise means or proportions

Usage

```
gwm(formula, data = parent.frame(), drop = FALSE, ...)
```

Arguments

formula
: A formula. The left-hand side specifies the response variable over which the mean or proportion will be taken. The right-hand side gives the explanatory variables, separated by +. Means or proportions are computed for every combination of the levels of the explanatory variables.

data
: A data frame in which to evaluate variables in formula. If not specified, variables will be taken from the current environment.

drop
: Logical flag indicating whether to drop unoccupied groups. Default FALSE. NOT YET IMPLEMENTED.

...
: Additional arguments; currently ignored.

Details

gwm (groupwise model) is a sort of training function for lm, meant to provide a basis for discussing inference and introducing resampling in a simple, intuitive setting of groupwise means or proportions. lm provides a better, more general facility. When using lm to recreate the results of gwm, include all the interaction terms (i.e., use * instead of +) and remove the intercept term. See the examples.

Value

mm returns an object of class groupwiseModel. The functions fitted.values, residuals, coefficients, and summary are useful for extracting various features of the value returned by mm

See Also

[lm](lm), [do](do)

Examples

```
gwm( wage ~ sex, data=CPS85 )
gwm( wage ~ sex + married, data = CPS85 )
# The same model, fit using lm() instead
lm( wage ~ sex * married - 1, data = CPS85)
do(5) * gwm( wage ~ sex + married, data = resample(CPS85))
mod <- gwm( width ~ domhand, data=KidsFeet)
summary(mod)
resid(mod)
fitted(mod)
```

---

inferArgs   *Infer arguments*

Description

The primary purpose is for inferring argument settings from names derived from variables occurring in a formula. For example, the default use is to infer limits for variables without having to call them `xlim` and `ylim` when the variables in the formula have other names. Other uses could easily be devised by specifying different `variants`.

Usage

```
inferArgs(vars, dots, defaults = alist(xlim = , ylim = , zlim = ),
  variants = c(".lim", "lim"))
```

Arguments

vars	a vector of variable names to look for
dots	a named list of argument values
defaults	named list or alist of default values for limits
variants	a vector of optional postfixes for limit-specifying variable names

Value

a named list or alist of limits. The names are determined by the names in `defaults`.

If multiple `variants` are matched, the first is used.

Examples

```
inferArgs(c('x','u','t'), list(t=c(1,3), x.lim=c(1,10), u=c(1,3), u.lim=c(2,4)))
inferArgs(c('x','u'), list(u=c(1,3)), defaults=list(xlim=c(0,1), ylim=NULL))
```

---

inspect   *Inspect objects*

Description

Print a short summary of the contents of an object. Most useful as a way to get a quick overview of the variables in data frame.

Usage

```
inspect(object, ...)

## S3 method for class 'list'
inspect(object, max.level = 2, ...)

## S3 method for class 'character'
inspect(object, ...)

## S3 method for class 'logical'
inspect(object, ...)

## S3 method for class 'numeric'
inspect(object, ...)

## S3 method for class 'factor'
inspect(object, ...)

## S3 method for class 'POSIXt'
inspect(object, ...)

## S3 method for class 'data.frame'
inspect(object, select = TRUE,
  digits = getOption("digits", 3), ...)

## S3 method for class 'inspected_data_frame'
print(x, digits = NULL, ...)
```

Arguments

object	a data frame or a vector
...	additional arguments passed along to specific methods
max.level	an integer giving the depth to which lists should be expanded
select	a logical, character (naming variables), or numeric vector or a function used to select variables to which FUN is applied. If a function, it should take a vector as input and return a single logical. See examples here and at link{dfapply}.
digits	and integer giving the number of digits to display
x	an object

Examples

```
inspect(Births78)
inspect(Births78, is.numeric)
```

---

integrateODE   _Integrate ordinary differential equations_

Description

A formula interface to integration of an ODE with respect to "t"

Usage

```
integrateODE(dyn, ..., tdur)
```

Arguments

dyn   a formula specifying the dynamics, e.g. dx ~ -a*x for $dx/dt = -ax$.

tdur   the duration of integration. Or, a list of the form list(from=5,to=10,dt=.001)

...   arguments giving additional formulas for dynamics in other variables, assignments of parameters, and assignments of initial conditions

Details

The equations must be in first-order form. Each dynamical equation uses a formula interface with the variable name given on the left-hand side of the formula, preceded by a d, so use dx~-k*x for exponential decay. All parameters (such as k) must be assigned numerical values in the argument list. All dynamical variables must be assigned initial conditions in the argument list. The returned value will be a list with one component named after each dynamical variable. The component will be a spline-generated function of t.

Value

a list with splined function of time for each dynamical variable

Examples

```
soln = integrateODE(dx~r*x*(1-x/k), k=10, r=.5, tdur=20, x=1)
soln$x(10)
soln$x(30) # outside the time interval for integration
plotFun(soln$x(t)~t, tlim=range(0,20))
soln2 = integrateODE(dx~y, dy~-x, x=1, y=0, tdur=10)
plotFun(soln2$y(t)~t, tlim=range(0,10))
# SIR epidemic
epi = integrateODE(dS~ -a*S*I, dI ~ a*S*I - b*I, a=0.0026, b=.5, S=762, I=1, tdur=20)
```

---

is.wholenumber   *Check for whole number values*

Description

Unlike `is.integer`, which checks the type of argument is `integer`, this function checks whether the value of the argument is an integer (within a specified tolerance).

Usage

```
is.wholenumber(x, tol = .Machine$double.eps^0.5)
```

Arguments

x   a vector

tol   a numeric tolerance

Details

This function is borrowed from the examples for `is.integer`

Value

a logical vector indicating whether x has a whole number value

Examples

```
is.wholenumber(1)
all(is.wholenumber(rbinom(100,10,.5)))
is.wholenumber((1:10)/2)
```

---

joinFrames   *Join data frames*

Description

Join data frames

Usage

```
joinFrames(...)

joinTwoFrames(left, right)
```

Arguments

`left, right`	data frames
`...`	data frames to be joined

Value

a data frame containing columns from each of data frames being joined.

`ladd`	*Add to Lattice Plots*

Description

Simplified lattice plotting by adding additional elements to existing plots.

Usage

```
ladd(x, data = NULL, ..., plot = trellis.last.object())
```

Arguments

`x`	callable graphical element to be added to a panel or panels in a lattice plot
`data`	a list containing objects that can be referred to in `x`. Panel functions also have access to the data already used in the panel by the underlying lattice plot. See [layer](#) for details.
`...`	additional arguments passed to [layer](#).
`plot`	a lattice plot to add to. Defaults to previous lattice plot.

Details

`ladd` is a wrapper around [layer](#) that simplifies certain common plotting additions. The same caveats that apply to that function apply here as well. In particular, `ladd` uses non-standard evaluation. For this reason care must be taken if trying to use `ladd` within other functions and the use of `data` may be required to pass information into the environment in which `x` will be evaluated.

Value

a trellis object

Author(s)

Randall Pruim (<rpruim@calvin.edu>)

See Also

[layer](#)

Examples

```
p <- xyplot(rnorm(100) ~rnorm(100))
print(p)
ladd(panel.abline(a=0,b=1))
ladd(panel.abline(h=0,col='blue'))
ladd(grid.text('Hello'))
ladd(grid.text(x=.95,y=.05,'text here',just=c('right','bottom')))
q <- xyplot(rnorm(100) ~rnorm(100)|factor(rbinom(100,4,.5)))
q <- update(q, layout=c(3,2))
ladd(panel.abline(a=0,b=1), plot=q)
ladd(panel.abline(h=0,col='blue'))
ladd( grid.text("(2,1)",gp=gpar(cex=3,alpha=.5)), columns=2, rows=1)
ladd( grid.text("p5",gp=gpar(cex=3,alpha=.5)), packets=5)
q
ladd( grid.text(paste(current.column(), current.row(),sep=','), gp=gpar(cex=3,alpha=.5)) )
histogram( ~eruptions, data=faithful )
# over would probably be better here, but the demonstrates what under=TRUE does.
ladd(panel.densityplot(faithful$eruptions, lwd=4), under=TRUE)
```

---

linear.algebra	*Functions for teaching linear algebra.*

Description

These functions provide a formula based interface to the construction of matrices from data and for
fitting. You can use them both for numerical vectors and for functions of variables in data frames.
These functions are intended to support teaching basic linear algebra with a particular connection
to statistics.

Usage

```
mat(A, data = parent.frame())

singvals(A, data = parent.frame())
```

Arguments

A	a formula. In `mat` and `singvals`, only the right-hand side is used.
data	a data frame from which to pull out numerical values for the variables in the formula
...	additional arguments (currently ignored)
	`mat` returns a model matrix
	To demonstrate singularity, use `singvals`.

Value

mat returns a matrix

singvals gives singular values for each column in the model matrix

See Also

[project](#)

[linearModel](#), which returns a function.

Examples

```
a <- c(1,0,0); b <- c(1,2,3); c <- c(4,5,6); x <- rnorm(3)
# Formula interface
mat(~a+b)
mat(~a+b+1)
if (require(mosaicData)) {
mat(~length+sex, data=KidsFeet)
singvals(~length*sex*width, data=KidsFeet)
}
```

---

logical2factor *Turn logicals into factors; leave other things alone*

Description

Turn logicals into factors; leave other things alone

Usage

```
logical2factor(x, ...)

## Default S3 method:
logical2factor(x, ...)

## S3 method for class 'data.frame'
logical2factor(x, ...)
```

Arguments

x   a vector or data frame

...   additional arguments (currently ignored)

Value

If x is a vector either x or the result of converting x into a factor with levels TRUE and FALSE (in that order); if x is a data frame, a data frame with all logicals converted to factors in this manner.

---

logit *Logit and inverse logit functions*

Description

Logit and inverse logit functions

Usage

```
logit(x)

ilogit(x)
```

Arguments

x   a numeric vector

Value

For `logit` the value is

$$log(x/(1 - x))$$

For `ilogit` the value is

$$exp(x)/(1 + exp(x))$$

Examples

```
p <- seq(.1, .9, by=.10)
l <- logit(p); l
ilogit(l)
ilogit(l) == p
```

MAD *All pairs mean and sum of absolute differences*

Description

All pairs mean and sum of absolute differences

Usage

```
MAD(x, ..., data = NULL, groups = NULL, na.rm = getOption("na.rm", FALSE))

SAD(x, ..., data = NULL, groups = NULL, na.rm = getOption("na.rm", FALSE))
```

Arguments

x	a numeric vector or a formula.
...	additional arguments passed through to MAD_ or SAD_. If x is a formala, ... should include an argument named data if the intent is to interpret the formala in a data frame.
data	a data frame in which to evaluate formulas (or bare names). Note that the default is data=parent.frame(). This makes it convenient to use this function interactively by treating the working envionment as if it were a data frame. But this may not be appropriate for programming uses. When programming, it is best to use an explicit data argument – ideally supplying a data frame that contains the variables mentioned.
groups	a grouping variable, typically a name of a variable in data
na.rm	a logical indicating whether NAs should be removed before calculaing.

Value

the mean or sum of the absolute differences between each pair of values in c(x,...).

See Also

link{mad}, MAD_

Examples

```
SAD(1:3)
MAD(1:3)
MAD(~eruptions, data=faithful)
```

MAD_   *All pairs mean and sum of absolute differences*

Description

All pairs mean and sum of absolute differences

Usage

```
MAD_(x, ..., na.rm = getOption("na.omit", FALSE))

SAD_(x, ..., na.rm = getOption("na.omit", FALSE))
```

Arguments

x	a numeric vector or a formula.
...	additional arguments appended to x
na.rm	a logical indicating whether NAs should be removed before calculaing.

Value

the mean or sum of the absolute differences between each pair of values in `c(x,...)`.

See Also

[mad](#)

---

maggregate   *Aggregate for mosaic*

Description

Compute function on subsets of a variable in a data frame.

Usage

```
maggregate(formula, data = parent.frame(), FUN, groups = NULL, subset,
  drop = FALSE, ..., .format = c("default", "table", "flat"),
  .overall = mosaic.par.get("aggregate.overall"), .multiple = FALSE,
  .name = deparse(substitute(FUN)), .envir = parent.frame())
```

Arguments

formula	a formula. Left side provides variable to be summarized. Right side and condition describe subsets. If the left side is empty, right side and condition are shifted over as a convenience.
data	a data frame. Note that the default is data=parent.frame(). This makes it convenient to use this function interactively by treating the working envionment as if it were a data frame. But this may not be appropriate for programming uses. When programming, it is best to use an explicit data argument – ideally supplying a data frame that contains the variables mentioned in formula.
FUN	a function to apply to each subset
groups	grouping variable that will be folded into the formula (if there is room for it). This offers some additional flexibility in how formulas can be specified.
subset	a logical indicating a subset of data to be processed.
drop	a logical indicating whether unused levels should be dropped.
.format	format used for aggregation. "default" and "flat" are equivalent.
.overall	currently unused
.multiple	a logical indicating whether FUN returns multiple values
.name	a name used for the resulting object
.envir	an environment in which to evaluate expressions
...	additional arguments passed to FUN

Value

a vector

Examples

```
if (require(mosaicData)) {
maggregate( cesd ~ sex, HELPrct, FUN=mean )
# using groups instead
maggregate( ~ cesd, groups = sex, HELPrct, FUN=sd )
# the next four all do the same thing
maggregate( cesd ~ sex + homeless, HELPrct, FUN=mean )
maggregate( cesd ~ sex | homeless, HELPrct, FUN=sd )
maggregate( ~ cesd | sex , groups= homeless, HELPrct, FUN=sd )
maggregate( cesd ~ sex, groups = homeless, HELPrct, FUN=sd )
# this is unusual, but also works.
maggregate( cesd ~ NULL , groups = sex, HELPrct, FUN=sd )
}
```

---

makeColorscheme   *Create a color generating function from a vector of colors*

Description

Create a color generating function from a vector of colors

Usage

```
makeColorscheme(col)
```

Arguments

col   a vector of colors

Value

a function that generates a vector of colors interpolated among the colors in col

Examples

```
cs <- makeColorscheme( c('red','white','blue') )
cs(10)
cs(10, alpha=.5)
```

---

makeFun	*Create a function from a formula*

Description

Provides an easy mechanism for creating simple "mathematical" functions via a formula interface.

Usage

```
makeFun(object, ...)

## S3 method for class 'function'
makeFun(object, ..., strict.declaration = TRUE,
  use.environment = TRUE, suppress.warnings = FALSE)

## S3 method for class 'formula'
makeFun(object, ..., strict.declaration = TRUE,
  use.environment = TRUE, suppress.warnings = FALSE)

## S3 method for class 'lm'
makeFun(object, ..., transformation = NULL)

## S3 method for class 'glm'
makeFun(object, ..., type = c("response", "link"),
  transformation = NULL)

## S3 method for class 'nls'
makeFun(object, ..., transformation = NULL)

## S3 method for class 'groupwiseModel'
makeFun(object, ..., transformation = NULL)
```

Arguments

object	an object from which to create a function. This should generally be specified without naming.
...	additional arguments in the form var = val that set default values for the inputs to the function.
strict.declaration	
	if TRUE (the default), an error is thrown if default values are given for variables not appearing in the object formula.
use.environment	
	if TRUE, then variables implicitly defined in the object formula can take default values from the environment at the time makeFun is called. A warning message alerts the user to this situation, unless suppress.warnings is TRUE.
suppress.warnings	
	A logical indicating whether warnings should be suppressed.

transformation a function used to transform the response. This can be useful to invert a transformation used on the response when creating the model. If NULL, an attempt will be made to infer the transformation from the model formula. A few simple transformations (`log`, `log2`, `sqrt`) are recognized. For other transformations, `transformation` should be provided explicitly.

type   one of `'response'` (default) or `'link'` specifying scale to be used for value of function returned.

Details

The definition of the function is given by the left side of a formula. The right side lists at least one of the inputs to the function. The inputs to the function are all variables appearing on either the left or right sides of the formula. Those appearing in the right side will occur in the order specified. Those not appearing in the right side will appear in an unspecified order.

When creating a function from a model created with `lm`, `glm`, or `nls`, the function produced is a wrapper around the corresponding version of `predict`. This means that having variables in the model with names that match arguments of `predict` will lead to potentially ambiguous situations and should be avoided.

Value

a function

Examples

```
f <- makeFun( sin(x^2 * b) ~ x & y & a); f
g <- makeFun( sin(x^2 * b) ~ x & y & a, a=2 ); g
h <- makeFun( a * sin(x^2 * b) ~ b & y, a=2, y=3); h
if (require(mosaicData)) {
model <- lm( log(length) ~ log(width), data=KidsFeet)
f <- makeFun(model, transformation = exp)
f(8.4)
head(KidsFeet,1)
}

model <- lm(wage ~ poly(exper,degree=2), data=CPS85)
fit <- makeFun(model)
xyplot(wage ~ exper, data=CPS85)
plotFun(fit(exper) ~ exper, add=TRUE)
if (require(mosaicData)) {
model <- glm(wage ~ poly(exper,degree=2), data=CPS85, family=gaussian)
fit <- makeFun(model)
xyplot(wage ~ exper, data=CPS85)
plotFun(fit(exper) ~ exper, add=TRUE)
}
if (require(mosaicData)) {
model <- nls( wage ~ A + B * exper + C * exper^2, data=CPS85, start=list(A=1,B=1,C=1) )
fit <- makeFun(model)
xyplot(wage ~ exper, data=CPS85)
plotFun(fit(exper) ~ exper, add=TRUE)
}
```

```
mod <- gwm(wage ~ sector, data = CPS85)
modfun <- makeFun(mod)
modfun(sector = "prof")
```

---

makeMap   *Make a map with* ggplot2

Description

makeMap takes in two sources of data that refer to geographical regions and merges them together. Depending on the arguments passed, it returns this merged data or a ggplot object constructed with the data.

Usage

```
makeMap(data = NULL, map = NULL, key = c(key.data, key.map), key.data,
  key.map, tr.data = identity, tr.map = identity, plot = c("borders",
  "frame", "none"))
```

Arguments

data	A dataframe with regions as cases
map	An object that can be fortified to a dataframe (ex: a dataframe itself, or a SpatialPolygonsDataFrame)
key	The combination of key.data and key.map
key.data	The column name in the data that holds the unique names of each region
key.map	The column name in the map that holds the unique names of each region
tr.data	A function of the transformation to be performed to the key.data column
tr.map	A function of the transformation to be performed to the key.map column
plot	The plot desired for the output. plot = "none" returns the merged data that is the result of merging the data and map together; plot="frame" returns an empty (unplottable) ggplot object; plot = "border" (the default) returns a ggplot object with one geom_polygon layer that shows the borders of the regions.

---

mean_   *Aggregating functions*

Description

The mosaic package makes several summary statistic functions (like mean and sd) formula aware.

Usage

```
mean_(x, ..., data = NULL, groups = NULL, na.rm = getOption("na.rm",
  FALSE))

mean(x, ...)

median(x, ..., data = NULL, groups = NULL, na.rm = getOption("na.rm",
  FALSE))

range(x, ..., data = NULL, groups = NULL, na.rm = getOption("na.rm",
  FALSE))

sd(x, ..., data = NULL, groups = NULL, na.rm = getOption("na.rm", FALSE))

max(x, ..., data = NULL, groups = NULL, na.rm = getOption("na.rm", FALSE))

min(x, ..., data = NULL, groups = NULL, na.rm = getOption("na.rm", FALSE))

sum(x, ..., data = NULL, groups = NULL, na.rm = getOption("na.rm", FALSE))

IQR(x, ..., data = NULL, groups = NULL, na.rm = getOption("na.rm", FALSE))

fivenum(x, ..., data = NULL, groups = NULL, na.rm = getOption("na.rm",
  FALSE))

iqr(x, ..., data = NULL, groups = NULL, na.rm = getOption("na.rm", FALSE))

prod(x, ..., data = NULL, groups = NULL, na.rm = getOption("na.rm",
  FALSE))

sum(x, ..., data = NULL, groups = NULL, na.rm = getOption("na.rm", FALSE))

favstats(x, ..., data = NULL, groups = NULL, na.rm = TRUE)

quantile(x, ..., data = NULL, groups = NULL, na.rm = getOption("na.rm",
  FALSE))

var(x, y = NULL, na.rm = getOption("na.rm", FALSE), ..., data = NULL)
```

```
cor(x, y = NULL, ..., data = NULL)

cov(x, y = NULL, ..., data = NULL)
```

Arguments

x	a numeric vector or a formula
data	a data frame in which to evaluate formulas (or bare names). Note that the default is data=parent.frame(). This makes it convenient to use this function interactively by treating the working envionment as if it were a data frame. But this may not be appropriate for programming uses. When programming, it is best to use an explicit data argument – ideally supplying a data frame that contains the variables mentioned.
groups	a grouping variable, typically a name of a variable in data
na.rm	a logical indicating whether NAs should be removed before computing
y	a numeric vector or a formula
...	additional arguments

Details

Many of these functions mask core R functions to provide an additional formula interface. Old behavior should be unchanged. But if the first argument is a formula, that formula, together with data are used to generate the numeric vector(s) to be summarized. Formulas of the shape x ~ a or ~ x | a can be used to produce summaries of x for each subsect defined by a. Two-way aggregation can be acheived using formulas of the form x ~ a + b or x ~ a | b. See the examples.

Note

Earlier versions of these functions supported a "bare name + data frame" interface. This functionality has been removed since it was (a) ambiguous in some cases, (b) unnecessary, and (c) difficult to maintain.

Examples

```
mean(HELPrct$age)
mean( ~ age, data=HELPrct)
mean(age ~ shuffle(sex), data=HELPrct)
mean(age ~ shuffle(sex), data=HELPrct, .format="table")
# wrap in data.frame() to auto-convert awkward variable names
data.frame(mean(age ~ shuffle(sex), data=HELPrct, .format="table"))
mean(age ~ sex + substance, data=HELPrct)
mean( ~ age | sex + substance, data=HELPrct)
mean( ~ sqrt(age), data=HELPrct)
sum( ~ age, data=HELPrct)
sd(HELPrct$age)
sd( ~ age, data=HELPrct)
sd(age ~ sex + substance, data=HELPrct)
var(HELPrct$age)
var( ~ age, data=HELPrct)
```

```
var(age ~ sex + substance, data=HELPrct)
IQR(width ~ sex, data=KidsFeet)
iqr(width ~ sex, data=KidsFeet)
favstats(width ~ sex, data = KidsFeet)

cor(length ~ width, data = KidsFeet)
cov (length ~ width, data = KidsFeet)
```

---

mid	*midpoints along a sequence*

Description

Compute a vector of midpoints between values in a numeric vector

Usage

```
mid(x)
```

Arguments

x   a numeric vector

Value

a vector of length 1 less than x

Examples

```
mid(1:5)
mid((1:5)^2)
```

---

mm	*Construct a model based on groupwise means*

Description

Calculate groupwise means, presenting the result as a model in the style of lm.

Usage

```
mm(formula, data = parent.frame(), fun = mean, drop = TRUE, ...)
```

Arguments

`formula`	A formula. The left-hand side specifies the variable over which the mean will be taken. The right-hand side gives the grouping variables, separated by &.
`data`	A data frame to which the formula variables refer. If not specified, variables will be taken from the current environment.
`fun`	The function used to calculate the means. Default: `mean`.
`drop`	Logical flag indicating whether to drop unoccupied groups. Default `TRUE`. NOT YET IMPLEMENTED.
`...`	Additional arguments to be passed to the `fun` doing the calculation.

Details

`mm` is a sort of training function for `lm`, meant to provide a basis for discussing inference and introducing resampling in a simple, intuitive setting of groupwise means. `lm` provides a better, more general facility. When using `lm` to recreate the results of `mm`, include all the interaction terms, that is, use `*` instead of `&`. See the examples.

Value

`mm` returns an object of class `groupwiseModel`. The functions `fitted.values`, `residuals`, `coefficients`, and `summary` are useful for extracting various features of the value returned by `mm`

See Also

`lm`, `do`

`modelVars`	*extract predictor variables from a model*

Description

extract predictor variables from a model

Usage

```
modelVars(model)
```

Arguments

`model`	a model, typically of class `lm` or `glm`

Value

a vector of variable names

Examples

```
model <- lm( wage ~ poly(exper,degree=2), data=CPS85 )
modelVars(model)
```

---

mosaic.options	*Setting options for mosaic package functions*

Description

A mechanism for setting options in the mosaic package.

Usage

```
mosaic.options(...)

mosaic.getOption(name)

mosaic.par.set(name, value, ..., theme, warn = TRUE, strict = FALSE)

mosaic.par.get(name = NULL)

restoreLatticeOptions()

mosaicLatticeOptions()
```

Arguments

name	the name of the option being set
value	the value to which to set the option
theme	a list appropriate for a mosaic theme
warn	a logical. UNUSED at present.
strict	a logical or numeric.
...	additional arguments that are turned into a list if a list cannot be inferred from theme, name, and value.

Details

restoreLatticeOptions returns any lattice options that were changed when the mosiac package was loaded back to their pre-mosaic state.

mosaicLatticeOptions sets a number of defaults for lattice graphics.

Description

These functions convert formauls into standard shapes, including by incorporating a groups argument.

Usage

```
mosaic_formula(formula, groups = NULL, envir = parent.frame(),
  max.slots = 3, groups.first = FALSE)

mosaic_formula_q(formula, groups = NULL, max.slots = 3,
  groups.first = FALSE, ...)
```

Arguments

formula	a formula
groups	a name used for grouping
envir	the environment in which the resulting formula may be evaluated. May also be NULL, a list, a data frame, or a pairlist.
max.slots	an integer specifying the maximum number of slots for the resulting formula. An error results from trying to create a formula that is too complex.
groups.first	a logical indicating whether groups should be inserted ahead of the condition (else after).
...	additional arguments (currently ignored)

Details

mosaic_formula_q uses nonstandard evaluation of groups that may be necessary for use within other functions. mosaic_formula is a wrapper around mosaic_formula_q and quotes groups before passing it along.

Examples

```
mosaic_formula( ~ x | z )
mosaic_formula( ~ x, groups=g )
mosaic_formula( y ~ x, groups=g )
# this is probably not what you want for interactive use.
mosaic_formula_q( y ~ x, groups=g )
# but it is for programming
foo <- function(x, groups=NULL) {
    mosaic_formula_q(x, groups=groups, envir=parent.frame())
}
foo( y ~ x , groups = g)
```

---

mPlot   *Interactive plotting*

Description

These functions provide a menu selection system (via **manipulate**) so that different aspects of a plot can be selected interactively. The **ggplot2** or **lattice** command for generating the plot currently being displayed can be copied to the console, whence it can be copied to a document for later direct, non-interactive use.

Usage

```
mPlot(data, format, default = format, system = c("lattice", "ggplot2"),
  show = FALSE, title = "", data_text = expr_text(data), ...)

mMap(data, default = "map", system = "ggplot2", show = FALSE,
  title = title, data_text = expr_text(data), ...)

mScatter(data, default = c("scatter", "jitter", "boxplot", "violin", "line"),
  system = c("lattice", "ggplot2"), show = FALSE, title = "",
  data_text = lazyeval::expr_text(data))

mUniplot(data, default = c("histogram", "density", "frequency polygon",
  "ASH plot"), system = c("lattice", "ggplot2"), show = FALSE, title = "",
  data_text = expr_text(data))
```

Arguments

data	a data frame containing the variables that might be used in the plot. Note that for maps, the data frame must contain coordinates of the polygons comprising the map and a variable for determining which corodiantes are part of the same region. See `sp2df` for one way to create such a data frame. Typically `merge` will be used to combine the map data with some auxilliary data to be displayed as fill color on the map, although this is not necessary if all one wants is a map.
format	a synonym for `default`.
default	default type of plot to create; one of `"scatter"`, `"jitter"`, `"boxplot"`, `"violin"`, `"histogram"`, `"density"`, `"frequency polygon"`, `"xyplot"`, or `"map"`. Unique prefixes suffice.
system	which graphics system to use (initially) for plotting (**ggplot2** or **lattice**). A check box will allow on the fly change of plotting system.
show	a logical, if TRUE, the code will be displayed each time the plot is changed.
title	a title for the plot
data_text	A text string describing the data. It must be possible to recover the data from this string using `eval`(). Typically users will not need to modify this from the default value.
...	additional arguments

Details

Only `mPlot` is required by end users. The other plotting functions are dispatched based on the value of `default`. Furthermore, `mplot` will dispatch mPlot when provided a data frame.

Currently maps are only supported in **ggplot2** and not in **lattice**.

Value

Nothing. Just for side effects.

Examples

```
## Not run:
mPlot(HELPrct, format="scatter")
mPlot(HELPrct, format="density")

## End(Not run)
```

---

mplot   *Generic plotting*

Description

Generic function plotting for R objects. Currently plots exist for `data.frames`, `lms`, (including `glms`).

Usage

```
mplot(object, ...)

## Default S3 method:
mplot(object, ...)

## S3 method for class 'lm'
mplot(object, which = c(1:3, 7), system = c("lattice",
  "ggplot2", "base"), ask = FALSE, multiplot = "package:gridExtra" %in%
  search(), par.settings = theme.mosaic(), level = 0.95,
  title = paste("model: ", deparse(object$call), "\n"), rows = TRUE, ...)

## S3 method for class 'data.frame'
mplot(object, format, default = format,
  system = c("lattice", "ggplot2"), show = FALSE, title = "", ...)

## S3 method for class 'summary.lm'
mplot(object, system = c("lattice", "ggplot2"),
  level = 0.95, par.settings = trellis.par.get(), rows = TRUE, ...)

## S3 method for class 'TukeyHSD'
```

```
mplot(object, system = c("lattice", "ggplot2"),
  ylab = "", xlab = "difference in means", title = paste0(attr(object,
  "conf.level") * 100, "% family-wise confidence level"),
  par.settings = trellis.par.get(), order = c("asis", "pval", "difference"),
  ...)
```

Arguments

`object`	an R object from which a plot will be constructed.
`...`	additional arguments. If `object` is an `lm`, subsets of these arguments are passed to `grid.arrange` and to the **lattice** plotting routines; in particular, `nrow` and `ncol` can be used to control the number of rows and columns used.
`which`	a numeric vector used to select from 7 potential plots
`system`	which graphics system to use (initially) for plotting (**ggplot2** or **lattice**). A check box will allow on the fly change of plotting system.
`ask`	if TRUE, each plot will be displayed separately after the user responds to a prompt.
`multiplot`	if TRUE and `ask == FALSE`, all plots will be displayed together.
`par.settings`	**lattice** theme settings
`level`	a confidence level
`title`	title for plot
`rows`	rows to show. This may be a numeric vector, TRUE (for all rows), or a character vector of row names.
`format, default`	
	default type of plot to create; one of `"scatter"`, `"jitter"`, `"boxplot"`, `"violin"`, `"histogram"`, `"density"`, `"frequency polygon"`, or `"map"`. Unique prefixes suffice.
`show`	a logical, if TRUE, the code will be displayed each time the plot is changed.
`ylab`	label for y-axis
`xlab`	label for x-axis
`order`	one of `"pval"`, `"diff"`, or `"asis"` determining the order of the `pair` factor, which determines the order in which the differences are displayed on the plot.
`data`	a data frame containing the variables that might be used in the plot.

Details

The method for models (lm and glm) is still a work in progress, but should be useable for relatively simple models. When the results for a logistic regression model created with [glm](...)() are satisfactory will depend on the format and structure of the data used to fit the model.

Value

Nothing. Just for side effects.

Examples

```
mplot( lm( width ~ length * sex, data=KidsFeet) )
mplot( lm( width ~ length * sex, data=KidsFeet), rows=2:3, which=7 )
## Not run:
mplot( HELPrct )
mplot( HELPrct, "histogram" )

## End(Not run)
mplot(summary(lm(width ~ length * sex, data=KidsFeet)), system="ggplot")
mplot(summary(lm(width ~ length * sex, data=KidsFeet)), rows=c("sex", "length"))
mplot(summary(lm(width ~ length * sex, data=KidsFeet)), rows=TRUE)
mplot(TukeyHSD( lm(age ~ substance, data=HELPrct) ) )
mplot(TukeyHSD( lm(age ~ substance, data=HELPrct) ), system="ggplot2" )
```

MSPE   *Mean Squared Prediction Error*

Description

A one-step calculation of mean square prediction error

Usage

```
MSPE(model, data, LL = TRUE)
```

Arguments

model	a model produced by lm, glm, or gwm.
data	a data frame.
LL	if TRUE, for categorical responses replace mean square error with minus mean log likelihood

Details

For categorical responses, the mean square prediction error is not ideal. Better to use the likelihood. LL = TRUE (the default) turns the calculation into the mean log likelihood per case, negated so that large values mean poor predictions

Examples

```
HELP <- HELPrct %>% sample_frac(.3)
MSPE( gwm( age ~ sex, data = HELP), HELPrct)
MSPE( gwm( age ~ 1, data = HELP), HELPrct)
MSPE( gwm( age ~ sex + homeless, data = HELP), HELPrct)
MSPE( gwm( sex ~ 1, data = HELP), HELPrct)
MSPE( gwm( sex ~ homeless, data = HELP), HELPrct)
MSPE( gwm( sex ~ homeless + substance, data = HELP), HELPrct)
```

---

mUSMap   *Make a US map with* ggplot2

Description

mUSMap takes in one dataframe that includes information about different US states. It merges this
dataframe with a dataframe that includes geographical coordinate information. Depending on the
arguments passed, it returns this data or a ggplot object constructed with the data.

Usage

```
mUSMap(data = NULL, key, fill = NULL, plot = c("borders", "frame",
  "none"), style = c("compact", "real"))
```

Arguments

data	A dataframe with US states as cases
key	The column name in the data that holds the unique names of each state
fill	A variable in the data used to specify the fill color of states in the map (note: if fill is not null, then plot cannot be set to "none")
plot	The plot desired for the output. plot = "none" returns the merged data that is the result of merging the data and the dataframe with the geographical coordinate information; plot = "frame" returns an empty (unplottable) ggplot object; plot = "border" (the default) returns a ggplot object with one geom_polygon layer that shows the borders of the states
style	The style in which to display the map. compact gives a polyconic projection with Alaska and Hawaii on the lower left corner; real gives the real size and position of all states without any projection.

Examples

```
USArrests2 <- USArrests %>% mutate(state = row.names(.))
mUSMap(USArrests2, key="state", fill = "UrbanPop")
```

---

mWorldMap   *Make a world map with* ggplot2

Description

mWorldMap takes in one dataframe that includes information about different countries. It merges
this dataframe with a dataframe that includes geographical coordinate information. Depending on
the arguments passed, it returns this data or a ggplot object constructed with the data.

Usage

```
mWorldMap(data = NULL, key = NA, fill = NULL, plot = c("borders",
  "frame", "none"))
```

Arguments

data	A dataframe with countries as cases
key	The column name in the data that holds the unique names of each country
fill	A variable in the data used to specify the fill color of countries in the map (note: if fill is not null, then plot cannot be set to "none")
plot	The plot desired for the output. plot = "none" returns the merged data that is the result of merging the data and the dataframe with the geographical coordinate information; plot = "frame" returns an empty (unplottable) ggplot object; plot = "border" (the default) returns a ggplot object with one geom_polygon layer that shows the borders of the countries

Examples

```
## Not run:
gdpData <- CIAdata("GDP")        # load some world data

mWorldMap(gdpData, key="country", fill="GDP")

gdpData <- gdpData %>% mutate(GDP5 = ntiles(-GDP, 5, format="rank"))
mWorldMap(gdpData, key="country", fill="GDP5")

mWorldMap(gdpData, key="country", plot="frame") +
geom_point()

mergedData <- mWorldMap(gdpData, key="country", plot="none")

ggplot(mergedData, aes(x=long, y=lat, group=group, order=order)) +
geom_polygon(aes(fill=GDP5), color="gray70", size=.5) + guides(fill=FALSE)

## End(Not run)
```

---

named   *List extraction*

Description

These functions create subsets of lists based on their names

Usage

```
named(l)

unnamed(l)

named_among(l, n)
```

Arguments

l	a list
n	a vector of character strings (potential names)

Value

a sublist of `l` determined by `names(l)`

---

nice_names	*Nice names*

Description

Convert a character vector into a similar character vector that would work better as names in a data frame by avoiding certain awkward characters

Usage

```
nice_names(x, unique = TRUE)
```

Arguments

x	a character vector
unique	a logical indicating whether returned values should be uniquified.

Value

a character vector

Examples

```
nice_names( c("bad name", "name (crazy)", "a:b", "two-way") )
```

---

ntiles   *Create vector based on roughly equally sized groups*

Description

Create vector based on roughly equally sized groups

Usage

```
ntiles(x, n = 3, format = c("rank", "interval", "mean", "median", "center",
  "left", "right"), digits = 3)
```

Arguments

x	a numeric vector
n	(approximate) number of quantiles
format	a specification of desired output format.
digits	desired number of digits for labeling of factors.

Value

a vector. The type of vector will depend on `format`.

Examples

```
if (require(mosaicData)) {
tally( ~ ntiles(age, 4), data=HELPrct)
tally( ~ ntiles(age, 4, format="center"), data=HELPrct)
tally( ~ ntiles(age, 4, format="interval"), data=HELPrct)
tally( ~ ntiles(age, 4, format="left"), data=HELPrct)
tally( ~ ntiles(age, 4, format="right"), data=HELPrct)
tally( ~ ntiles(age, 4, format="mean"), data=HELPrct)
tally( ~ ntiles(age, 4, format="median"), data=HELPrct)
bwplot( i2 ~ ntiles(age, n=5, format="interval"), data=HELPrct)
}
```

---

numD   *Numerical Derivatives*

Description

Constructs the numerical derivatives of mathematical expressions

Usage

```
numD(formula, ..., .hstep = NULL, add.h.control = FALSE)

setInterval(C, wrt, h)

setCorners(C, var1, var2, h)

dfdx(.function, .wrt, .hstep)

d2fdxdy(.function, .var1, .var2, .hstep)

d2fdx2(.function, .wrt, .hstep)

numerical.first.partial(f, wrt, h, av)

numerical.second.partial(f, wrt, h, av)

numerical.mixed.partial(f, var1, var2, h, av)
```

Arguments

`formula`	a mathematical expression (see examples and [plotFun](#))
`...`	additional parameters, typically default values for mathematical parameters
`.hstep`	numerical finite-difference step (default is 1e-6 or 1e-4 for first and second-order derivatives, respectively)
`add.h.control`	arranges the returned function to have a `.hstep` argument that cann be used to demonstrate convergence and error
`C`	list of arguments for evaluating the function at the "center" point
`wrt`	character string naming the variable with respect to which differentiation is to be done
`h`	the finite-difference step size
`var1`	character string naming the first variable with respect to which differentiation is to be done
`var2`	character string naming the second variable with respect to which differentiation is to be done
`.function`	function to be differentiated
`.wrt`	character string naming the variable with respect to which differentiation is to be done
`.var1`	character string naming the first variable with respect to which differentiation is to be done
`.var2`	character string naming the second variable with respect to which differentiation is to be done
`f`	function to differentiate
`av`	arguments to the function calling this
`.step`	the finite-difference step size

Details

Uses a simple finite-difference scheme to evaluate the derivative. The function created will not contain a formula for the derivative. Instead, the original function is stored at the time the derivative is constructed and that original function is re-evaluated at the finitely-spaced points of an interval. If you redefine the original function, that won't affect any derivatives that were already defined from it. Numerical derivatives, particularly high-order ones, are unstable. The finite-difference parameter `.hstep` is set, by default, to give reasonable results for first- and second-order derivatives. It's tweaked a bit so that taking a second derivative by differentiating a first derivative will give reasonably accurate results. But, if taking a second derivative, much better to do it in one step to preserve numerical accuracy.

Value

a function implementing the derivative as a finite-difference approximation

Numerical partials

These functions are not indended for direct use. They just package up the numerical differentiation process to make functions returned by numD and D easier to read.

Note

WARNING: In the expressions, do not use variable names beginning with a dot, particularly `.f` or `.h`

Helper function for numD for unmixed partials

Helper function for numD for mixed partials

Helper function for numD for first-order derivs.

Helper function for numD for second-order mixed partials

Helper function for numD for second-order derivs

Not for direct use. This just packages up the numerical differentiation process to make functions returned by numD and D easier to read.

Not for direct use. This just packages up the numerical differentiation process to make functions returned by numD and D easier to read.

Author(s)

Daniel Kaplan (<kaplan@macalester.edu>)

See Also

D, symbolicD, makeFun, antiD, plotFun

Examples

```
g = numD( a*x^2 + x*y ~ x, a=1)
g(x=2,y=10)
gg = numD( a*x^2 + x*y ~ x&x, a=1)
gg(x=2,y=10)
ggg = numD( a*x^2 + x*y ~ x&y, a=1)
ggg(x=2,y=10)
h = numD( g(x=x,y=y,a=a) ~ y, a=1)
h(x=2,y=10)
f = numD( sin(x)~x, add.h.control=TRUE)
plotFun( f(3,.hstep=h)~h, hlim=range(.00000001,.000001))
ladd( panel.abline(cos(3),0))
```

---

n_missing   *counting missing elements*

Description

counting missing elements

Usage

```
n_missing(..., type = c("any", "all"))
```

Arguments

...	vectors of equal length to be checked in parallel for missing values.
type	one of "any" (default) or "all".

Examples

```
if (require(NHANES)) {
  tally( ~ is.na(Height) + is.na(Weight), data = NHANES, margins = TRUE)
  NHANES %>%
    summarise(
      mean.wt = mean(Weight, na.rm = TRUE),
      missing.Wt = n_missing(Weight),
      missing.WtAndHt = n_missing(Weight, Height, type = "all"),
      missing.WtOrHt = n_missing(Weight, Height, type = "any")
      )
    }
```

---

orrr   *Odds Ratio and Relative Risk for 2 x 2 Contingency Tables*

Description

This function calculates the odds ratio and relative risk for a 2 x 2 contingency table and a confidence interval (default conf.level is 95 percent) for the each estimate. x should be a matrix, data frame or table. "Successes" should be located in column 1 of x, and the treatment of interest should be located in row 2. The odds ratio is calculated as (Odds row 2) / (Odds row 1). The confidence interval is calculated from the log(OR) and backtransformed.

Usage

```
orrr(x, conf.level = 0.95, verbose = !quiet, quiet = TRUE, digits = 3,
  relrisk = FALSE)

oddsRatio(x, conf.level = 0.95, verbose = !quiet, quiet = TRUE,
  digits = 3)

relrisk(x, conf.level = 0.95, verbose = !quiet, quiet = TRUE,
  digits = 3)

## S3 method for class 'oddsRatio'
print(x, digits = 4, ...)

## S3 method for class 'relrisk'
print(x, digits = 4, ...)

## S3 method for class 'oddsRatio'
summary(object, digits = 4, ...)

## S3 method for class 'relrisk'
summary(object, digits = 4, ...)
```

Arguments

x	a 2 X 2 matrix, data frame or table of counts
conf.level	the confidence interval level
verbose	a logical indicating whether verbose output should be displayed
quiet	a logical indicating whether verbose outoput should be supressed
digits	number of digits to display
relrisk	a logical indicating whether the relative risk should be returned instead of the odds ratio
...	additional arguments
object	an R object to print or summarise. Here an object of class "oddsRatio" or "relrisk".

Value

an odds ratio or relative risk. If verpose is true, more details and the confidence intervals are displayed.

Author(s)

Kevin Middleton (<kmm@csusb.edu>); modified by R Pruim.

See Also

[chisq.test](), [fisher.test]()

Examples

```
M1 <- matrix(c(14, 38, 51, 11), nrow = 2)
M1
oddsRatio(M1)

M2 <- matrix(c(18515, 18496, 1427, 1438), nrow = 2)
rownames(M2) <- c("Placebo", "Aspirin")
colnames(M2) <- c("No", "Yes")
M2
oddsRatio(M2)
oddsRatio(M2, verbose=TRUE)
relrisk(M2, verbose=TRUE)
if (require(mosaicData)) {
relrisk(tally(~ homeless + sex, data=HELPrct) )
do(3) * relrisk( tally( ~ homeless + shuffle(sex), data=HELPrct) )
}
```

---

panel.levelcontourplot

Lattice plot that draws a filled contour plot

Description

Used within plotFun

Usage

```
panel.levelcontourplot(x, y, z, subscripts = 1, at, shrink, labels = TRUE,
  label.style = c("mixed", "flat", "align"), contour = FALSE,
  region = TRUE, col = add.line$col, lty = add.line$lty,
  lwd = add.line$lwd, border = "transparent", ...,
  col.regions = regions$col, filled = TRUE, alpha.regions = regions$alpha)
```

Arguments

x	x on a grid
y	y on a grid
z	zvalues for the x and y
subscripts	which points to plot
at	cuts for the contours
shrink	what does this do?
labels	draw the contour labels
label.style	where to put the labels
contour	logical draw the contours
region	logical color the regions
col	color for contours
lty	type for contours
lwd	width for contour
border	type of border
...	dots additional arguments
col.regions	a vector of colors or a function (topo.colors by default) for generating such
filled	whether to fill the contours with color
alpha.regions	transparency of regions

---

panel.lmbands   *show confidence and preciction bands on plots*

Description

show confidence and preciction bands on plots

Usage

```
panel.lmbands(x, y, interval = "confidence", level = 0.95, model = lm(y ~
  x), band.col = c(conf = slcol[3], pred = slcol[2]), band.lty = c(conf =
  slty[3], pred = slty[2]), band.show = TRUE, fit.show = TRUE,
  band.alpha = 0.6, band.lwd = 1, npts = 100, ...)
```

Arguments

`x, y`	numeric vectors
`interval`	a vector subset of `'confidence'` and `'prediction'`
`level`	confidence level
`model`	model to be used for generating bands
`band.col`	a vector of length 1 or 2 giving the color of bands
`band.lty`	a vector of length 1 or 2 giving the line type for bands
`band.show`	logical vector of length 1 or 2 indicating whether confidence and prediction bands should be shown
`fit.show`	logical indicating whether the model fit should be shown
`band.alpha`	a vector of length 1 or 2 alpha level for bands
`band.lwd`	a vector of length 1 or 2 giving line width for bands
`npts`	resolution parameter for bands (increase to get better resolution)
`...`	additional arguments

`panel.plotFun`	*Panel function for plotting functions*

Description

Panel function for plotting functions

Usage

```
panel.plotFun(object, ..., type = "l", npts = NULL, zlab = NULL,
  filled = TRUE, levels = NULL, nlevels = 10, surface = FALSE,
  col.regions = topo.colors, alpha = NULL, discontinuity = NULL,
  discontinuities = NULL)
```

Arguments

`object`	an object (e.g., a formula) describing a function
`...`	additional arguments, typically processed by `lattice` panel functions such as `panel.xyplot` or `panel.levelplot`. Frequently used arguments include
	`lwd` line width
	`lty` line type
	`col` a color
`type`	type of plot (`"l"` by default)
`npts`	an integer giving the number of points (in each dimension) to sample the function
`zlab`	label for z axis (when in surface-plot mode)

filled	fill with color between the contours (TRUE by default)
levels	levels at which to draw contours
nlevels	number of contours to draw (if `levels` not specified)
surface	a logical indicating whether to draw a surface plot rather than a contour plot
col.regions	a vector of colors or a function (`topo.colors` by default) for generating such
alpha	number from 0 (transparent) to 1 (opaque) for the fill colors
discontinuity	a positive number determining how sensitive the plot is to potential discontinuity. Larger values result in less sensitivity. The default is 1. Use `discontinuity = Inf` to disable discontinuity detection. Discontinuity detection uses a crude numerical heuristic and may not give the desired results in all cases.
discontinuities	
	a vector of input values at which a function is discontinuous or NULL to use a heuristic to auto-detect.

See Also

plotFun

Examples

```
x <- runif(30,0,2*pi)
d <- data.frame( x = x,  y = sin(x) + rnorm(30,sd=.2) )
xyplot( y ~ x, data=d )
ladd(panel.plotFun( sin(x) ~ x, col='red' ) )
xyplot( y ~ x | rbinom(30,1,.5), data=d )
ladd(panel.plotFun( sin(x) ~ x, col='red', lty=2 ) )    # plots sin(x) in each panel
```

---

panel.plotFun1 *Panel function for plotting functions*

Description

Panel function for plotting functions

Usage

```
panel.plotFun1(..f.., ..., x, y, type = "l",
  col = trellis.par.get("superpose.line")$col, npts = NULL, zlab = NULL,
  filled = TRUE, levels = NULL, nlevels = 10, surface = FALSE,
  alpha = NULL, discontinuity = NULL, discontinuities = NULL)
```

Arguments

`..f..`	an object (e.g., a formula) describing a function
`...`	additional arguments, typically processed by `lattice` panel functions such as `panel.xyplot` or `panel.levelplot`. Frequently used arguments include

> `lwd` line width
> `lty` line type
> `col` a color

`x, y`	ignored, but there for compatibility with other lattice panel functions
`type`	type of plot (″l″ by default)
`col`	a vector of colors
`npts`	an integer giving the number of points (in each dimension) to sample the function
`zlab`	label for z axis (when in surface-plot mode)
`filled`	fill with color between the contours (`TRUE` by default)
`levels`	levels at which to draw contours
`nlevels`	number of contours to draw (if `levels` not specified)
`surface`	a logical indicating whether to draw a surface plot rather than a contour plot
`alpha`	number from 0 (transparent) to 1 (opaque) for the fill colors
`discontinuity`	a positive number determining how sensitive the plot is to potential discontinuity. Larger values result in less sensitivity. The default is 1. Use `discontinuity = Inf` to disable discontinuity detection. Discontinuity detection uses a crude numerical heuristic and may not give the desired results in all cases.
`discontinuities`	
	a vector of input values at which a function is discontinuous or `NULL` to use a heuristic to auto-detect.

See Also

plotFun

Examples

```
x <- runif(30,0,2*pi)
d <- data.frame( x = x,  y = sin(x) + rnorm(30,sd=.2) )
xyplot( y ~ x, data=d )
ladd(panel.plotFun1( sin, col='red' ) )
xyplot( y ~ x | rbinom(30,1,.5), data=d )
ladd(panel.plotFun1( sin, col='red', lty=2 ) )    # plots sin(x) in each panel
```

parse.formula   *Parse formulas*

Description

utilities for exptracting portions of formulas.

Usage

```
parse.formula(formula, ...)

rhs(x, ...)

lhs(x, ...)

condition(x, ...)

operator(x, ...)

## S3 method for class 'formula'
rhs(x, ...)

## S3 method for class 'formula'
lhs(x, ...)

## S3 method for class 'formula'
condition(x, ...)

## S3 method for class 'formula'
operator(x, ...)

## S3 method for class 'parsedFormula'
rhs(x, ...)

## S3 method for class 'parsedFormula'
lhs(x, ...)

## S3 method for class 'parsedFormula'
operator(x, ...)

## S3 method for class 'parsedFormula'
condition(x, ...)
```

Arguments

formula,	a formula
x,	an object (currently a `formula` or `parsedFormula`)

```
...                     additional arguments, current ignored
```

Details

currently this is primarily concerned with extracting the operator, left hand side, right hand side
(minus any condition) and the condition. Improvements/extensions may come in the future.

Value

an object of class `parsedFormula` from which information is easy to extract

---

pdist   *Illustrated probability calculations from distributions*

Description

Illustrated probability calculations from distributions

Usage

```
pdist(dist = "norm", q, plot = TRUE, verbose = FALSE, invisible = FALSE,
  digits = 4L, xlim, ylim, vlwd = NULL,
  vcol = trellis.par.get("add.line")$col, rot = 45, resolution = 5000L,
  ...)

xpgamma(...)

xpt(...)

xpchisq(...)

xpf(...)

xpbinom(...)

xppois(...)

xpgeom(...)

xpnbinom(...)
```

Arguments

dist	a character discription of a distribution, for example "norm", "t", or "chisq"
q	a vector of quantiles
plot	a logical indicating whether a plot should be created
verbose	a logical

invisible	a logical
digits	the number of digits desired
xlim	x limits
ylim	y limits
vlwd	width of vertical lines
vcol	color of vertical lines
rot	angle for rotating text indicating probability
resolution	number of points used for detecting discreteness and generating plots. The defaut value of 5000 should work well except for discrete distributions that have many distinct values, especially if these values are not evenly spaced.
...	additional arguments, including parameters of the distribution and additional options for the plot

Details

The most general function is `pdist` which can work with any distribution for which a p-function exists. As a convenience, wrappers are provided for several common distributions.

Value

a vector of probabilities; a plot is printed as a side effect

See Also

[qdist](), [xpnorm](), [xqnorm]().

Examples

```
pdist("norm", -2:2)
pdist("norm", seq(80,120, by=10), mean=100, sd=10)
pdist("chisq", 2:4, df=3)
pdist("f", 1, df1=2, df2=10)
pdist("gamma", 2, shape=3, rate=4)
```

| perctable | *Cross tabulation displayed as percents or proportions* |

Description

`perctable` and `proptable` use the cross-classifying factors to build a contingency table of the percents or proportions at each combination of factor levels.

Usage

```
perctable(...)

proptable(...)
```

Arguments

`...`	arguments passed directly to [`table`](); typically one or more objects which can be interpreted as factors (including character strings), or a list (or data frame) whose components can be so interpreted.

Details

See [`table`]().

Value

a contingency table, an object of class "table", an array of percentage or proportion values. Note that unlike S the result is always an array, a 1D array if one factor is given.

Examples

```
# perctable() has been deprecated.  Use tally() instead
example(tally)
```

---

plotCumfreq   *Cumulative frequency plots*

Description

A high-level function for producing a cumulative frequency plot using `lattice` graphics.

Usage

```
plotCumfreq(x, data, ...)

## S3 method for class 'formula'
plotCumfreq(x, data = NULL, subscripts, ...)

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

prepanel.cumfreq(x, ...)

panel.cumfreq(x, type = c("smooth", "step"), groups = NULL, ...)
```

Arguments

`x`	a formula or numeric vector
`data`	a data frame in which `x` is evaluated if `x` is a formula.
`subscripts`	as in lattice plots
`type`	smooth or step-function?

groups	grouping variable
...	other lattice arguments

See Also

[histogram](), [densityplot]()

Examples

```
plotCumfreq(~eruptions, faithful, xlab = 'duration of eruptions')
```

---

plotDist	*Plots of Discrete and Continuous Distributions*

Description

Provides a simple way to generate plots of pdfs, probability mass functions, cdfs, probability histograms, and normal-quantile plots for distributions known to R.

Usage

```
plotDist(dist, ..., add, under = FALSE, packets = NULL, rows = NULL,
  columns = NULL, kind = c("density", "cdf", "qq", "histogram"),
  xlab = "", ylab = "", breaks = NULL, type, resolution = 5000,
  params = NULL)
```

Arguments

dist	A string identifying the distribution. This should work with any distribution that has associated functions beginning with 'd', 'p', and 'q' (e.g, [dnorm](), [pnorm](), and [qnorm]()). dist should match the name of the distribution with the initial 'd', 'p', or 'q' removed.
add	a logical indicating whether the plot should be added to the previous lattice plot. If missing, it will be set to match under.
under	a logical indicating whether adding should be done in a layer under or over the existing layers when add = TRUE.
packets, rows, columns	
	specification of which panels will be added to when add is TRUE. See [layer]().
kind	one of "density", "cdf", "qq", or "histogram" (or prefix of any of these)
xlab, ylab	as per other lattice functions
breaks	a vector of break points for bins of histograms, as in [histogram]()
type	passed along to various lattice graphing functions
resolution	number of points to sample when generating the plots
params	a list containing parameters for the distribution. If NULL (the default), this list is created from elements of ... that are either unnamed or have names among the formals of the appropriate distribution function. See the examples.
...	other arguments passed along to lattice graphing routines

Details

`plotDist` determines whether the distribution is continuous or discrete by seeing if all the sampled quantiles are unique. A discrete random variable with many possible values could fool this algorithm and be considered continuous.

The plots are done referencing a data frame with variables x and y giving points on the graph of the pdf, pmf, or cdf for the distribution. This can be useful in conjuction with the `groups` argument. See the examples.

Examples

```
plotDist('norm')
plotDist('norm', type='h')
plotDist('norm', kind='cdf')
plotDist('exp',  kind='histogram')
plotDist('binom', params=list( 25, .25))       # explicit params
plotDist('binom', 25, .25)                      # params inferred
plotDist('norm', mean=100, sd=10, kind='cdf')  # params inferred
plotDist('binom', 25, .25, xlim=c(-1,26) )     # params inferred
plotDist('binom', params=list( 25, .25), kind='cdf')
plotDist('beta', params=list( 3, 10), kind='density')
plotDist('beta', params=list( 3, 10), kind='cdf')
plotDist( "binom", params=list(35,.25),
          groups= y < dbinom(qbinom(0.05, 35, .25), 35,.25) )
plotDist( "binom", params=list(35,.25),
          groups= y < dbinom(qbinom(0.05, 35, .25), 35,.25),
          kind='hist')
plotDist("norm", mean=10, sd=2, col="blue", type="h")
plotDist("norm", mean=12, sd=2, col="red", type="h", under=TRUE)
plotDist("binom", size=100, prob=.30) +
  plotDist("norm", mean=30, sd=sqrt(100 * .3 * .7))
plotDist("chisq", df=4, groups = x > 6, type="h")
if (require(mosaicData)) {
histogram( ~age|sex, data=HELPrct)
m <- mean( ~age|sex, data=HELPrct)
s <- sd(~age|sex, data=HELPrct)
plotDist( "norm", mean=m[1], sd=s[1], col="red", add=TRUE, packets=1)
plotDist( "norm", mean=m[2], sd=s[2], col="blue", under=TRUE, packets=2)
}
```

---

plotFun   *Plotting mathematical expressions*

Description

Plots mathematical expressions in one and two variables.

Usage

```
plotFun(object, ..., plot = trellis.last.object(), add = NULL,
  under = FALSE, xlim = NULL, ylim = NULL, npts = NULL, ylab = NULL,
  xlab = NULL, zlab = NULL, filled = TRUE, levels = NULL,
  nlevels = 10, labels = TRUE, surface = FALSE, groups = NULL,
  col = trellis.par.get("superpose.line")$col, col.regions = topo.colors,
  type = "l", alpha = NULL, discontinuities = NULL, discontinuity = 1,
  interactive = rstudio_is_available())
```

Arguments

object	a mathematical expression or a function "of one variable" which will converted to something intuitively equivalent to object(x) ~ x. (See examples)
...	additional parameters, typically processed by lattice functions such as [xyplot](#), [levelplot](#) or their panel functions. Frequently used parameters include
	main main title for plot
	sub subtitle for plot
	lwd line width
	lty line type
	col a color or a (small) integer indicating which color in the current color scheme is desired.
	Additionally, these arguments can be used to specify parameters for the function being plotted and to specify the plotting window with natural names. See the examples for such usage.
plot	a trellis object; by default, the most recently created trellis plot. When add is TRUE, the new function will be plotted into a layer added to this object.
add	if TRUE, then add a layer to an existing plot rather than creating a new plot. If NULL, this will be determined by the value of under.
under	if TRUE, then new layer is added beneath existing layers
xlim	limits for x axis (or use variable names, see examples)
ylim	limits for y axis (or use variable names, see examples)
npts	number of points for plotting.
ylab	label for y axis
xlab	label for x axis
zlab	label for z axis (when in surface-plot mode)
filled	fill with color between the contours (TRUE by default)
levels	levels at which to draw contours
nlevels	number of contours to draw (if levels not specified)
labels	if FALSE, don't label contours
surface	draw a surface plot rather than a contour plot
groups	grouping argument ala lattice graphics
col	vector of colors for line graphs and contours

col.regions	a vector of colors or a function (topo.colors by default) for generating such
type	type of plot ("l" by default)
alpha	number from 0 (transparent) to 1 (opaque) for the fill colors
discontinuities	
	a vector of input values at which a function is discontinuous or NULL to use a heuristic to auto-detect.
discontinuity	a positive number determining how sensitive the plot is to potential discontinuity. Larger values result in less sensitivity. The default is 1. Use discontinuity = Inf to disable discontinuity detection. Discontinuity detection uses a crude numerical heuristic and may not give the desired results in all cases.
interactive	a logical indicating whether the surface plot should be interactive.

Details

makes plots of mathematical expressions using the formula syntax. Will draw both line plots and contour/surface plots (for functions of two variables). In RStudio, the surface plot comes with sliders to set orientation. If the colors in filled surface plots are too blocky, increase npts beyond the default of 50, though npts=300 is as much as you're likely to ever need. See examples for overplotting a constraint function on an objective function.

Value

a trellis object

Examples

```
plotFun( a*sin(x^2)~x, xlim=range(-5,5), a=2 )  # setting parameter value
plotFun( u^2 ~ u, ulim=c(-4,4) )                # limits in terms of u
# Note roles of ylim and y.lim in this example
plotFun( y^2 ~ y, ylim=c(-2,20), y.lim=c(-4,4) )
# Combining plot elements to show the solution to an inequality
plotFun( x^2 -3 ~ x, xlim=c(-4,4), grid=TRUE )
ladd( panel.abline(h=0,v=0,col='gray50') )
plotFun( (x^2 -3) * (x^2 > 3) ~ x, type='h', alpha=.1, lwd=4, col='lightblue', add=TRUE )
plotFun( sin(x) ~ x,
   groups=cut(x, findZeros(sin(x) ~ x, within=10)$x),
   col=c('blue','green'), lty=2, lwd=3, xlim=c(-10,10) )
plotFun( sin(x) ~ x,
   groups=cut(x, findZeros(sin(x) ~ x, within=10)$x),
   col=c(1,2), lty=2, lwd=3, xlim=c(-10,10) )
## plotFun( sin(2*pi*x/P)*exp(-k*t)~x+t, k=2, P=.3)
f <- rfun( ~ u & v )
plotFun( f(u=u,v=v) ~ u & v, u.lim=range(-3,3), v.lim=range(-3,3) )
plotFun( u^2 + v < 3 ~ u & v, add=TRUE, npts=200 )
if (require(mosaicData)) {
# display a linear model using a formula interface
model <- lm(wage ~ poly(exper,degree=2), data=CPS85)
fit <- makeFun(model)
xyplot(wage ~ exper, data=CPS85)
plotFun(fit(exper) ~ exper, add=TRUE, lwd=3, col="red")
```

```
# Can also just give fit since it is a "function of one variable"
plotFun(fit, add=TRUE, lwd=2, col='white')
}
# Attempts to find sensible axis limits by default
plotFun( sin(k*x)~x, k=0.01 )
# Plotting a linear model with multiple predictors.
mod <- lm(length ~ width * sex, data=KidsFeet)
fitted.length <- makeFun(mod)
xyplot(length ~ width, groups=sex, data=KidsFeet, auto.key=TRUE)
plotFun(fitted.length(width, sex="B") ~ width, add=TRUE, col=1)
plotFun(fitted.length(width, sex="G") ~ width, add=TRUE, col=2)
```

---

plotModel   *Plot a regression model*

Description

Visualize a regression model amid the data that generated it.

Usage

```
plotModel(mod, ...)
```

Arguments

mod	A model of type lm or glm
...	arguments passed to xyplot or rgl::plot3d.

Details

The goal of this function is to assist with visualization of statistical models. Namely, to plot the model on top of the data from which the model was fit.

The primary plot type is a scatter plot. The x-axis can be assigned to one of the predictors in the model. Additional predictors are thought of as co-variates. The data and fitted curves are partitioned by these covariates. When the number of components to this partition is large, a random subset of the fitted curves is displayed to avoid visual clutter.

If the model was fit on one quantitative variable (e.g. SLR), then a scatter plot is drawn, and the model is realized as parallel or non-parallel lines, depending on whether interaction terms are present.

Eventually we hope to support 3-d visualizations of models with 2 quantitative predictors using the rgl package.

Currently, only linear regression models and generalized linear regression models are supported.

Value

A lattice or ggplot2 graphics object.

Caution

This is still underdevelopment. The API is subject to change, and some use cases may not work yet. Watch for improvments in subsequent versions of the package.

Author(s)

Ben Baumer, Galen Long, Randall Pruim

See Also

[plotPoints](), [plotFun]()

Examples

```
require(mosaic)

mod <- lm( mpg ~ factor(cyl), data = mtcars)
plotModel(mod)

# SLR
mod <- lm( mpg ~ wt, data = mtcars)
plotModel(mod, pch = 19)

# parallel slopes
mod <- lm( mpg ~ wt + factor(cyl), data=mtcars)
plotModel(mod)

# multiple categorical vars
mod <- lm( mpg ~ wt + factor(cyl) + factor(vs) + factor(am), data = mtcars)
plotModel(mod)
plotModel(mod, mpg ~ am)

# interaction
mod <- lm( mpg ~ wt + factor(cyl) + wt:factor(cyl), data = mtcars)
plotModel(mod)

# polynomial terms
mod <- lm( mpg ~ wt + I(wt^2), data = mtcars)
plotModel(mod)

# GLM
mod <- glm(vs ~ wt, data=mtcars, family = 'binomial')
plotModel(mod)

# GLM with interaction
mod <- glm(vs ~ wt + factor(cyl), data=mtcars, family = 'binomial')
plotModel(mod)
# 3D model
mod <- lm( mpg ~ wt + hp, data = mtcars)
plotModel(mod)
```

```
# parallel planes
mod <- lm( mpg ~ wt + hp + factor(cyl) + factor(vs), data = mtcars)
plotModel(mod)

# interaction planes
mod <- lm( mpg ~ wt + hp + wt * factor(cyl), data = mtcars)
plotModel(mod)
plotModel(mod, system="g") + facet_wrap( ~ cyl )
```

---

plotPoints   *Scatter plot of points*

Description

Make or add a scatter plot in a manner coordinated with plotFun.

Usage

```
plotPoints(x, data = parent.frame(), add = NULL, under = FALSE,
  panelfun = panel.xyplot, plotfun = xyplot, ...,
  plot = trellis.last.object())
```

Arguments

x	A formula specifying y ~ x or z ~ x&y
data	Data frame containing the variables to be plotted. If not specified, the variables will be looked up in the local environment
add	If TRUE, add points as a new layer to an existing plot. If NULL, the value of under will be used.
under	If TRUE, the new layer will be underneat existing layers.
panelfun	Lattice panel function to be used for adding. Set only if you want something other than a scatter plot. Mainly, this is intended to add new functionality through other functions.
plotfun	Lattice function to be used for initial plot creation. Set only if you want something other than a scatter plot. Mainly, this is intended to add new functionality through other functions.
plot	a trellis plot, by default the most recently created one. If add is TRUE, new points will be added as a new layer to plot.
...	additional arguments

Value

A trellis graphics object

See Also

plotFun

Examples

```
if (require(mosaicData)) {
plotPoints( width ~ length, data=KidsFeet, groups=sex, pch=20)
f <- makeFun( lm( width ~ length * sex, data=KidsFeet))
plotFun( f(length=length,sex="G")~length, add=TRUE, col="pink")
plotFun( f(length=length,sex="B")~length, add=TRUE)
}
```

---

predict.groupwiseModel

Evaluate a groupwise model given new data

Description

If newdata is not specified, the data originally used for fitting will be used.

Usage

```
## S3 method for class 'groupwiseModel'
predict(object, newdata = object$data,
  type = c("class", "likelihood", "prob"), level = NULL, ...)
```

Arguments

object	a groupwise model
newdata	new data from which to compute fitted valeus.
type	one of "class", "likelihood", or "prob"
level	an optional character string specifying the level for which probabilities are to be reported. Defaults to the first class of the potential outputs. Set to ".all" to see probabilities for all levels.
...	additional arguments (currently ignored)

Details

setting the type is needed only for classifiers. "class" will give just the class as output. "likelihood" will give the probability of the observed outcome (in newdata) given the model. "prob" will give the probability of the class named in level

print.msummary.lm   *Modified summaries*

Description

msummary provides modified summary objects that typically produce output that is either identical to or somewhat terser than their summary analogs. The contents of the object itself are unchanged (except for an augmented class) so that other downstream functions should work as before.

Usage

```
## S3 method for class 'msummary.lm'
print(x, digits = max(3L, getOption("digits") - 3L),
  symbolic.cor = x$symbolic.cor,
  signif.stars = getOption("show.signif.stars"), ...)

## S3 method for class 'msummary.glm'
print(x, digits = max(3L, getOption("digits") - 3L),
  symbolic.cor = x$symbolic.cor,
  signif.stars = getOption("show.signif.stars"), ...)

msummary(object, ...)

## Default S3 method:
msummary(object, ...)

## S3 method for class 'lm'
msummary(object, ...)

## S3 method for class 'glm'
msummary(object, ...)
```

Arguments

x	an object to summarize
digits	desired number of digits to display
symbolic.cor	see summary
signif.stars	a logical indicating whether to diplay stars to indicate significance
...	aditional arguments
object	an object to summarise

Examples

```
msummary(lm(Sepal.Length ~ Species, data = iris))
```

---

project   *Projections*

Description

Compute projections onto the span of a vector or a model space, dot products, and vector lengths in Euclidean space.

Usage

```
project(x, ...)

## S4 method for signature 'formula'
project(x, u = NULL, data = parent.frame(2),
  coefficients = TRUE, ...)

## S4 method for signature 'numeric'
project(x, u = rep(1, length(x)), type = c("vector",
  "length", "coef"), ...)

## S4 method for signature 'matrix'
project(x, u, data = parent.frame())

vlength(x, ...)

dot(u, v)
```

Arguments

x	a numeric vector (all functions) or a formula (only for `project`). Left-hand sides of formulas should be a single quantity
...	additional arguments
u	a numeric vector
data	a data frame.
coefficients	For `project(y ~ x)` indicates whether the projection coeffients should be returned or the projection vector.
type	one of `"length"` or `"vector"` determining the type of the returned value
v	a numeric vector

Details

`project` (preferably pronounced "pro-JECT" as in "projection") does either of two related things: (1) Given two vectors as arguments, it will project the first onto the second, returning the point in the subspace of the second that is as close as possible to the first vector. (2) Given a formula as an argument, will work very much like `lm()`, constructing a model matrix from the right-hand side of the formula and projecting the vector on the left-hand side onto the subspace of that model matrix.

In (2), rather than returning the projected vector, `project()` returns the coefficients on each of the vectors in the model matrix. UNLIKE `lm()`, the intercept vector is NOT included by default. If you want an intercept vector, include +1 in your formula.

Value

`project` returns the projection of x onto u (or its length if u and v are numeric vectors and type == "length")

`vlength` returns the length of the vector (i.e., the square root of the sum of the squares of the components)

`dot` returns the dot product of u and v

See Also

```
link{project}
```

Examples

```
x1 <- c(1,0,0); x2 <- c(1,2,3); y1 <- c(3,4,5); y2 <- rnorm(3)
# projection onto the 1 vector gives the mean vector
mean(y2)
project(y2, 1)
# return the length of the vector, rather than the vector itself
project(y2, 1, type='length')
project(y1 ~ x1 + x2) -> pr; pr
# recover the projected vector
cbind(x1,x2) %*% pr -> v; v
project( y1 ~ x1 + x2, coefficients=FALSE )
dot( y1 - v, v ) # left over should be orthogonal to projection, so this should be ~ 0
if (require(mosaicData)) {
project(width~length+sex, data=KidsFeet)
}
vlength(rep(1,4))
if (require(mosaicData)) {
m <- lm( length ~ width, data=KidsFeet )
# These should be the same
vlength( m$effects )
vlength( KidsFeet$length)
# So should these
vlength( tail(m$effects, -2) )
sqrt(sum(resid(m)^2))
}
v <- c(1,1,1); w <- c(1,2,3)
u <- v / vlength(v)  # make a unit vector
# The following should be the same:
project(w,v, type="coef") * v
project(w,v)
# The following are equivalent
abs(dot( w, u ))
vlength( project( w, u) )
vlength( project( w, v) )
```

```
project( w, v, type='length' )
```

---

prop   *Compute proportions, percents, or counts for a single level*

Description

Compute proportions, percents, or counts for a single level

Usage

```
prop(x, data = parent.frame(), useNA = "no", ..., level = NULL,
  long.names = TRUE, sep = ".", format = c("proportion", "percent",
  "count"), quiet = TRUE, pval.adjust = FALSE)

prop1(..., pval.adjust = TRUE)

count(x, data = parent.frame(), ..., format = "count")

perc(x, data = parent.frame(), ..., format = "percent")
```

Arguments

x	an R object, usually a formula
data	a data frame in which x is to be evaluated
useNA	an indication of how NA's should be handled. By default, they are ignored.
level	the level for which counts, proportions or percents are calculated
long.names	a logical indicating whether long names should be when there is a conditioning variable
sep	a character used to separate portions of long names
format	one of proportion, percent, or count, possibly abbrevaited
quiet	a logical indicating whether messages regarding the target level should be suppressed.
pval.adjust	a logical indicating whether the "p-value" adjustment should be applied. This adjustment adds 1 to the numerator and denominator counts.
...	arguments passed through to tally

Details

prop1 is intended for the computation of p-values from randomization distributions and differs from prop only in its default value of pval.adjust.

Note

For 0-1 data, level is set to 1 by default since that a standard coding scheme for success and failure.

Examples

```
if (require(mosaicData)) {
prop( ~sex, data=HELPrct)
prop( ~sex, data=HELPrct, level='male')
count( ~sex | substance, data=HELPrct)
prop( ~sex | substance, data=HELPrct)
perc( ~sex | substance, data=HELPrct)
}
```

| prop.test | *Exact and Approximate Tests for Proportions* |

Description

The mosaic `prop.test` provides wrapper functions around the function of the same name in **stats**. These wrappers provide an extended interface (including formulas). `prop.test` performs an approximate test of a simple null hypothesis about the probability of success in a Bernoulli or multinomial experiment from summarized data or from raw data.

Usage

```
prop.test(x, n, p = NULL, alternative = c("two.sided", "less", "greater"),
  conf.level = 0.95, data = NULL, success = NULL, ...)
```

Arguments

x	count of successes, length 2 vector of success and failure counts, a formula, or a character, numeric, or factor vector containing raw data.
n	sample size (successes + failures) or a data frame (for the formula interface)
p	a vector of probabilities of success. The length of p must be the same as the number of groups specified by x, and its elements must be greater than 0 and less than 1.
alternative	character string specifying the alternative hypothesis, must be one of `"two.sided"` (default), `"greater"` or `"less"`. You can specify just the initial letter. Only used for testing the null that a single proportion equals a given value, or that two proportions are equal; ignored otherwise.
conf.level	confidence level of the returned confidence interval. Must be a single number between 0 and 1. Only used when testing the null that a single proportion equals a given value, or that two proportions are equal; ignored otherwise.
data	a data frame (if missing, n may be a data frame)
success	level of variable to be considered success. All other levels are considered failure.
...	additional arguments (often ignored). When x is a formula, groups can be used to compare groups: x = ~ var, groups=g is equivalent to x = var ~ g . See the examples.

Details

conf.level = 0.95, ...)

This is a wrapper around `prop.test` to simplify its use when the raw data are available, in which case an extended syntax for `prop.test` is provided.

Value

an `htest` object

Note

When `x` is a 0-1 vector, 0 is treated as failure and 1 as success. Similarly, for a logical vector `TRUE` is treated as success and `FALSE` as failure.

See Also

`binom.test`, `prop.test`

Examples

```
# Several ways to get a confidence interval for the proportion of Old Faithful
# eruptions lasting more than 3 minutes.
prop.test( faithful$eruptions > 3 )
prop.test(97,272)
faithful$long <- faithful$eruptions > 3
prop.test( faithful$long )
prop.test( ~long , data = faithful )
prop.test( homeless ~ sex, data = HELPrct )
prop.test( ~ homeless | sex, data = HELPrct )
prop.test( ~ homeless, groups = sex, data = HELPrct )
```

---

qdata   *The Data Distribution*

Description

Density, distribution function, quantile function, and random generation from data.

Usage

```
qdata(formula, p = seq(0, 1, 0.25), data = NULL, ...)

cdata(formula, p = 0.95, data = NULL, ...)

pdata(formula, q, data = NULL, ...)

rdata(formula, n, data = NULL, ...)
```

```
ddata(formula, q, data = NULL, ...)
```

Arguments

`formula`	a formula or a vector
`p`	a vector of probabilities
`data`	a data frame in which to evaluate `formula`
`q`	a vector of quantiles
`n`	number of values to sample
`...`	additional arguments passed to `quantile` or `sample`

Value

For `qdata`, a vector of quantiles

for `cdata`, a named numerical vector or a data frame giving upper and lower limits and the central proportion requested

For `pdata`, a vector of probabilities

For `rdata`, a vector of sampled values.

For `ddata`, a vector of probabilities (empirical densities)

Examples

```
data(iris)
qdata(Sepal.Length ~ Species, 0.5, data=iris)
qdata(~Sepal.Length, p = 0.5, groups=Species, data=iris)
qdata(iris$Sepal.Length, p = 0.5)
qdata(~ Sepal.Length, p = 0.5, data=iris)
qdata(~ Sepal.Length, p = 0.5, groups=Species, data=iris)
data(iris)
cdata(iris$Sepal.Length, 0.5)
cdata( ~ Sepal.Length, 0.5, data = iris)
cdata( ~ Sepal.Length, 0.5, data = iris)
cdata( ~ Sepal.Length | Species, data = iris, p = .5)
data(iris)
pdata(iris$Sepal.Length, 3:6)
pdata( ~ Sepal.Length, 3:6, data=iris)
data(iris)
rdata(iris$Species, 10)
rdata(~Species, n = 10, data=iris)
rdata(Sepal.Length ~ Species,  n = 5, data=iris)
data(iris)
ddata(iris$Species, 'setosa')
ddata(~Species, 'setosa', data=iris)
```

---

qdata_v   *The Data Distribution*

Description

Utility functions for density, distribution function, quantile function, and random generation from data.

Usage

```
qdata_v(x, p = seq(0, 1, 0.25), na.rm = TRUE, ...)

qdata_f(x, ..., data = NULL, groups = NULL, na.rm = TRUE)

cdata_v(x, p = 0.95, na.rm = TRUE, ...)

cdata_f(x, ..., data = NULL, groups = NULL, na.rm = TRUE)

pdata_v(x, q, lower.tail = TRUE, ...)

pdata_f(x, ..., data = NULL, groups = NULL, na.rm = TRUE)

rdata_v(x, n, replace = TRUE, ...)

rdata_f(x, ..., data = NULL, groups = NULL, na.rm = TRUE)

ddata_v(x, q, ..., data = NULL, log = FALSE, na.rm = TRUE)

ddata_f(x, ..., data = NULL, groups = NULL, na.rm = TRUE)
```

Arguments

x	a vector containing the data
p	a vector of probabilities
na.rm	a logical indicating whether NAs should be removed before computing.
data	a data frame in which to evaluate formula
groups	a grouping variable, typically the name of a variable in data
q	a vector of quantiles
lower.tail	a logical indicating whether to use the lower or upper tail probability
n	number of values to sample
replace	a logical indicating whether to sample with replacement
log	a logical indicating whether the result should be log transformed
...	additional arguments passed to quantile or sample

See Also

[ddata](), [pdata](), [qdata](), [rdata](), [cdata]()

---

qdist   *Illustrated quantile calculations from distributions*

Description

Illustrated quantile calculations from distributions

Usage

```
qdist(dist = "norm", p, plot = TRUE, verbose = FALSE, invisible = FALSE,
  resolution = 5000L, digits = 4L, xlim, ylim, vlwd = NULL,
  vcol = trellis.par.get("add.line")$col, rot = 45, ...)

xqgamma(...)

xqt(...)

xqchisq(...)

xqf(...)

xqbinom(...)

xqpois(...)

xqgeom(...)

xqnbinom(...)
```

Arguments

dist	a character discription of a distribution, for example "norm", "t", or "chisq"
p	a vector of probabilities
plot	a logical indicating whether a plot should be created
verbose	a logical
invisible	a logical
resolution	number of points used for detecting discreteness and generating plots. The defaut value of 5000 should work well except for discrete distributions that have many distinct values, especially if these values are not evenly spaced.
digits	the number of digits desired
xlim	x limits. By default, these are chosen to show the central 99.8% of the distribution.

ylim	y limits
vlwd	width of vertical lines
vcol	color of vertical lines
rot	angle for rotating text indicating probability
...	additional arguments, including parameters of the distribution and additional options for the plot

Details

The most general function is `qdist` which can work with any distribution for which a q-function exists. As a convenience, wrappers are provided for several common distributions.

Value

a vector of quantiles; a plot is printed as a side effect

Examples

```
qdist("norm", seq(.2, .8, by = 0.10))
xqnorm(seq(.2, .8, by = 0.10), mean = 100, sd = 10)
qdist("unif", .5)
xqgamma(.5, shape = 3, scale = 4)
xqchisq(c(.25,.5,.75), df = 3)
xpbinom( c(480, 500, 510), size = 1000, prob = 0.48)
xpbinom( c(40, 60), size = 100, prob = 0.5)
xqpois( c(0.25, 0.5, 0.75), lambda = 6, lwd = 3, vlwd = 2)
```

---

r.squared   *Extract r-squared value*

Description

Attempts to extract an r-squared value from a model or model-like object.

Usage

```
r.squared(x, ...)
```

Arguments

x	an object
...	additional arguments

---

rand *Random Regressors*

Description

A utility function for producing random regressors with a specified number of degrees of freedom.

Usage

```
rand(df = 1, rdist = rnorm, args = list(), nrow, seed = NULL)
```

Arguments

df	degrees of freedom, i.e., number of random regressors
rdist	random distribution function for sampling
args	arguments for rdist
nrow	number of rows in resulting matrix. This can often be omitted in the context of functions like lm where it is inferred from the data frame, if one is provided.
seed	seed for random number generation

Value

A matrix of random variates with df columns. In its intended use, the number of rows will be selected to match the size of the data frame supplied to lm

Examples

```
rand(2,nrow=4)
rand(2,rdist=rpois, args=list(lambda=3), nrow=4)
summary(lm( waiting ~ eruptions + rand(1), faithful))
```

---

read.file *Read data files*

Description

A wrapper around various file reading functions.

Usage

```
read.file(file, header = T, na.strings = "NA", comment.char = NULL,
  filetype = c("default", "csv", "txt", "tsv", "fw", "rdata"),
  stringsAsFactors = FALSE, readr = FALSE, package = NULL, ...)
```

Arguments

`file`	character: The name of the file which the data are to be read from. This may also be a complete URL or a path to a compressed file. If it does not contain an absolute path, the file name is relative to the current working directory, `getwd()`. Tilde-expansion is performed where supported. See `read.table` for more details.
`header`	logical; For `.txt` and `.csv` files, this indicates whether the first line of the file includes variables names.
`na.strings`	character: strings that indicate missing data.
`comment.char`	character: a character vector of length one containing a single character or an empty string. Use "" to turn off the interpretation of comments altogether.
`filetype`	one of `"default"`, `"csv"`, `"txt"`, or `"rdata"` indicating the type of file being loaded. The default is to use the filename to guess the type of file.
`stringsAsFactors`	
	a logical indicating whether strings should be converted to factors. This has no affect when using `readr`.
`readr`	a logical indicating whether functions from the `readr` package should be used, if available.
`package`	if specified, files will be searched for among the documentation files provided by the package.
`...`	additional arguments passed on to `read.table`, or `load` or one of the functions in the `readr` package. Note that a message will indicate which underlying function is being used.

Details

Unless `filetype` is specified, `read.file` uses the (case insensitive) file extension to determine how to read data from the file. If `file` ends in `.rda` or `.rdata`, then `load` is used to load the file. If `file` ends in `.csv`, then `read_csv` or `read.csv` is used. Otherwise, `read.table` is used.

Value

A data frame, unless `file` unless `filetype` is `"rdata"`, in which case arbitrary objects may be loaded and a character vector holding the names of the loaded objects is returned invisibly.

See Also

`read.csv`, `read.table`, `read_table`, `read_csv`, `load`.

Examples

```
## Not run:
Dome <- read.file("http://www.mosaic-web.org/go/datasets/Dome.csv")

## End(Not run)
Mustangs <- read.file("MustangPrice.csv", package="mosaic")
```

---

relm   *Resample a Linear Model*

Description

Residual resampling from a linear model

Usage

```
relm(model, ..., envir = environment(formula(model)))
```

Arguments

model   a linear model object produced using [lm](#).

...   additional arguments passed through to [resample](#).

envir   an environment in which to (re)evaluate the linear model.

---

repeater-class   *Repeater objects*

Description

Repeater objects can be used with the * operator to repeat things multiple time using a different syntax and different output format from that used by, for example, [replicate](#).

Slots

n: Object of class "numeric" indicating how many times to repeat something.

cull: Object of class "function" that culls the ouput from each repetition.

mode: Object of class "character" indicating the output mode ('default', 'data.frame', 'matrix', 'vector', or 'list'). For most purposes 'default' (the default) should suffice.

algorithm: an algorithm number.

parallel: a logical indicating whether to attempt parallel execution.

See Also

[do](#)

---

resample   *More Random Samples*

Description

These functions simplify and unify sampling in various ways.

Usage

```
resample(..., replace = TRUE)

deal(...)

shuffle(x, replace = FALSE, prob = NULL, groups = NULL,
  orig.ids = FALSE)

sample(x, size, replace = FALSE, ...)

## Default S3 method:
sample(x, size, replace = FALSE, prob = NULL,
  groups = NULL, orig.ids = FALSE, ...)

## S3 method for class 'data.frame'
sample(x, size, replace = FALSE, prob = NULL,
  groups = NULL, orig.ids = TRUE, fixed = names(x), shuffled = c(),
  invisibly.return = NULL, ...)

## S3 method for class 'matrix'
sample(x, size, replace = FALSE, prob = NULL,
  groups = NULL, orig.ids = FALSE, ...)

## S3 method for class 'factor'
sample(x, size, replace = FALSE, prob = NULL,
  groups = NULL, orig.ids = FALSE, drop.unused.levels = FALSE, ...)
```

Arguments

replace	Should sampling be with replacement?
x	Either a vector of one or more elements from which to choose, or a positive integer.
prob	A vector of probability weights for obtaining the elements of the vector being sampled.
groups	a vector (or variable in a data frame) specifying groups to sample within. This will be recycled if necessary.
orig.ids	a logical; should origianal ids be included in returned data frame?

size	a non-negative integer giving the number of items to choose.
fixed	a vector of column names. These variables are shuffled en masse, preserving associations among these columns.
shuffled	a vector of column names. these variables are reshuffled individually (within groups if groups is specified), breaking associations among these columns. examples.
invisibly.return	
	a logical, should return be invisible?
drop.unused.levels	
	a logical, should unused levels be dropped?
...	additional arguments passed to sample or sample.

Details

These functions are wrappers around sample providing different defaults and natural names.

Examples

```
# 100 Bernoulli trials -- no need for replace=TRUE
resample(0:1, 100)
tally(resample(0:1, 100))
if (require(mosaicData)) {
Small <- sample(KidsFeet, 10)
resample(Small)
tally(~ sex, data=resample(Small))
tally(~ sex, data=resample(Small))
# fixed marginals for sex
tally(~ sex, data=Small)
tally(~ sex, data=resample(Small, groups=sex))
# shuffled can be used to reshuffle some variables within groups
# orig.id shows where the values were in original data frame.
Small <- mutate(Small,
   id1 = paste(sex,1:10, sep=":"),
   id2 = paste(sex,1:10, sep=":"))
resample(Small, groups=sex, shuffled=c("id1","id2"))
}
deal(Cards, 13)     # A Bridge hand
shuffle(Cards)
```

---

rescale	*Rescale*

Description

Rescale vectors or variables within data frames. This can be useful for comparing vectors that are on different scales, for example in parallel plots or heatmaps.

Usage

```
rescale(x, range, domain = NULL, ...)

## S3 method for class 'data.frame'
rescale(x, range = c(0, 1), domain = NULL, ...)

## S3 method for class 'factor'
rescale(x, range, domain = NULL, ...)

## S3 method for class 'numeric'
rescale(x, range = c(0, 1), domain = NULL, ...)

## Default S3 method:
rescale(x, range = c(0, 1), domain = NULL, ...)

## S3 method for class 'character'
rescale(x, range = c(0, 1), domain = NULL, ...)
```

Arguments

x	an R object to rescale
range	a numeric vector of length 2
domain	a numeric vector of length 2 or `NULL`
...	additional arguments

---

rflip   *Tossing Coins*

Description

These functions simplify simulating coin tosses for those (students primarily) who are not yet familiar with the binomial distributions or just like this syntax and verbosity better.

Usage

```
rflip(n = 1, prob = 0.5, quiet = FALSE, verbose = !quiet)

## S3 method for class 'cointoss'
print(x, ...)

nflip(n = 1, prob = 0.5, ...)
```

Arguments

n	the number of coins to toss
prob	probability of heads on each toss
quiet	a logical. If TRUE, less verbose output is used.
verbose	a logical. If TRUE, more verbose output is used.
x	an object
...	additional arguments

Value

for `rflip`, a cointoss object

for `nflip`, a numeric vector

Examples

```
rflip(10)
rflip(10, prob=1/6, quiet=TRUE)
do(5) * rflip(10)
as.numeric(rflip(10))
nflip(10)
```

---

rfun	*Generate a natural-looking function*

Description

Produce a random function that is the sum of Gaussian random variables

`rpoly2` generates a random 2nd degree polynomial (as a function)

Usage

```
rfun(vars = ~x & y, seed = NULL, n = 0)

rpoly2(vars = ~x & y, seed = NULL)
```

Arguments

vars	a formula; the LHS is empty and the RHS indicates the variables used for input to the function (separated by &)
seed	seed for random number generator, passed to set.seed.
n	the number of Gaussians. By default, this will be selected randomly.

Details

rfun is an easy way to generate a natural-looking but random function with ups and downs much as you might draw on paper. In two variables, it provides a good way to produce a random landscape that is smooth. Things happen in the domain -5 to 5. The function is pretty flat outside of that. Use seed to create a fixed function that will be the same for everybody

These functions are particularly useful for teaching calculus.

Value

a function with the appropriate number of inputs

a function defined by a 2nd degree polynomial with coefficients selected randomly according to a Unif(-1,1) distribution.

Examples

```
f <- rfun( ~ u & v)
plotFun(f(u,v)~u&v,u=range(-5,5),v=range(-5,5))
myfun <- rfun(~ u & v, seed=1959)
g <- rpoly2( ~ x&y&z, seed=1964)
plotFun(g(x,y,z=2)~x&y,xlim=range(-5,5),ylim=range(-5,5))
```

---

rkintegrate *A simple Runge-Kutte integrator*

Description

Integrates ordinary differential equations using a Runge-Kutta method

Usage

```
rkintegrate(fun, x0, tstart = 0, tend = 1, dt = NULL)
```

Arguments

fun	the dynamical function with arguments state (a vector) and t.
x0	the initial condition, a vector with one element for each state variable
tstart	starting time
tend	ending time for integration
dt	step size for integration

Details

This is mainly for internal use by integrateODE.

Value

a list containing x, a matrix of the state with one row for each time step and a vector t containing the times of those steps.

Author(s)

Daniel Kaplan (<kaplan@macalester.edu>)

| rlatlon | *Sample longitude and latitude on a sphere* |

Description

Randomly samples longitude and latitude on earth so that equal areas are (approximately) equally likely to be sampled. (Approximation assumes earth as a perfect sphere.)

Usage

```
rlatlon(...)

rlonlat(...)

rgeo(n = 1, latlim = c(-90, 90), lonlim = c(-180, 180), verbose = FALSE)

rgeo2(n = 1, latlim = c(-90, 90), lonlim = c(-180, 180),
  verbose = FALSE)
```

Arguments

...	arguments passed through to other functions
n	number of random locations
latlim, lonlim	range of latitudes and longitudes to sample within, only implemented for rgeo.
verbose	return verbose output that includes Euclidean coordinates on unit sphere as well as longitude and latitude.

Details

rgeo and rgeo2 differ in the algorithms used to generate random positions. Each assumes a spherical globe. rgeo uses that fact that each of the x, y and z coordinates is uniformly distributed (but not independent of each other). Furthermore, the angle about the z-axis is uniformly distributed and independent of z. This provides a straightforward way to generate Euclidean coordinates using runif. These are then translated into latitude and longitude.

rlatlon is an alias for rgeo and rlonlat is too, expect that it reverses the order in which the lattitude and longitute values are returned.

rgeo2 samples points in a cube by independently sampling each coordinate. It then discards any point outside the sphere contained in the cube and projects the non-discarded points to the sphere. This method must oversample to allow for the discarded points.

Value

a data frame with variables `long` and `lat`. If verbose is TRUE, then x, y, and z coordinates are also included in the data frame.

See Also

[deg2rad](), [googleMap]() and [latlon2xyz]().

Examples

```
rgeo(4)
# sample from a region that contains the continental US
rgeo( 4, latlim=c(25,50), lonlim=c(-65,-125) )
rgeo2(4)
```

---

rspin   *Simulate spinning a spinnner*

Description

This is essentially `rmultinom` with a different interface.

Usage

```
rspin(n, probs, labels = 1:length(probs))
```

Arguments

n	number of spins of spinner
probs	a vector of probabilities. If the sum is not 1, the probabiliies will be rescaled.
labels	a character vector of labels for the categories

Examples

```
rspin(20, prob=c(1,2,3), labels=c("Red", "Blue", "Green"))
do(2) * rspin(20, prob=c(1,2,3), labels=c("Red", "Blue", "Green"))
```

---

rsquared *Extract r-squared value*

Description

Attempts to extract an r-squared value from a model or model-like object.

Usage

```
rsquared(x, ...)
```

Arguments

x   an object

...   additional arguments

---

rstudio_is_available *Check whether RStudio is in use*

Description

This functions checks that RStudio is in use. It will likely be removed from this package once the versions of RStudio in popular use rely on the manipulate package on CRAN which will provide its own version.

Usage

```
rstudio_is_available()
```

Value

a logical

---

set.rseed   *Set seed in parallel compatible way*

Description

When the parallel package is used, setting the RNG seed for reproducibility involves more than
simply calling set.seed. set.rseed takes care of the additional overhead.

Usage

```
set.rseed(seed)
```

Arguments

seed   seed for the random number generator

Details

If the parallel package is not on the search path, then set.seed is called. If parallel is on the
search path, then the RNG kind is set to "L'Ecuyer-CMRG", the seed is set and mc.reset.stream
is called.

Examples

```
# These should give identical results, even if the `parallel' package is loaded.
set.rseed(123); do(3) * resample(1:10, 2)
set.rseed(123); do(3) * resample(1:10, 2)
```

---

sp2df   *Transforms a shapefile into a dataframe*

Description

This function takes in a shapefile (formal class of SpatialPolygonsDataFrame) and transforms it
into a dataframe

Usage

```
sp2df(map, ...)
```

Arguments

map   A map object of class SpatialPolygonsDataFrame

...   Other arguments, currently ignored

Value

A dataframe, in which the first 7 columns hold geographical information (ex: long and lat)

Examples

```
## Not run:
if(require(maptools)) {
  data(wrld_simpl)
  worldmap <- sp2df(wrld_simpl)
}

if ( require(ggplot2) && require(maptools) ) {
  data(wrld_simpl)
  World <- sp2df(wrld_simpl)
  World2 <- merge(World, Countries, by.x="NAME", by.y="maptools", all.y=FALSE)
  Mdata <- merge(Alcohol, World2, by.x="country", by.y="gapminder", all.y=FALSE)
  Mdata <- Mdata[order(Mdata$order),]
  qplot( x=long, y=lat, fill=ntiles(alcohol,5),
         data=subset(Mdata, year==2008), group = group,
                     geom="polygon")
}

## End(Not run)
```

---

standardName   *Standardization of Geographic Names*

Description

Often different sources of geographical data will use different names for the same region. These utilities make it easier to merge data from different sources by converting names to standardized forms.

Usage

```
standardName(x, standard, ignore.case = TRUE, returnAlternatives = FALSE,
  quiet = FALSE)

standardCountry(x, ignore.case = TRUE, returnAlternatives = FALSE,
  quiet = FALSE)

standardState(x, ignore.case = TRUE, returnAlternatives = FALSE,
  quiet = FALSE)
```

Arguments

x	A vector with the region names to standardize
standard	a named vector providing the map from non-standard names (names of vector) to standard names (values of vector)
ignore.case	a logical indicating whether case should be ignored when matching.
returnAlternatives	
	a logical indicating whether all alternatives should be returned in addition to the standard name.
quiet	a logical indicating whether warnings should be surpressed

Details

standardName This is the most general standardizing function. In addition to x, this function requires another argument: standard - a named vector in which each name is a particular spelling of the region name in question and the corresponding value is the standardized version of that region name

standardCountry This function will standardize the country names in x to the standard ISO_a3 country code format. If returnAlternatives is set to TRUE, this function will also return the the named vector used to standardize the country names

standardState This function will standardize the US state names in x to the standard two-letter abbreviations. If returnAlternatives is set to TRUE, this function will also return the the named vector used to standardize the state names

In all three cases, any names not found in standard will be left unaltered. Unless supressed, a warning message will indicate the number of such cases, if there are any.

StatSpline   *Geoms and stats for spline smoother*

Description

Functions to allow spline smoothing with ggplot2

Usage

```
StatSpline

stat_spline(mapping = NULL, data = NULL, geom = "line",
  position = "identity", na.rm = FALSE, show.legend = NA,
  inherit.aes = TRUE, weight = NULL, df = NULL, spar = NULL,
  cv = FALSE, all.knots = FALSE, nknots = stats::.nknots.smspl,
  df.offset = 0, penalty = 1, control.spar = list(), tol = NULL, ...)

geom_spline(mapping = NULL, data = NULL, stat = "spline",
  position = "identity", na.rm = FALSE, show.legend = NA,
  inherit.aes = TRUE, weight = NULL, df = NULL, spar = NULL,
  cv = FALSE, all.knots = FALSE, nknots = stats::.nknots.smspl,
  df.offset = 0, penalty = 1, control.spar = list(), tol = NULL, ...)
```

Arguments

`mapping`	An aesthetic mapping produced with [aes](){.}() or [aes_string]().
`data`	A data frame.
`geom`	A geom.
`position`	A position object.
`na.rm`	A logical indicating whether a warning should be issued when missing values are removed before plotting.
`show.legend`	A logical indicating whether legends should be included for this layer. If `NA`, legends will be inclued for each aesthetic that is mapped.
`inherit.aes`	A logical indicating whether aesthetics should be inherited. When `FALSE`, the supplied `mapping` will be the only aesthetics used.
`weight`	An optional vector of weights. See [smooth.spline]().
`df`	desired equivalent degrees of freedom. See [smooth.spline]() for details.
`spar`	A smoothing parameter, typically in (0,1]. See [smooth.spline]() for details.
`cv`	A logical. See [smooth.spline]() for details.
`all.knots`	A logical. See [smooth.spline]() for details.
`nknots`	An integer or function giving the number of knots to use when `all.knots = FALSE`. See [smooth.spline]() for details.
`df.offset`	A numerical value used to increase the degrees of freedom when using GVC. See [smooth.spline]() for details.
`penalty`	the coefficient of the penalty for degrees of freedom in the GVC cireterion. See [smooth.spline]() for details.
`control.spar`	An optional list used to control root finding when the parameter `spar` is computed. See [smooth.spline]() for details.
`tol`	A tolerance for same-ness or uniqueness of the x values. The values are binned into bins of size tol and values which fall into the same bin are regarded as the same. Must be strictly positive (and finite). When NULL, IQR(x) * 10e-6 is used.
`...`	Additional arguments
`stat`	A stat.

Format

An object of class `StatSpline` (inherits from `Stat`, `ggproto`) of length 3.

Examples

```
ggplot(Births) + geom_spline(aes(x = date, y=births, colour = wday))
ggplot(Births) + geom_spline(aes(x = date, y=births, colour = wday), nknots = 10)
```

---

statTally   *Tally test statistics*

Description

Tally test statistics from data and from multiple draws from a simulated null distribution

Usage

```
statTally(sample, rdata, FUN, direction = NULL, alternative = c("default",
  "two.sided", "less", "greater"), sig.level = 0.1, center = NULL,
  stemplot = dim(rdata)[direction] < 201, q = c(0.5, 0.9, 0.95, 0.99),
  fun = function(x) x, xlim, ...)
```

Arguments

sample	sample data
rdata	a matrix of randomly generated data under null hypothesis.
FUN	a function that computes the test statistic from a data set. The default value does nothing, making it easy to use this to tabulate precomputed statistics into a null distribution. See the examples.
direction	1 or 2 indicating whether samples in rdata are in rows (1) or columns (2).
alternative	one of default, two.sided, less, or greater
sig.level	significance threshold for wilcox.test used to detect lack of symmetry
center	center of null distribution
stemplot	indicates whether a stem plot should be displayed
q	quantiles of sampling distribution to display
fun	same as FUN so you don't have to remember if it should be capitalized
xlim	limits for the horizontal axis of the plot.
...	additional arguments passed to histogram

Value

A lattice plot showing the sampling distribution.

As side effects, information about the empirical sampling distribution and (optionally) a stem plot are printed to the screen.

Examples

```
# is my spinner fair?
x <- c(10, 18, 9, 15)    # counts in four cells
rdata <- rmultinom(999, sum(x), prob=rep(.25, 4))
statTally(x, rdata, fun=max)  # unusual test statistic
statTally(x, rdata, fun=var)  # equivalent to chi-squared test
# Can also be used with test stats that are precomputed.
if (require(mosaicData)) {
D <- diffmean( age ~ sex, data=HELPrct); D
nullDist <- do(999) * diffmean( age ~ shuffle(sex), data=HELPrct)
statTally( D, nullDist)
}
```

---

surround   *Format strings for pretty output*

Description

Format strings for pretty output

Usage

```
surround(x, pre = " ", post = " ", width = 8, ...)
```

Arguments

x	a vector
pre	text to prepend onto string
post	text to postpend onto string
width	desired width of string
...	additional arguments passed to [format](#)

Value

a vector of strings padded to the desired width

Examples

```
surround(rbinom(10,20,.5), " ", " ", width=4)
surround(rnorm(10), " ", " ", width=8, digits = 2, nsmall = 2)
```

---

swap   *Swap values among columns of a data frame*

Description

Swap values among columns of a data frame

Usage

```
swap(data, which)
```

Arguments

data	a data frame
which	a formula or an integer or character vector specifying columns in `data`

Details

swap is not a particularly speedy function. It is intended primarily as an aid for teaching randomization for paired designs. Used this way, the number of randomizations should be kept modest (approximately 1000) unless you are very patient.

Examples

```
if (require(tidyr)) {
  Sleep2 <- sleep %>% spread( key=group, val=extra )
  names(Sleep2) <- c("subject", "drug1", "drug2")
  swap(Sleep2, drug1 ~ drug2)
  mean( ~(drug1 - drug2), data=Sleep2)
  do(3) * mean( ~(drug1 - drug2), data=Sleep2 %>% swap(drug1 ~ drug2) )
}
```

---

symbolicD   *Symbolic Derivatives*

Description

Constructs symbolic derivatives of some mathematical expressions

Usage

```
symbolicD(formula, ..., .order = NULL)
```

Arguments

`formula`	a mathematical expression (see examples and [`plotFun`](plotFun))
`.order`	a number specifying the order of a derivative with respect to a single variable
`...`	additional parameters, typically default values for mathematical parameters

Details

Uses the built-in symbolic differentiation function to construct a formula for the derivative and packages this up as a function. The `.order` argument is just for convenience when programming high-order derivatives, e.g. the 5th derivative w.r.t. one variable.

Value

a function implementing the derivative

Author(s)

Daniel Kaplan (<kaplan@macalester.edu>)

See Also

[`D`](D), [`numD`](numD), [`makeFun`](makeFun), [`antiD`](antiD), [`plotFun`](plotFun)

Examples

```
symbolicD( a*x^2 ~ x)
symbolicD( a*x^2 ~ x&x)
symbolicD( a*sin(x)~x, .order=4)
symbolicD( a*x^2*y+b*y ~ x, a=10, b=100 )
```

---

symbolicInt	*Find the symbolic integral of a formula*

Description

Find the symbolic integral of a formula

Use recursion to find a symbolic antiderivative

Attempts symbolic integration of some mathematical/arithmetical forms

Attempts symbolic integration of some mathematical forms

Attempts symbolic integration of some mathematical forms using trigonometric substitution

Takes a call and returns its affine coefficients.

Usage

```
symbolicInt(form, ...)

symbolicAntiD(form, ...)

.intArith(form, ...)

.intMath(form, ...)

.intTrig(form, num, den, .x.)

.affine.exp(tree, .x.)
```

Arguments

`form`	an object of type formula to be integrated. Rhs of formula indicates which variable to integrate with respect to. Must only have one variable.
`num`	numerator
`den`	denominator
`.x.`	the variable name
`tree`	the expression to be analyzed
`...`	extra parameters

Details

This symbolic integrator recognizes simple polynomials and functions such as `sin`, `cos`, `tan`, `sinh`, `cosh`, `tanh`, `sqrt`, and `exp`.

It will not perform more complicated substitutions or integration by parts.

Value

symbolicInt returns a function whose body is the symbolic antiderivative of the formula. If this method does not recognize the formula, it will return an error.

a formula implementing giving symbolic anti-derivative. If the formula isn't found by the algorithm, an error is thrown.

An expression with the integral, or throws an error if unsuccessful.

An expression with the integral, or throws an error if unsuccessful.

An expression with the integral, or throws an error if unsuccessful.

A list with values of a and b satisfying a*.x.+b = tree. If the expression is not affine, returns an empty list.

---

tally   *Tabulate categorical data*

Description

Tabulate categorical data

Usage

```
tally(x, ...)

## S3 method for class 'tbl'
tally(x, wt, sort = FALSE, ..., envir = parent.frame())

## S3 method for class 'data.frame'
tally(x, wt, sort = FALSE, ..., envir = parent.frame())

## S3 method for class 'formula'
tally(x, data = parent.frame(2), format = c("count",
  "proportion", "percent", "data.frame", "sparse", "default"),
  margins = FALSE, quiet = TRUE, subset, groups = NULL, useNA = "ifany",
  groups.first = FALSE, ...)
```

Arguments

x	an object
...	additional arguments passed to [table](#)
wt	for weighted tallying, see [tally](#) in **dplyr**
sort	a logical, see [tally](#) in **dplyr**
envir	an environment in which to evaluate
data	a data frame or environment in which evaluation occurs. Note that the default is data=parent.frame(). This makes it convenient to use this function interactively by treating the working envionment as if it were a data frame. But this may not be appropriate for programming uses. When programming, it is best to use an explicit data argument – ideally supplying a data frame that contains the variables mentioned
format	a character string describing the desired format of the results. One of `'default'`, `'count'`, `'proportion'`, `'percent'`, `'data.frame'`, `'sparse'`, or `'default'`. In case of `'default'`, counts are used unless there is a condition, in which case proportions are used instead. Note that prior to version 0.9.3, `'default'` was the default, now it is `'count'`. `'data.frame'` converts the table to a data frame with one row per cell; `'sparse'` additionally removes any rows with 0 counts.
margins	a logical indicating whether marginal distributions should be displayed.
quiet	a logical indicating whether messages about order in which marginal distributions are calculated should be surpressed. See [addmargins](#).

subset	an expression evaluating to a logical vector used to select a subset of data
groups	used to specify a condition as an alternative to using a formula with a condition.
useNA	as in `table`, but the default here is `"ifany"`.
groups.first	a logical indicating whether groups should be inserted ahead of the condition (else after).

Details

The **dplyr** package also exports a `tally` function. If x inherits from class `"tbl"` or `"data frame"`, then **dplyr**'s `tally()` is called. This makes it easier to have the two packages coexist.

Otherwise, tally() is designed as an alternative to `table()` and `xtabs()`. The primary use case it to describe a (possibly multi-dimensional) table using a formula. For a table of counts, each component of the formala becomes one of the dimensions of the cross table. For tables of proportions or percents, conditional proportions and percents are computed, conditioned on each level of all "secondary" (i.e., conditioning) variables, defined as everything other than the left hand side, if there is a left hand side to the formala; and everything except the right hand side if the left hand side of the formula is empty. Note that groups is folded into the formula prior to this determination and becomes part of the conditioning.

When marginal totals are added, they are added for all of the conditioning dimensions, and proportions should sum to 1 for each level of the conditioning variables. This can be useful to make it clear which conditional proportions are beign computed.

See the examples for some typical use cases.

Value

A object of class `"table"`, unless passing through to **dplyr** or converted to a data frame because format is `"data.frame"` or `"sparse"`.

Note

The curent implementation when format = `"sparse"` first creates the full data frame and then removes the unneeded rows. So the savings is in terms of space, not time.

Examples

```
tally( ~ substance, data = HELPrct)
tally( ~ substance + sex , data = HELPrct)
tally( sex ~ substance, data = HELPrct)   # equivalent to tally( ~ sex | substance, ... )
tally( ~ substance | sex , data = HELPrct)
tally( ~ substance | sex , data = HELPrct, format = 'count', margins = TRUE)
tally( ~ substance + sex , data = HELPrct, format = 'percent', margins = TRUE)
tally( ~ substance | sex , data = HELPrct, format = 'percent', margins = TRUE)
# force NAs to show up
tally( ~ sex, data = HELPrct, useNA = "always")
# show NAs if any are there
tally( ~ link, data = HELPrct)
# ignore the NAs
tally( ~ link, data = HELPrct, useNA = "no")
```

---

theme.mosaic *Lattice Theme*

Description

A theme for use with lattice graphics.

Usage

```
theme.mosaic(bw = FALSE, lty = if (bw) 1:7 else 1, lwd = 2, ...)

col.mosaic(bw = FALSE, lty = if (bw) 1:7 else 1, lwd = 2, ...)
```

Arguments

bw	whether color scheme should be "black and white"
lty	vector of line type codes
lwd	vector of line widths
...	additional named arguments passed to trellis.par.set

Value

Returns a list that can be supplied as the theme to trellis.par.set().

Note

These two functions are identical. col.mosaic is named similarly to col.whitebg, but since more than just colors are set, theme.mosaic is a preferable name.

See Also

trellis.par.set, show.settings

Examples

```
trellis.par.set(theme=theme.mosaic())
show.settings()
trellis.par.set(theme=theme.mosaic(bw=TRUE))
show.settings()
```

---

theme_map   *ggplot2 theme for maps*

Description

A very plain **ggplot2** theme that is good for maps.

Usage

```
theme_map(base_size = 12)
```

Arguments

base_size   the base font size for the theme.

Details

This theme is largely based on an example posted by Winston Chang at the **ggplot2** Google group forum.

TukeyHSD.lm   *Additional interfaces to TukeyHSD*

Description

[TukeyHSD](#) requires use of [aov](#). Since this is a hinderence for beginners, wrappers have been provided to remove this need.

Usage

```
## S3 method for class 'lm'
TukeyHSD(x, which, ordered = FALSE, conf.level = 0.95, ...)

## S3 method for class 'formula'
TukeyHSD(x, which, ordered = FALSE, conf.level = 0.95,
  data = parent.frame(), ...)
```

Arguments

x   an object, for example of class lm or formula

which, ordered, conf.level, ...
                   just as in [TukeyHSD](#) from the base package

data   a data frame. NB: This does not come second in the argument list.

Examples

```
## These should all give the same results
if (require(mosaicData)) {
  model <- lm(age ~ substance, data=HELPrct)
  TukeyHSD(model)
  TukeyHSD( age ~ substance, data=HELPrct)
  TukeyHSD(aov(age ~ substance, data=HELPrct))
}
```

---

t_test   *Student's t-Test*

Description

Performs one and two sample t-tests. The mosaic `t.test` provides wrapper functions around the function of the same name in **stats**. These wrappers provide an extended interface that allows for a more systematic use of the formula interface.

Usage

```
t_test(x, y = NULL, ..., data = parent.frame())

t.test(x, y=NULL, ..., data = parent.frame())
```

Arguments

x	a formula or a non-empty numeric vector
y	an optional non-empty numeric vector or formula
data	a data frame
...	additional arguments, see `t.test` in the **stats** package. When x is a formula, groups can be used to compare groups: x = ~ var, groups=g is equivalent to x = var ~ g . See the examples.

Details

This is a wrapper around `t.test` from the **stats** package to extend the functionality of the formula interface. In particular, one can now use the formula interface for a 1-sample t-test. Before, the formula interface was only permitted for a 2-sample test. The type of formala that can be used for the 2-sample test has also be broadened. See the examples.

Value

an object of class `htest`

See Also

[prop.test](#), [binom.test](#), [t.test](#)

Examples

```
if (require(mosaicData)) {
  t.test(~ age, data=HELPrct)
  t.test(age ~ sex, data=HELPrct)
  t.test(~ age | sex, data=HELPrct)
  t.test(~ age, groups=sex, data=HELPrct)
}
```

---

update_ci   *Update confidence interval*

Description

Update the confidence interval portion of an object returned from binom.test using one of several alternative methods.

Usage

```
update_ci(object, method = c("clopper-pearson", "wald", "agresti-coull",
  "plus4", "score"))
```

Arguments

object	An "htest" object produced by binom.test
method	a method for computing a confidence interval for a propotion.

Value

an "htest" object with an updated confidence interval

See Also

binom.test

---

value   *Extract value from an object*

Description

Functions like integrate() and nlm() return objects that contain more information that simply the value of the integration or optimization. value() extracts the primary value from such objects. Currenlty implemented situations include the output from integrate(), nlm(), adaptIntegrate(), and uniroot().

Usage

```
value(object, ...)

## S3 method for class 'integrate'
value(object, ...)

## Default S3 method:
value(object, ...)
```

Arguments

object   an object from which a "value" is to be extracted.

...   additional arguments (currently ignored).

Examples

```
integrate(sin, 0, 1) %>% value()
nlm(cos, p = 0) %>% value()
uniroot(cos, c(0, 2)) %>% value()
```

---

vector2df *Convert a vector to a data frame*

Description

Convert a vector into a 1-raw data frame using the names of the vector as column names for the data frame

Usage

```
vector2df(x, nice_names = FALSE)
```

Arguments

x   a vector

nice_names   a logical indicating whether names should be nicified

Value

a data frame

Examples

```
vector2df(c(1, b = 2, `(Intercept)` = 3))
vector2df(c(1, b = 2, `(Intercept)` = 3), nice_names = TRUE)
```

---

xchisq.test   *Augmented Chi-squared test*

Description

This augmented version of [chisq.test](#) provides more verbose output.

Usage

```
xchisq.test(x, y = NULL, correct = TRUE, p = rep(1/length(x), length(x)),
  rescale.p = FALSE, simulate.p.value = FALSE, B = 2000)
```

Arguments

```
x, y, correct, p, rescale.p, simulate.p.value, B
```
                 as in [chisq.test](#).

See Also

[chisq.test](#)

Examples

```
# Physicians' Health Study data
phs <- cbind(c(104,189),c(10933,10845))
rownames(phs) <- c("aspirin","placebo")
colnames(phs) <- c("heart attack","no heart attack")
phs
xchisq.test(phs)
```

---

xhistogram   *Augmented histograms*

Description

The **mosaic** package adds some additional functionality to [histogram](#)(), making it simpler to
obtain certain common histogram adornments. This is done be resetting the default panel and
prepanel functions used by histogram.

Usage

```
xhistogram(...)

xhistogramBreaks(x, center = NULL, width = NULL, nint, ...)

prepanel.xhistogram(x, breaks = xhistogramBreaks, ...)

panel.xhistogram(x, dcol = trellis.par.get("plot.line")$col, dalpha = 1,
  dlwd = 2, gcol = trellis.par.get("add.line")$col, glwd = 2,
  fcol = trellis.par.get("superpose.polygon")$col, dmath = dnorm,
  verbose = FALSE, dn = 100, args = NULL, labels = FALSE,
  density = NULL, under = FALSE, fit = NULL, start = NULL,
  type = "density", v, h, groups = NULL, center = NULL, width = NULL,
  breaks, nint = round(1.5 * log2(length(x)) + 1), stripes = c("vertical",
  "horizontal", "none"), alpha = 1, ...)
```

Arguments

x	a formula or a numeric vector
center	center of one of the bins
width	width of the bins
nint	approximate number of bins
breaks	break points for histogram bins, a function for computing such, or a method hist knows about given as a character string. When using the **mosaic** package defaults, xhistogramBreaks is used.
dcol	color of density curve
dalpha	alpha for density curve
dlwd, glwd	like `lwd` but affecting the density line and guide lines, respectively
gcol	color of guidelines
fcol	fill color for histogram rectangles
dmath	density function for density curve overlay
verbose	be verbose?
dn	number of points to sample from density curve
args	a list of additional arguments for `dmath`
labels	should counts/densities/precents be displayed or each bin?
density	a logical indicating whether to overlay a density curve
under	a logical indicating whether the density layers should be under or over other layers of the plot.
fit	a character string describing the distribution to fit. Known distributions include "exponential", "normal", "lognormal" , "poisson", "beta", "geometric", "t", "weibull", "cauchy", "gamma", "chisq", and "chi-squared"
start	numeric value passed to fitdistr

type	one of `'density'`, `'count'`, or `'percent'`
h, v	a vector of values for additional horizontal and vertical lines
groups	as per `histogram`
stripes	one of ″vertical″, ″horizontal″, or ″none″, indicating how bins should be striped when `groups` is not NULL
alpha	transparency level
panel	a panel function
...	additional arguments passed from `histogram` to the panel function; by default when the **mosaic** package has been loaded this will `panel.xhistogram`.

Details

The primary additional functionality added to `histogram`() are the arguments `width` and `center` which provide a simple way of describing equal-sized bins, and `fit` which can be used to overlay the density curve for one of several distributions. The `groups` argument can be used to color the bins. The primary use for this is to shade tails of histograms, but there may be other uses as well.

Value

`xhistogramBreaks` returns a vector of break points

Note

Versions of **lattice** since 0.20-21 support setting custom defaults for `breaks`, `panel`, and `prepanel` used by `histogram()`, so `xhistogram()` is no longer needed. As a result, `xhistogram()` (which was required in earlier versions of **mosaic** is no longer needed and has been removed.

See Also

`histogram`, `mosaicLatticeOptions`(), and `restoreLatticeOptions`().

Examples

```
if (require(mosaicData)) {
histogram(~age | substance, HELPrct, v=35, fit='normal')
histogram(~age, HELPrct, labels=TRUE, type='count')
histogram(~age, HELPrct, groups=cut(age, seq(10,80,by=10)))
histogram(~age, HELPrct, groups=sex, stripes='horizontal')
histogram(~racegrp, HELPrct, groups=substance,auto.key=TRUE)
xhistogramBreaks(1:10, center=5, width=1)
xhistogramBreaks(1:10, center=5, width=2)
xhistogramBreaks(0:10, center=15, width=3)
xhistogramBreaks(1:100, center=50, width=3)
xhistogramBreaks(0:10, center=5, nint=5)
}
```

---

xpnorm *Augmented versions of pnorm and qnorm*

Description

These functions behave similarly to the functions with the initial x removed from their names but add more verbose output and graphics.

Usage

```
xpnorm(q, mean = 0, sd = 1, plot = TRUE, verbose = TRUE,
  invisible = FALSE, digits = 4, lower.tail = TRUE, log.p = FALSE,
  xlim = mean + c(-4, 4) * sd, ylim = c(0, 1.4 * dnorm(mean, mean, sd)),
  vlwd = 2, vcol = trellis.par.get("add.line")$col, rot = 45,
  manipulate = FALSE, ...)

xqnorm(p, mean = 0, sd = 1, plot = TRUE, verbose = TRUE, digits = 4,
  lower.tail = TRUE, log.p = FALSE, xlim, ylim, invisible = FALSE,
  vlwd = 2, vcol = trellis.par.get("add.line")$col, rot = 45, ...)
```

Arguments

q	quantile
mean, sd	parameters of normal distribution.
plot	logical. If TRUE, show an illustrative plot.
verbose	logical. If TRUE, display verbose output.
invisible	logical. If TRUE, return value invisibly.
digits	number of digits to display in output.
lower.tail	logical. If FALSE, use upper tail probabilities.
log.p	logical. If TRUE, uses the log of probabilities.
xlim, ylim	limits for plotting.
vlwd, vcol	line width and color for vertical lines.
rot	angle of rotation for text labels.
manipulate	logical. If TRUE and in RStudio, then sliders are added for interactivity.
p	probability
...	additional arguments.

See Also

[histogram](), [chisq.test](), [pnorm](), [qnorm](), [qqmath](), and [plot]().

Examples

```
xpnorm(650, 500, 100)
xqnorm(.75, 500, 100)
## Not run:
if (rstudio_is_available() & require(manipulate)) {
  manipulate(xpnorm(score, 500, 100, verbose=verbose),
    score = slider(200,800),
   verbose = checkbox(TRUE, label="Verbose Output")
  )
}

## End(Not run)
```

---

xqqmath   *Augmented version of* qqmath

Description

Augmented version of qqmath

Usage

```
xqqmath(x, data = NULL, panel = "panel.xqqmath", ...)

panel.xqqmath(x, qqmathline = !(fitline || idline), idline = FALSE,
  fitline = NULL, slope = NULL, intercept = NULL, overlines = FALSE,
  groups = NULL, ..., col.line = trellis.par.get("add.line")$col,
  pch = 16, lwd = 2, lty = 2)
```

Arguments

x, data, panel, xqqmath, ...	as in qqmath
qqmathline	a logical: should line be displayed passing through first and third quartiles?
idline	a logical; should the line y=x be added to the plot?
fitline	a logical; should a ftted line be added to plot? Such a line will use slope and intercept if provided, else the standard deviation and mean of the data. If slope is specified, the line will be added unless fitline is FALSE.
slope	slope for added line
intercept	intercept for added line
overlines	a logical: should lines be on top of qq plot?
groups, pch, lwd, lty	as in lattice plots
col.line	color to use for added lines

Value

a trellis object

Examples

```
x <- rnorm(100)
xqqmath( ~ x)                  # with quartile line
xqqmath( ~ x, fitline = TRUE)  # with fitted line
xqqmath( ~ x, idline = TRUE)   # with y = x
x <- rexp(100, rate = 10)
xqqmath( ~ x, distribution = qexp)     # with quartile line
xqqmath( ~ x, distribution = qexp, slope = 1/10)
xqqmath( ~ x, distribution = qexp, slope = mean(x))
```

---

xyz2latlon   *Convert back and forth between latitude/longitude and XYZ-space*

Description

Convert back and forth between latitude/longitude and XYZ-space

Usage

```
xyz2latlon(x, y, z)

latlon2xyz(latitude, longitude)

lonlat2xyz(longitude, latitude)
```

Arguments

x, y, z   numeric vectors

latitude, longitude

    vectors of latitude and longitude values

Value

a matrix each row of which describes the latitudes and longitudes

a matrix each row of which contains the x, y, and z coordinates of a point on a unit sphere

See Also

[deg2rad](), [googleMap](), and [rgeo]().

Examples

```
xyz2latlon(1, 1, 1)     # point may be on sphere of any radius
xyz2latlon(0, 0, 0)     # this produces a NaN for latitude
latlon2xyz(30, 45)
lonlat2xyz(45, 30)
```

---

zscore   *Compute z-scores*

Description

Compute z-scores

Usage

```
zscore(x, na.rm = getOption("na.rm", FALSE))
```

Arguments

x	a numeric vector
na.rm	a logical indicating whether missing values should be removed

Examples

```
iris %>%
  group_by(Species) %>%
  mutate(zSepal.Length = zscore(Sepal.Length)) %>%
  head()
```

Index