Package 'utilities'

November 28, 2021

Type Package

Title Data Utility Functions

Version 0.4.0

Date 2021-06-09

Author Ben O'Neill [aut, cre]

Maintainer Ben O'Neill <ben.oneill@hotmail.com>

URL https://github.com/ben-oneill/utilities/

Description Data utility functions for use in probability and statistics. Includes functions for computing higher-moments for samples and their decompositions.
Also includes utilities to examine functional mappings between factor variables and other variables in a data set.

License MIT + file LICENSE

Encoding UTF-8

Imports stats

Suggests ggplot2, ggdag (>= 0.2.4), gmp, gridExtra, matrixStats

RoxygenNote 7.1.1

NeedsCompilation no

Repository CRAN

Date/Publication 2021-11-28 15:20:02 UTC

R topics documented:

`datasets.str` *Structure of Available Datasets*

Description

`datasets.str` returns the structure of available datasets

Usage

```
datasets.str(package = NULL)
```

Arguments

`package`	The package/packages containing the datasets of interest

Details

Datasets are often available in packages loaded into `R` and it is useful to know the structure of these datasets. This function shows the user the strucure of all available datasets in a specified package or packages. (If the user does not specify a `package`) then the function searches over all available packages.

Value

A data frame listing available data sets, invisibly

Examples

```
datasets.str("datasets")
```

kurtosis *Sample Kurtosis*

Description

kurtosis returns the sample kurtosis of a data vector/matrix

Usage

```
kurtosis(x, kurt.type = NULL, kurt.excess = FALSE, na.rm = FALSE)
```

Arguments

x	A data vector/matrix
kurt.type	The type of kurtosis statistic used ('Moment', 'Fisher Pearson' or 'Adjusted Fisher Pearson')
kurt.excess	Logical value; if TRUE the function gives the excess kurtosis (instead of raw kurtosis)
na.rm	Logical value; if TRUE the function removes NA values

Details

This function computes the sample kurtosis for a data vector or matrix. For a vector input the function returns a single value for the sample kurtosis of the data. For a matrix input the function treats each column as a data vector and returns a vector of values for the sample kurtosis of each of these datasets. The function can compute different types of kurtosis statistics using the kurt.type input.

Value

The sample kurtosis of the data vector/matrix

Examples

```
kurtosis(rnorm(1000))
kurtosis(rexp(1000))
```

log *Logarithm Function*

Description

log returns the logarithm of the input

Usage

```
log(x, base = exp(1), gradient = FALSE, hessian = FALSE)

log2(x, gradient = FALSE, hessian = FALSE)

log10(x, gradient = FALSE, hessian = FALSE)
```

Arguments

x	An input value (numeric/complex scalar or vector)
base	The base for the logarithm (a positive scalar value)
gradient	Logical; if TRUE the output will include a 'gradient' attribute
hessian	Logical; if TRUE the output will include a 'hessian' attribute

Details

This version of the logarithm function allows both numeric and complex inputs, including negative numeric values. If the output of the logarithm has no complex part then the output is given as a numeric value. It also allows the user to generate the gradient and Hessian.

Value

The logarithm of the input

Examples

```
log(1)
log(-1)
log10(-10, TRUE, TRUE)
```

mappings *Examine mappings between factor variables in a data-frame*

Description

mappings determines the mappings between factor variables in a data-frame

Usage

```
mappings(data, na.rm = TRUE, all.vars = FALSE, plot = TRUE)
```

Arguments

data	A data-frame (or an object coercible to a data-frame)
na.rm	Logical value; if TRUE the function removes NA values from consideration
all.vars	Logical value; if TRUE the function only examines factor variables in the data-frame; if FALSE the function examines all variables in the data-frame (caution is required in interpretation of output)
plot	Logical value; if TRUE the function plots the DAG for the mappings (requires ggplot2 and ggdag to work)

Details

In preliminary data analysis prior to statistical modelling, it is often useful to investigate whether there are mappings between factor variables in a data-frame in order to see if any of these factor variables are redundant (i.e., fully determined by other factor variables). This function takes an input data-frame data and examines whether there are any mappings between the factor variables. (Note that the function will interpret all character variables as factors but will not interpret numeric or logical variables as factors.) The output is a list showing the uniqueness of the binary relations between the factor variables (a logical matrix showing left-uniqueness in the binary relations), the mappings between factor variables, the redundant and non-redundant factor variables, and the directed acyclic graph (DAG) of these mappings (the last element requires the user to have the ggdag package installed; it is omitted if the package is not installed). If plot = TRUE the function also returns a plot of the DAG (if ggdag and ggplot2 packages are installed).

Note that the function also allows the user to examine mappings between all variables in the data-frame (i.e., not just the factor variables) by setting all.vars = TRUE. The output from this analysis should be interpreted with caution; one-to-one mappings between non-factor variables are common (e.g., when two variables are continuous it is almost certain that they will be in a one-to-one mapping), and so the existence of a mapping may not be indicative of variable redundancy.

Note on operation: If na.rm = FALSE then the function analyses the mappings between the factors/variables without removing NA values. In this case an NA value is treated as a missing value that could be any outcome. Consequently, for purposes of determining whether there is a mapping between the variables, an NA value is treated as if it were every possible value. The mapping is falsified if there are at least two identical values in the domain (which may include one or more NA values) that map to different values in the codomain (which may include one or more NA values).

Value

A list object of class 'mappings' giving information on the mappings between the variables

Examples

```
DATA <- data.frame(
  VAR1 = c(0,1,2,2,0,1,2,0,0,1),
  VAR2 = c('A','B','B','B','A','B','B','A','A','B'),
  VAR3 = 1:10,
  VAR4 = c('A','B','C','D','A','B','D','A','A','B'),
  VAR5 = c(1:5,1:5)
)

# Apply mappings
mappings(DATA, all.vars = TRUE, plot = FALSE)
```

---

moments *Sample Moments*

Description

moments returns the sample moments of a data vector/matrix

Usage

```
moments(
  x,
  skew.type = NULL,
  kurt.type = NULL,
  kurt.excess = FALSE,
  na.rm = TRUE,
  include.sd = FALSE
)
```

Arguments

x	A data vector/matrix/list
skew.type	The type of kurtosis statistic used ('Moment', 'Fisher Pearson' or 'Adjusted Fisher Pearson')
kurt.type	The type of kurtosis statistic used ('Moment', 'Fisher Pearson' or 'Adjusted Fisher Pearson')
kurt.excess	Logical value; if TRUE the function gives the excess kurtosis (instead of raw kurtosis)
na.rm	Logical value; if TRUE the function removes NA values
include.sd	Logical value; if TRUE the output includes a column for the sample standard deviation (if needed)

Details

This function computes the sample moments for a data vector, matrix or list (sample mean, sample variance, sample skewness and sample kurtosis). For a vector input the function returns a single value for each sample moment of the data. For a matrix or list input the function treats each column/element as a data vector and returns a matrix of values for the sample moments of each of these datasets. The function can compute different types of skewness and kurtosis statistics using the `skew.type`, `kurt.type` and `kurt.excess` inputs. (For details on the different types of skewness and kurtosis statistics, see Joanes and Gill 1998.)

Value

A data frame containing the sample moments of the data vector/matrix

Examples

```
#Create some subgroups of mock data and a pooled dataset
set.seed(1)
N    <- c(28, 44, 51)
SUB1 <- rnorm(N[1])
SUB2 <- rnorm(N[2])
SUB3 <- rnorm(N[3])
DATA <- list(Subgroup1 = SUB1, Subgroup2 = SUB2, Subgroup3 = SUB3)
POOL <- c(SUB1, SUB2, SUB3)

#Compute sample moments for subgroups and pooled data
MOMENTS <- moments(DATA)
POOLMOM <- moments(POOL)

#Compute pooled moments via sample decomposition
sample.decomp(moments = MOMENTS)
```

---

nlm.prob *Nonlinear minimisation/maximisation allowing probability vectors as inputs*

Description

`nlm.prob` minimises/maximises a function allowing probability vectors as inputs

Usage

```
nlm.prob(
  f,
  p,
  prob.vectors = list(1:length(p)),
  ...,
  lambda = 1,
  eta0max = 1e+10,
```

```
  maximise = FALSE,
  maximize = maximise,
  hessian = FALSE,
  typsize = rep(1, length(p)),
  fscale = 1,
  print.level = 0,
  ndigit = 12,
  gradtol = 1e-06,
  stepmax = max(1000 * sqrt(sum((p/typsize)^2)), 1000),
  steptol = 1e-06,
  iterlim = 100,
  check.analyticals = TRUE
)
```

Arguments

f	The objective function to be minimised; output should be a single numeric value.
p	Starting argument values for the minimisation.
prob.vectors	A list specifying which sets of elements are constrained to be a probability vector (each element in the list should be a vector specifying indices in the argument vector; elements cannot overlap into multiple probability vectors).
...	Additional arguments to be passed to f via nlm
lambda	The tuning parameter used in the softmax transformation for the optimisation (a single positive numeric value).
eta0max	The maximum absolute value for the elements of eta0 (the starting value in the unconstrained optimisation problem).
maximise, maximize	Logical value; if TRUE the function maximises the objective function instead of mimimising.
hessian	Logical; if TRUE then the output of the function includes the Hessian of f at the minimising point.
typsize	An estimate of the size of each parameter at the minimum.
fscale	An estimate of the size of f at the minimum.
print.level	This argument determines the level of printing which is done during the minimisation process. The default value of 0 means that no printing occurs, a value of 1 means that initial and final details are printed and a value of 2 means that full tracing information is printed.
ndigit	The number of significant digits in the function f.
gradtol	A positive scalar giving the tolerance at which the scaled gradient is considered close enough to zero to terminate the algorithm. The scaled gradient is a measure of the relative change in f in each direction p[i] divided by the relative change in p[i].
stepmax	A positive scalar which gives the maximum allowable scaled step length. stepmax is used to prevent steps which would cause the optimisation function to overflow, to prevent the algorithm from leaving the area of interest in parameter space, or

to detect divergence in the algorithm. stepmax would be chosen small enough to prevent the first two of these occurrences, but should be larger than any anticipated reasonable step.

steptol A positive scalar providing the minimum allowable relative step length.

iterlim A positive integer specifying the maximum number of iterations to be performed before the routine is terminated.

check.analyticals
Logical; if TRUE then the analytic gradients and Hessians (if supplied) are checked against numerical derivatives at the initial parameter values. This can help detect incorrectly formulated gradients or Hessians.

Details

This is a variation of the stats::nlm function for nonlinear minimisation. The present function is designed to minimise an objective function with one or more arguments that are probability vectors. (The objective function may also have other arguments that are not probability vectors.) The function uses the same inputs as the stats::nlm function, except that the user can use the input prob.vectors to specify which inputs are constrained to be probability vectors. This input is a list where each element in the list specifies a set of indices for the argument of the objective function; the specified set of indices is constrained to be a probability vector (i.e., each corresponding argument is non-negative and the set of these arguments must sum to one). The input prob.vectors may list one or more probability vectors, but they must use disjoint elements of the argument (i.e., a variable in the argument cannot appear in more than one probability vector).

Optimisation is performed by first converting the objective function into unconstrained form using the softmax transformation and its inverse to convert from unconstrained space to probability space and back. Optimisation is done on the unconstrained objective function and the results are converted back to probability space to solve the constrained optimisation problem. For purposes of conversion, this function allows specification of a tuning parameter lambda for the softmax and inverse- softmax transformations. (This input can either be a single tuning value used for all conversions, or a vector of values for the respective probability vectors; if the latter, there must be one value for each element of the prob.vector input.)

Most of the input descriptions below are adapted from the corresponding descriptions in stat::nlm, since our function is a wrapper to that function. The additional inputs for this function are prob.vectors, lambda and eta0max. The function also adds an option maximise to conduct maximisation instead of minimisation.

Value

A list showing the computed minimising point and minimum of f and other related information.

Examples

```
x <- rbinom(100, 1, .2)
nlm.prob(function(p) sum(dbinom(x,1,p[2],log=TRUE)), c(.5, .5), maximise = TRUE)
```

PFD *Prime Factor Decomposition (PFD)*

Description

PFD converts a positive integer to its prime-factor decomposition or *vice versa*

Usage

```
PFD(x)

## S3 method for class 'prime.factor.decomposition'
print(x, quote = FALSE, ...)
```

Arguments

x	An input vector/matrix/array (can be a vector of integers/bigz or PFDs)
quote	logical, indicating whether or not strings should be printed with surrounding quotes.
...	further arguments passed to or from other methods.

Details

This function converts a vector of integers to a corresponding character vector giving the prime-factor decomposition in a condensed form. The input can be a vector of integers or a 'bigz' vector containing large integers. In either case the function returns the corresponding vector of the prime-factor decomposition (PFD) values, written in a condensed character form. The function also converts back from the PFD form to an integer/bigz vector.

This function depends on the gmp package.

Value

If the input is integer/bigz then the output is the PFD; if the input is PFD then the output is integer/bigz

Examples

```
PFD(1:10)
stopifnot(all.equal(1:100, PFD(PFD(1:100))))
```

plot.data.mappings *Plot components from data mapping*

Description

This needs ggplot2 and ggdag to function correctly.

Usage

```
## S3 method for class 'data.mappings'
plot(x, node.size = 1, text.size = 1, line.width = 1, ...)
```

Arguments

x	a data mapping
node.size	node size
text.size	label size for a node
line.width	line width
...	not used

Value

nothing

queue *Generate queuing information from arrival and use times*

Description

queue returns queuing information for users and service facilities.

Usage

```
queue(
  n,
  arrive,
  use.full,
  wait.max = NULL,
  revive = 0,
  close.arrive = Inf,
  close.service = Inf,
  close.full = Inf
)
```

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

## S3 method for class 'queue'
plot(
  x,
  print = TRUE,
  gap = NULL,
  line.width = 2,
  line.colors = NULL,
  line.colours = line.colors,
  ...
)

## S3 method for class 'queue'
summary(object, probs = NULL, probs.decimal.places = 2, ...)

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

## S3 method for class 'summary.queue'
plot(
  x,
  print = TRUE,
  count = FALSE,
  bar.colors = NULL,
  bar.colours = bar.colors,
  ...
)
```

Arguments

n	Number of service facilities at the amenity (positive integer)
arrive	Vector of arrival-times for the users (non-negative numeric values)
use.full	Vector of (intended) use-times for the users (non-negative numeric values)
wait.max	Vector of maximum-waiting-times for the users (non-negative numeric values)
revive	Revival-time for service facilities
close.arrive	Closure-time for new arrivals (no new arrivals allowed)
close.service	Closure-time for new services (no new services allowed)
close.full	Closure-time for all services (all existing services are terminated)
x, object	a queue object
...	further arguments passed to or from other methods.
print, gap, line.width, line.colors, line.colours	plotting paramaters
probs	summary quantiles to be included in output.

`probs.decimal.places`
: rounds the output to specified number of decimal places.

`count`
: absolute or relative frequencies

`bar.colors, bar.colours`
: plotting parameters

Details

This function computes takes inputs giving the arrival times and (intended) use times for a set of users at an amenity, plus the number of service facilities at the amenity. The function computes full information on the use of the facilities by the users, including their waiting time, actual use time, leaving time, and the facility that was used by each user.

In addition to the required inputs, the function also accepts inputs for a maximum-waiting time for each user; if the user waits up to this time then the user will leave without service. The user can also impose closure times on new arrivals, new services, or termination of services.

Note: Service facilities are assumed to be allocated to users on a "first-come, first-served" basis; in the event that more than one service facility is available for a user then the user is allocated to facilities first-to-last based on the facility number (i.e., the allocation favours the earlier facilities and it is not exchangeable with respect to the facility number).

Value

If all inputs are correctly specified then the function will return a list of class `queue` containing queuing information for the users and service facilities

Examples

```
q <- queue(2, 4:6, 7:9)
summary(q)
plot(q)
plot(summary(q))
```

`rm.attr` *Remove (non-protected) attributes from an object*

Description

`rm.attr` removes (non-protected) attributes from an object

Usage

```
rm.attr(
  object,
  list.levels = Inf,
  protected = c("class", "dim", "names", "dimnames", "rownames", "colnames")
)
```

Arguments

object	An object to operate on attributes from the object
list.levels	A non-negative integer specifying the number of levels of lists to apply the removal to
protected	A character vector containing the names of protected attributes (not to be removed)

Details

This function removes non-protected attributes from an R object. If the object is a list then the function will remove attributes within elements of the list down to the level specified by the list.levels input. (By default the function removes attributes from all levels of lists.) If you do not want to remove attributes from elements of a list (but still remove attributes from the outer level) you can set list.levels = 0 to do this..

Value

The object is returned with non-protected attributes removed

Examples

```
a <- structure(list(structure(1, x=2, names=3),
               list(0, structure(3, x=4, names=5))),
               x=3, names = 4)
str(rm.attr(a, 1))
```

---

sample.all *All Sampling Variations/Permutations*

Description

sample.all returns a matrix of all sampling variations/permutations from a set of integers

Usage

```
sample.all(n, size = n, replace = FALSE, prob = NULL)
```

Arguments

n	Number of integers to sample from
size	Length of the sample vectors
replace	Logical value; if FALSE the sampling is without replacement; if TRUE the sampling is with replacement
prob	Probability vector giving the sampling probability for each element (must be a probability vector with length n)

Details

This function computes all sample vectors of size `size` composed of the elements `1,...,n`, either with or without replacement of elements. If `size = n` and `replace = TRUE` then the list of all sample vectors corresponds to a list of all permutations of the integers `1,...,n`.

Value

A matrix of all permutations of the elements `1,...,n` (rows of the matrix give the permutations)

Examples

```
sample.all(n = 4, replace = FALSE)
```

---

sample.decomp *Sample decomposition*

Description

`sample.decomp` returns the data-frame of sample statistics for sample groups and their pooled sample

Usage

```
sample.decomp(
  moments = NULL,
  n = NULL,
  sample.mean = NULL,
  sample.sd = NULL,
  sample.var = NULL,
  sample.skew = NULL,
  sample.kurt = NULL,
  names = NULL,
  pooled = NULL,
  skew.type = NULL,
  kurt.type = NULL,
  kurt.excess = NULL,
  include.sd = FALSE
)
```

Arguments

`moments`	A data-frame of moments (an object of class 'moments')
`n`	A vector of sample sizes
`sample.mean`	A vector of sample means
`sample.sd`	A vector of sample standard deviations
`sample.var`	A vector of sample variances

sample.skew	A vector of sample skewness
sample.kurt	A vector of sample kurotsis
names	A vector of names for the sample groups
pooled	The number of the pooled group (if the pooled group is already present)
skew.type	The type of skewness statistic used ('Moment', 'Fisher Pearson' or 'Adjusted Fisher Pearson')
kurt.type	The type of kurtosis statistic used ('Moment', 'Fisher Pearson' or 'Adjusted Fisher Pearson')
kurt.excess	Logical value; if TRUE the sample kurtosis is the excess kurtosis (instead of the raw kurtosis)
include.sd	Logical value; if TRUE the output includes a column for the sample standard deviation (if needed)

Details

It is often useful to take a set of sample groups with known sample statistics and aggregate these into a single pooled sample and find the sample statistics of the pooled sample. Likewise, it is sometimes useful to take a set of sample groups and a pooled group with known sample statistics and determine the statistics of the other group required to complete the pooled sample. Both of these tasks can be accomplished using decomposition formulae for the sample size, sample mean and sample variance (or sample standard deviation). This function implements either of these two decomposition methods to find the sample statistics of the pooled sample or the other group remaining to obtain the pooled sample. The user inputs vectors for the sample size, sample mean and sample variance (or sample standard deviation). By default the groups are taken to be separate groups and the function computes the sample statistics for the pooled sample However, the user can input the number pooled sample as the input pooled; in this case that group is treated as the pooled sample and the function computes the other sample group required to obtain this pooled sample. The function returns a data-frame showing the sample statistics for all the groups including the pooled sample.

Value

A data-frame of all groups showing their sample sizes and sample moments

See Also

moments

---

skewness *Sample Skewness*

Description

skewness returns the sample skewness of a data vector/matrix

Usage

```
skewness(x, skew.type = NULL, na.rm = FALSE)
```

Arguments

x	A data vector/matrix
skew.type	The type of skewness statistic used ('Moment', 'Fisher Pearson' or 'Adjusted Fisher Pearson')
na.rm	Logical value; if TRUE the function removes NA values

Details

This function computes the sample skewness for a data vector or matrix. For a vector input the function returns a single value for the sample skewness of the data. For a matrix input the function treats each column as a data vector and returns a vector of values for the sample skewness of each of these datasets. The function can compute different types of skewness statistics using the skew.type input.

Value

The sample skewness of the data vector/matrix

Examples

```
skewness(rnorm(1000))
skewness(rexp(1000))
```

---

softmax *Softmax and inverse-softmax functions*

Description

softmax returns the value of the softmax function. softmaxinv returns the value of the inverse-softmax function.

Usage

```
softmax(eta, lambda = 1, gradient = FALSE, hessian = FALSE)

softmaxinv(p, lambda = 1, gradient = FALSE, hessian = FALSE)
```

Arguments

eta	A numeric vector input
lambda	Tuning parameter (a single positive value)
gradient	Logical; if TRUE the output will include a 'gradient' attribute
hessian	Logical; if TRUE the output will include a 'hessian' attribute
p	A probability vector (i.e., numeric vector of non-negative values that sum to one)

Details

The softmax function is a bijective function that maps a real vector with length m-1 to a probability vector with length m with all non-zero probabilities. The softmax function is useful in a wide range of probability and statistical applications. The present functions define the softmax function and its inverse, both with a tuning parameter.

Value

Value of the softmax function

Examples

```
softmax(5:7)
softmaxinv(softmax(5:7))
```

Index