# Package 'doBy'

May 8, 2026

**Version** 4.7.1

**Title** Groupwise Statistics, LSmeans, Linear Estimates, Utilities

**Description** Utility package containing: Main categories: Working with grouped data: 'do' something to data when stratified 'by' some variables. General linear estimates. Data handling utilities. Functional programming, in particular restrict functions to a smaller domain. Miscellaneous functions for data handling. Model stability in connection with model selection. Miscellaneous other tools.

**Encoding** UTF-8

**VignetteBuilder** knitr

**LazyData** true

**LazyDataCompression** xz

**URL** https://github.com/hojsgaard/doBy

**License** GPL (>= 2)

**Depends** R (>= 4.2.0), methods

**Imports** boot, broom, cowplot, Deriv, dplyr, forecast, ggplot2, MASS, Matrix, modelr, microbenchmark, rlang, purrr, tibble, tidyr,

**Suggests** geepack, knitr, lme4, markdown, rmarkdown, multcomp, pbkrtest (>= 0.5.2), survival, testthat (>= 2.1.0)

**RoxygenNote** 7.3.3

**NeedsCompilation** no

**Author** Ulrich Halekoh [aut, cph], Søren Højsgaard [aut, cre, cph]

**Maintainer** Søren Højsgaard <sorenh@math.aau.dk>

**Repository** CRAN

**Date/Publication** 2025-12-02 06:10:39 UTC

## Contents

---

`.rhsf2list` *Convert right hand sided formula to a list*

---

## Description

Convert right hand sided formula to a list

### Usage

```
.rhsf2list(f)
```

### Arguments

| | |
|---|---|
| f | A right hand sided formula |

---

## add_int *Add interaction columns to data frame*

---

### Description

Add interaction columns to data frame

### Usage

```
add_int(.data, .formula)
```

### Arguments

| | |
|---|---|
| .data | dataframe |
| .formula | right hand sided formula |

### Value

dataframe

### Author(s)

Søren Højsgaard

---

## add_pred *Add predicted values of different types to dataframe*

---

### Description

Add predicted values of different types to dataframe

### Usage

```
add_pred(data, model, var = "pred", type = NULL, transformation = NULL)
```

### Arguments

| | |
|---|---|
| data | dataframe or tibble |
| model | model object |
| var | name of new variable in dataframe / tibble |
| type | type of predicted value |
| transformation | A possible transformation of predicted variable, e.g. reciprocal(), log() etc |

### Value

dataframe / tibble

### Author(s)

Søren Højsgaard

### Examples

```
data(cars)
lm1 <- lm(dist ~ speed + I(speed^2), data=cars)
lm1 |> response() |> head()
cars <- cars |> add_pred(lm1)
cars |> head()
cars <- cars |> add_resid(lm1)
cars
```

---

## add_resid *Add residuals of different types to dataframe*

---

### Description

Add residuals of different types to dataframe

### Usage

```
add_resid(data, model, var = "resid", type)
```

### Arguments

| | |
|---|---|
| data | dataframe or tibble |
| model | model object |
| var | name of new variable in dataframe / tibble |
| type | type of residual value |

### Value

dataframe / tibble

### Author(s)

Søren Højsgaard

### Examples

```
data(cars)
lm1 <- lm(dist ~ speed + I(speed^2), data=cars)
lm1 |> response() |> head()
cars <- cars |> add_pred(lm1)
cars |> head()
cars <- cars |> add_resid(lm1)
cars
```

---

## align_coefs *Align and combine fixed-effect coefficients from multiple models*

---

### Description

Extracts and aligns the fixed-effect estimates from a list of fitted model objects, returning them in a single tidy data frame with consistent columns for easy comparison. Works with a mix of model types such as `lm`, `glm`, `gls`, `lmer`, etc.

For models without p-values (e.g., `lmer`), the function computes approximate Wald statistics and two-sided normal p-values.

### Usage

```
align_coefs(models)
```

### Arguments

| | |
|---|---|
| `models` | A named list of fitted model objects. Each element should be a model that can be passed to `broom::tidy()`. |

### Value

A tibble with columns:

**model** The name of the model (from the list).

**term** The term name (coefficient).

**estimate** The estimated coefficient.

**std.error** The standard error.

**statistic** The Wald statistic (estimate / std.error).

**p.value** Two-sided normal p-value.

## Examples

```
# Example using the built-in CO2 dataset
data(CO2)

# Fit models
lm_fit  <- lm(uptake ~ conc + Type + Treatment, data = CO2)
glm_fit <- glm(uptake ~ conc + Type + Treatment, family = Gamma(identity), data = CO2)

# Combine estimates
models_list <- list(lm = lm_fit, glm = glm_fit)
result <- align_coefs(models_list)
print(result)
```

---

`beets` *beets data*

---

## Description

Yield and sugar percentage in sugar beets from a split plot experiment. Data is obtained from a split plot experiment. There are 3 blocks and in each of these the harvest time defines the "whole plot" and the sowing time defines the "split plot". Each plot was 25 square meters and the yield is recorded in kg. See 'details' for the experimental layout.

## Usage

```
beets
```

## Format

The format is: chr "beets"

## Details

```
Experimental plan
Sowing times            1        4. april
                        2       12. april
                        3       21. april
                        4       29. april
                        5       18. may
Harvest times           1        2. october
                        2       21. october
Plot allocation:
              Block 1     Block 2     Block 3
            +-----------|-----------|-----------+
      Plot  | 1 1 1 1 1 | 2 2 2 2 2 | 1 1 1 1 1 | Harvest time
       1-15 | 3 4 5 2 1 | 3 2 4 5 1 | 5 2 3 4 1 | Sowing time
            |-----------|-----------|-----------|
```

```
Plot  | 2 2 2 2 2 | 1 1 1 1 1 | 2 2 2 2 2 | Harvest time
16-30 | 2 1 5 4 3 | 4 1 3 2 5 | 1 4 3 2 5 | Sowing time
      +-----------|-----------|-----------+
```

## References

Ulrich Halekoh, Søren Højsgaard (2014)., A Kenward-Roger Approximation and Parametric Bootstrap Methods for Tests in Linear Mixed Models - The R Package pbkrtest., Journal of Statistical Software, 58(10), 1-30., https://www.jstatsoft.org/v59/i09/

## Examples

```
data(beets)

beets$bh <- with(beets, interaction(block, harvest))
summary(aov(yield ~ block + sow + harvest + Error(bh), beets))
summary(aov(sugpct ~ block + sow + harvest + Error(bh), beets))
```

---

`binomial_to_bernoulli_data` *Convert binomial data to bernoulli data*

---

## Description

Convert binomial data to bernoulli data by expanding dataset.

## Usage

```
binomial_to_bernoulli_data(
  data.,
  y,
  size,
  type = c("rest", "total"),
  response_name = "response",
  rest_name = NULL
)
```

## Arguments

| | |
|---|---|
| `data.` | A dataframe |
| `y` | Column with 'successes' in binomial distribution `y~bin(size, p)` |
| `size` | Column with 'failures', i.e. size-y or 'total', i.e. size. |
| `type` | Whether `size` is rest (i.e. 'failures') or 'total' |
| `response_name` | Name of response variable in output dataset. |
| `rest_name` | Name of 'failures' in column `response_name`. |

## Examples

```
dat <- budworm
dat <- dat[dat$dose %in% c(1,2), ]
dat$ntotal <- 5
dat
dat.a <- dat |>
  binomial_to_bernoulli_data(ndead, ntotal, type="total")
dat.b <- dat |>
  dplyr::mutate(nalive=ntotal-ndead) |> dplyr::select(-ntotal) |>
  binomial_to_bernoulli_data(ndead, nalive, type="rest")

m0 <- glm(cbind(ndead, ntotal-ndead) ~ dose + sex, data=dat, family=binomial())
m1 <- glm(ndead / ntotal ~ dose + sex, data=dat, weight=ntotal, family=binomial())
ma <- glm(response ~ dose + sex, data=dat.a, family=binomial())
mb <- glm(response ~ dose + sex, data=dat.b, family=binomial())

dat.a$response
dat.b$response ## Not same and therefore the following do not match

all.equal(coef(m0), coef(ma))
all.equal(coef(m0), coef(mb))
all.equal(coef(m1), coef(ma))
all.equal(coef(m1), coef(mb))
```

---

# bquote_fun_list *Backquote a list of functions*

---

## Description

Backquote a list of functions

## Usage

```
bquote_fun_list(fun_list)
```

## Arguments

| | |
|---|---|
| fun_list | List of functions |

## See Also

base::bquote(), set_default(), section_fun()

## Examples

```
## Evaluate a list of functions
f1 <- function(x){x + 1}
f2 <- function(x){x + 8}
```

```
f1_ <- set_default(f1, list(x=10))
f2_ <- set_default(f2, list(x=10))

f1_(); f2_()

fn_list  <- list(f1_, f2_)
fn_list_ <- bquote_fun_list(fn_list)

eval(fn_list[[1]])     ## No
sapply(fn_list, eval)  ## No

eval(fn_list_[[1]])    ## Yes
sapply(fn_list_, eval) ## Yes
```

---

by-lapply *Formula based version of lapply and sapply*

---

## Description

This function is a wrapper for calling lapply on the list resulting from first calling splitBy.

## Usage

```
lapply_by(data, formula, FUN, ...)

lapplyBy(formula, data = parent.frame(), FUN, ...)

sapply_by(data, formula, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)

sapplyBy(
  formula,
  data = parent.frame(),
  FUN,
  ...,
  simplify = TRUE,
  USE.NAMES = TRUE
)
```

## Arguments

| | |
|---|---|
| data | A dataframe. |
| formula | A formula describing how data should be split. |
| FUN | A function to be applied to each element in the split list, see 'Examples' below. |
| ... | optional arguments to FUN. |
| simplify | Same as for sapply |
| USE.NAMES | Same as for sapply |

## Value

A list.

## Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

## See Also

splitBy, split_by

## Examples

```
fun <- function(x) range(x$uptake)
lapplyBy(~Treatment + Type, data=CO2, FUN=fun)
sapplyBy(~Treatment + Type, data=CO2, FUN=fun)

# Same as
lapply(splitBy(~Treatment + Type, data=CO2), FUN=fun)
```

---

# by-lmby *List of lm objects with a common model*

---

## Description

The data is split into strata according to the levels of the grouping factors and individual lm fits are obtained for each stratum.

## Usage

```
lm_by(data., formula., id = NULL, ...)

lmBy(formula., data., id = NULL, ...)
```

## Arguments

| | |
|---|---|
| data. | A dataframe |
| formula. | A linear model formula object of the form y ~ x1 + ... + xn \| g1 + ... + gm. In the formula object, y represents the response, x1, ..., xn the covariates, and the grouping factors specifying the partitioning of the data according to which different lm fits should be performed. |
| id | A formula describing variables from data which are to be available also in the output. |
| ... | Additional arguments passed on to lm(). |

## Value

A list of lm fits.

### Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

### Examples

```
lm_lst <- lmBy(1 / uptake ~ log(conc) | Treatment, data=CO2)
coef(lm_lst)

fitted(lm_lst)
residuals(lm_lst)

summary(lm_lst)
coef(summary(lm_lst))
coef(summary(lm_lst), simplify=TRUE)
```

---

## by-order *Ordering (sorting) rows of a data frame*

---

### Description

Ordering (sorting) rows of a data frame by the certain variables in the data frame. This function is essentially a wrapper for the `order()` function - the important difference being that variables to order by can be given by a model formula.

### Usage

```
order_by(data, formula)

orderBy(formula, data)
```

### Arguments

| | |
|---|---|
| `data` | A dataframe |
| `formula` | The right hand side of a formula |

### Details

The sign of the terms in the formula determines whether sorting should be ascending or decreasing; see examples below

### Value

The ordered data frame

### Author(s)

Søren Højsgaard, <sorenh@math.aau.dk> and Kevin Wright

## See Also

transformBy, transform_by, splitBy, split_by

## Examples

```
orderBy(~ conc + Treatment, CO2)
## Sort decreasingly by conc
orderBy(~ - conc + Treatment, CO2)
## Same as:
order_by(CO2, c("conc", "Treatment"))
order_by(CO2, c("-conc", "Treatment"))
```

---

# by-sample *Sampling from a data frame*

---

## Description

A data frame is split according to some variables in a formula, and a sample of a certain fraction of each is drawn.

## Usage

```
sample_by(data, formula, frac = 0.1, replace = FALSE, systematic = FALSE)

sampleBy(
  formula,
  frac = 0.1,
  replace = FALSE,
  data = parent.frame(),
  systematic = FALSE
)
```

## Arguments

| | |
|---|---|
| data | A data frame. |
| formula | A formula defining the grouping of the data frame. |
| frac | The part of data to be sampled. |
| replace | Is the sampling with replacement. |
| systematic | Should sampling be systematic. |

## Details

If systematic=FALSE (default) then frac gives the fraction of data sampled. If systematic=TRUE and frac=.2 then every 1/.2 i.e. every 5th observation is taken out.

## Value

A dataframe.

## See Also

orderBy, order_by, splitBy, split_by, summaryBy, summary_by, transformBy, transform_by

## Examples

```
data(dietox)
sampleBy(formula = ~ Evit + Cu, frac=.1, data = dietox)
```

---

# by-split *Split a data frame into groups defined by variable(s)*

---

## Description

Split a dataframe according to the levels of variables in the dataframe. Uses vparse() to interpret flexible input.

## Usage

```
split_by(data., ..., omit = TRUE)

splitBy(formula, data, omit = TRUE)

## S3 method for class 'splitByData'
head(x, n = 6L, ...)

## S3 method for class 'splitByData'
tail(x, n = 6L, ...)

split_by.legacy(data, formula, drop = TRUE)

splitBy.legacy(formula, data = parent.frame(), drop = TRUE)
```

## Arguments

| | |
|---|---|
| data. | A data frame (or tibble) to split |
| ... | Variables defining the groups |
| omit | If TRUE (default), group-defining variables are omitted in each split group |
| formula | A right hand sided formula (for the old interface) |
| data | A data frame (for the old interface) |
| x | A splitByData object. |
| n | An integer vector. |
| drop | Obsolete |

## Value

An object of class \"splitByData\" (a named list with group attributes)

## Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

## See Also

orderBy, order_by, summaryBy, summary_by, transformBy, transform_by

## Examples

```
split_by(CO2, ~Treatment+Type)
split_by(CO2, Treatment, Type)
split_by(CO2, c("Treatment", "Type"))
split_by(CO2, "Treatment", "Type")

x <- split_by(CO2, "Treatment", "Type")
head(x, 3)
tail(x, 3)

## Via wrapper:
foo2 <- function(x) {
 x <- rlang::enquo(x)
 split_by(CO2, !!x)
}
foo2(~Treatment)

## The "Old" interface
splitBy(~Treatment + Type, CO2)
splitBy(~Treatment + Type, data=CO2)
splitBy(c("Treatment", "Type"), data=CO2)
```

---

# by-subset *Finds subsets of a dataframe which is split by variables in a formula.*

---

## Description

A data frame is split by a formula into groups. Then subsets are found within each group, and the result is collected into a data frame.

## Usage

```
subset_by(data, formula, subset, select, drop = FALSE, join = TRUE, ...)

subsetBy(
  formula,
```

```
  subset,
  data = parent.frame(),
  select,
  drop = FALSE,
  join = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| data | A data frame. |
| formula | A right hand sided formula or a character vector of variables to split by. |
| subset | logical expression indicating elements or rows to keep: missing values are taken as false. |
| select | expression, indicating columns to select from a data frame. |
| drop | passed on to [ indexing operator. |
| join | If FALSE the result is a list of data frames (as defined by 'formula'); if TRUE one data frame is returned. |
| ... | further arguments to be passed to or from other methods. |

## Value

A data frame.

## Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

## See Also

splitBy, split_by

## Examples

```
data(dietox)
subsetBy(~Evit, Weight < mean(Weight), data=dietox)
```

---

# by-summary *Groupwise summary statistics*

---

## Description

Computes summary statistics by groups, similar to the summary procedure in SAS. A more flexible alternative to base R's aggregate.

## Usage

```
summary_by(
  data,
  formula,
  id = NULL,
  FUN = mean,
  keep.names = FALSE,
  p2d = FALSE,
  order = TRUE,
  full.dimension = FALSE,
  var.names = NULL,
  fun.names = NULL,
  ...
)

summaryBy(
  formula,
  data = parent.frame(),
  id = NULL,
  FUN = mean,
  keep.names = FALSE,
  p2d = FALSE,
  order = TRUE,
  full.dimension = FALSE,
  var.names = NULL,
  fun.names = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| data | A data frame. |
| formula | A formula specifying response and grouping variables. |
| id | A formula indicating variables to retain (not grouped by). |
| FUN | A function or list of functions to apply to the response variables. |
| keep.names | Logical; keep original variable names if only one function is applied. |
| p2d | Replace parentheses in output names with dots? |
| order | Logical; should result be ordered by grouping variables? |
| full.dimension | Logical; if TRUE, repeat rows so output matches input size. |
| var.names | Optional custom names for response variables. |
| fun.names | Optional custom names for functions applied. |
| ... | Additional arguments passed to functions in FUN. |

## Details

Extra arguments in ... are passed to all functions in FUN. If needed, wrap functions to handle these consistently (e.g., for na.rm = TRUE).

## Value

A data frame of grouped summary statistics.

## Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

## See Also

aggregate, orderBy, transformBy, splitBy

## Examples

```
data(CO2)

# Simple groupwise mean
summaryBy(uptake ~ Type + Treatment, data = CO2, FUN = mean)
summaryBy(cbind(uptake, conc) ~ Type + Treatment, data = CO2, FUN = mean)

# Compare with
aggregate(cbind(uptake, conc) ~ Type + Treatment, data = CO2, FUN = mean)

## Using '.' on the right hand side of a formula means to stratify by
## all variables not used elsewhere:
summaryBy(uptake ~ ., data = CO2, FUN = mean)

# Multiple functions using a custom summary function
myfun <- function(x, ...)
  c(m = mean(x, na.rm = TRUE), v = var(x, na.rm = TRUE), n = length(x))
summaryBy(uptake ~ Type + Treatment, data = CO2, FUN = myfun)

# Summary on transformed variables
# works:
summaryBy(cbind(lu=log(uptake), conc) ~ Type, data = CO2, FUN = mean)
# fails:
#summaryBy(cbind(log(uptake), conc) ~ Type, data = CO2, FUN = mean)
```

---

# by-transform *Function to make groupwise transformations*

---

## Description

Function to make groupwise transformations of data by applying the transform function to subsets of data.

## Usage

```
transform_by(data, formula, ...)

transformBy(formula, data, ...)
```

### Arguments

| | |
|---|---|
| data | A data frame |
| formula | A formula with only a right hand side, see examples below |
| ... | Further arguments of the form tag=value |

### Details

The ... arguments are tagged vector expressions, which are evaluated in the data frame data. The tags are matched against names(data), and for those that match, the value replace the corresponding variable in data, and the others are appended to data.

### Value

The modified value of the dataframe data.

### Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

### See Also

orderBy, order_by, summaryBy, summary_by, splitBy, split_by

### Examples

```
data(dietox)
transformBy(~Pig, data=dietox, minW=min(Weight), maxW=max(Weight),
    gain=diff(range(Weight)))
```

---

## carcass *Lean meat contents of 344 pig carcasses*

---

### Description

Measurement of lean meat percentage of 344 pig carcasses together with auxiliary information collected at three Danish slaughter houses

### Usage

```
carcass
```

## Format

carcassall: A data frame with 344 observations on the following 17 variables.

`weight` Weight of carcass

`lengthc` Length of carcass from back toe to head (when the carcass hangs in the back legs)

`lengthf` Length of carcass from back toe to front leg (that is, to the shoulder)

`lengthp` Length of carcass from back toe to the pelvic bone

`Fat02`, `Fat03`, `Fat11`, `Fat12`, `Fat13`, `Fat14`, `Fat16` Thickness of fat layer at different locations on the back of the carcass (FatXX refers to thickness at (or rather next to) rib no. XX. Notice that 02 is closest to the head

`Meat11`, `Meat12`, `Meat13` Thickness of meat layer at different locations on the back of the carcass, see description above

`LeanMeat` Lean meat percentage determined by dissection

`slhouse` Slaughter house; a factor with levels `slh1` and `slh2`.

`sex` Sex of the pig; a factor with levels `castrate` and `female`.

`size` Size of the carcass; a factor with levels `normal` and `large`. Here, `normal` refers to carcass weight under 80 kg; `large` refers to carcass weights between 80 and 110 kg.

## Details

: Notice that there were slaughtered large pigs only at one slaughter house.

## Note

carcass: Contains only the variables Fat11, Fat12, Fat13, Meat11, Meat12, Meat13, LeanMeat

## Source

Busk, H., Olsen, E. V., Brøndum, J. (1999) Determination of lean meat in pig carcasses with the Autofom classification system, Meat Science, 52, 307-314

## Examples

```
data(carcass)
head(carcass)
```

child_growth *Berkeley Growth Study data*

## Description

dataframe with heights of 39 boys and 54 girls from age 1 to 18 and the ages at which they were collected.

## Usage

```
child_growth
```

## Format

**gender** Gender of child

**age** Age at time of data recording

**subject** Identification for each child

**height** Height of child

## Details

Notice that the ages are not equally spaced. Data are taken from the fda package (growth) but put in long format here.

## References

Ramsay, James O., Hooker, Giles, and Graves, Spencer (2009), _Functional data analysis with R and Matlab_, Springer, New York.

Ramsay, James O., and Silverman, Bernard W. (2005), _Functional Data Analysis, 2nd ed._, Springer, New York.

Ramsay, James O., and Silverman, Bernard W. (2002), _Applied Functional Data Analysis_, Springer, New York.

Tuddenham, R. D., and Snyder, M. M. (1954) "Physical growth of California boys and girls from birth to age 18", _University of California Publications in Child Development_, 1, 183-364.

codstom *Diet of Atlantic cod in the Gulf of St. Lawrence (Canada)*

## Description

Stomach content data for Atlantic cod (*Gadus morhua*) in the Gulf of St.Lawrence, Eastern Canada. Note: many prey items were of no interest for this analysis and were regrouped into the "Other" category.

## Usage

```
codstom
```

## Format

A data frame with 10000 observations on the following 10 variables.

`region` a factor with levels `SGSL` `NGSL` representing the southern and northern Gulf of St. Lawrence, respectively

`ship.type` a factor with levels `2` `3` `31` `34` `90` `99`

`ship.id` a factor with levels `11558` `11712` `136148` `136885` `136902` `137325` `151225` `151935` `99433`

`trip` a factor with levels `10` `11` `12` `179` `1999` `2` `2001` `20020808` `3` `4` `5` `6` `7` `8` `88` `9` `95`

`set` a numeric vector

`fish.id` a numeric vector

`fish.length` a numeric vector, length in mm

`prey.mass` a numeric vector, mass of item in stomach, in g

`prey.type` a factor with levels `Ammodytes_sp` `Argis_dent` `Chion_opil` `Detritus` `Empty` `Eualus_fab` `Eualus_mac` `Gadus_mor` `Hyas_aran` `Hyas_coar` `Lebbeus_gro` `Lebbeus_pol` `Leptocl_mac` `Mallot_vil` `Megan_norv` `Ophiuroidea` `Other` `Paguridae` `Pandal_bor` `Pandal_mon` `Pasiph_mult` `Sabin_sept` `Sebastes_sp` `Them_abys` `Them_comp` `Them_lib`

## Details

Cod are collected either by contracted commerical fishing vessels (`ship.type` 90 or 99) or by research vessels. Commercial vessels are identified by a unique `ship.id`.

Either one research vessel or several commercial vessels conduct a survey (`trip`), during which a trawl, gillnets or hooked lines are set several times. Most trips are random stratified surveys (depth-based stratification).

Each trip takes place within one of the `region`s. The `trip` label is only guaranteed to be unique within a region and the `set` label is only guaranteed to be unique within a `trip`.

For each fish caught, the `fish.length` is recorded and the fish is allocated a `fish.id`, but the `fish.id` is only guaranteed to be unique within a `set`. A subset of the fish caught are selected for stomach analysis (stratified random selection according to fish length; unit of stratification is the

set for research surveys, the combination ship.id and stratum for surveys conducted by commercial vessels, although strata are not shown in codstom).

The basic experimental unit in this data set is a cod stomach (one stomach per fish). Each stomach is uniquely identified by a combination of `region`, `ship.type`, `ship.id`, `trip`, `set`, and `fish.id`.

For each prey item found in a stomach, the species and mass of the prey item are recorded, so there can be multiple observations per stomach. There may also be several prey items with the same `prey.type` in the one stomach (for example many `prey.types` have been recoded `Other`, which produced many instances of `Other` in the same stomach).

If a stomach is empty, a single observation is recorded with `prey.type` `Empty` and a `prey.mass` of zero.

## Source

Small subset from a larger dataset (more stomachs, more variables, more `prey.types`) collected by D. Chabot and M. Hanson, Fisheries & Oceans Canada <chabotd@dfo-mpo.gc.ca>.

## Examples

```
data(codstom)
str(codstom)
# removes multiple occurences of same prey.type in stomachs
codstom1 <- summaryBy(prey.mass ~
                      region + ship.type + ship.id + trip + set + fish.id + prey.type,
                      data = codstom,
                      FUN = sum)

# keeps a single line per stomach with the total mass of stomach content
codstom2 <- summaryBy(prey.mass ~ region + ship.type + ship.id + trip + set + fish.id,
                      data = codstom,
                      FUN = sum)

# mean prey mass per stomach for each trip
codstom3 <- summaryBy(prey.mass.sum ~ region + ship.type + ship.id + trip,
                      data = codstom2, FUN = mean)

## Not run:
# wide version, one line per stomach, one column per prey type
library(reshape)
codstom4 <- melt(codstom, id = c(1:7, 9))
codstom5 <- cast(codstom4,
                 region + ship.type + ship.id + trip + set + fish.id + fish.length ~
                 prey.type, sum)
k <- length(names(codstom5))
prey_col <- 8:k
out <- codstom5[,prey_col]
out[is.na(out)] <- 0
codstom5[,prey_col] <- out
codstom5$total.content <- rowSums(codstom5[, prey_col])

## End(Not run)
```

## crickets *crickets data*

### Description

Mating songs of male tree crickets.

### Usage

```
crickets
```

### Format

This data frame contains:

**species:** Species, (exis, nius), see details

**temp:** temperature

**pps:** pulse per second

### Details

Walker (1962) studied the mating songs of male tree crickets. Each wingstroke by a cricket produces a pulse of song, and females may use the number of pulses per second to identify males of the correct species. Walker (1962) wanted to know whether the chirps of the crickets Oecanthus exclamationis (abbreviated exis) and Oecanthus niveus (abbreviated nius) had different pulse rates. See the biostathandbook for details. (The abbreviations are made from the the first two and last two letters of the species.) Walker measured the pulse rate of the crickets (variable `pps`) at a variety of temperatures (`temp`):

### Examples

```
data(crickets)
coplot(pps ~ temp | species, data=crickets)
```

## crimeRate *crimeRate*

### Description

Crime rates per 100,000 inhabitants in states of the USA for different crime types in 1977.

### Usage

```
crimeRate
```

## Format

This data frame contains:

**state:** State of the USA
**murder:** crime of murder
**rape:**
**robbery:**
**assault:**
**burglary:** residential theft
**larceny:** unlawful taking of personal property (pocket picking)
**autotheft:**

## Examples

```
data(crimeRate)
```

---

`crime_rate` *crimeRate*

---

## Description

Crime rates per 100,000 inhabitants in states of the USA for different crime types in 1977.

## Usage

```
crime_rate
```

## Format

This data frame contains:

**murder:** crime of murder
**rape:**
**robbery:**
**assault:**
**burglary:** residential theft
**larceny:** unlawful taking of personal property (pocket picking)
**autotheft:**

## Examples

```
data(crime_rate)
```

## cropyield *Yield from Danish agricultural production of grain and root crop.*

### Description

Yield from Danish agricultural production of grain and root crop.

### Usage

```
cropyield
```

### Format

A dataframe with 97 rows and 7 columns.

`year` From 1901 to 1997.
`precip` Milimeter precipitation.
`yield` Million feed units (see details).
`area` Area in 1000 ha for grains and root crop.
`fertil` 1000 tons fertilizer.
`avgtmp1` Average temperature April-June (3 months).
`avgtmp2` Average temperature July-Octobre (4 months).

### Details

A feed unit is the amount of energy in a kg of barley.

### References

Danmarks statistik (Statistics Denmark).

## cv_glm_fitlist *Cross-validation for list of glm objects*

### Description

Cross-validation for list of glm objects

### Usage

```
cv_glm_fitlist(data., fit_list, K = 10)
```

### Arguments

| | |
|---|---|
| `data.` | A data frame |
| `fit_list` | A list of glm objects |
| `K` | Number of folds |

`data-wine` *Chemical composition of wine*

## Description

Using chemical analysis determine the origin of wines

## Usage

```
data(wine)
```

## Format

A data frame with 178 observations on the following 14 variables.

`Cult` a factor with levels `v1` `v2` `v3`: 3 different graph varieties

`Alch` Alcohol

`Mlca` Malic acid

`Ash` Ash

`Aloa` Alcalinity of ash

`Mgns` Magnesium

`Ttlp` Total phenols

`Flvn` Flavanoids

`Nnfp` Nonflavanoid phenols

`Prnt` Proanthocyanins

`Clri` Color intensity

`Hue` Hue

`Oodw` OD280/OD315 of diluted wines

`Prln` Proline

## Details

Data comes from the UCI Machine Learning Repository. The grape variety `Cult` is the class identifier.

## Source

Frank, A. & Asuncion, A. (2010). UCI Machine Learning Repository https://archive.ics.uci.edu/ml/. Irvine, CA: University of California, School of Information and Computer Science.

## References

See references at https://archive.ics.uci.edu/ml/datasets/Wine/

## Examples

```
data(wine)
## maybe str(wine) ; plot(wine) ...
```

---

`data_breastcancer` *Gene expression signatures for p53 mutation status in 250 breast cancer samples*

---

## Description

Perturbations of the p53 pathway are associated with more aggressive and therapeutically refractory tumours. We preprocessed the data using Robust Multichip Analysis (RMA). Dataset has been truncated to the 1000 most informative genes (as selected by Wilcoxon test statistics) to simplify computation. The genes have been standardized to have zero mean and unit variance (i.e. z-scored).

## Usage

```
breastcancer
```

## Format

A data frame with 250 observations on 1001 variables. The first 1000 columns are numerical variables; the last column (named `code`) is a factor with levels `case` and `control`.

## Details

The factor `code` defines whether there was a mutation in the p53 sequence (code=case) or not (code=control).

## Source

Chris Holmes, `<c.holmes@stats.ox.ac.uk>`

## References

Miller et al (2005, PubMed ID:16141321)

## Examples

```
data(breastcancer)
bc <- breastcancer
pairs(bc[,1:5], col=bc$code)

train <- sample(1:nrow(bc), 50)
table(bc$code[train])
## Not run:
library(MASS)
```

```
z <- lda(code ~ ., data=bc, prior = c(1, 1) / 2, subset = train)
pc <- predict(z, bc[-train, ])$class
pc
bc[-train, "code"]
table(pc, bc[-train, "code"])

## End(Not run)
```

---

`data_budworm` *Budworm data*

---

## Description

Experiment on the toxicity to the tobacco budworm Heliothis virescens of doses of the pyrethroid trans-cypermethrin to which the moths were beginning to show resistance. Batches of 20 moths of each sex were exposed for three days to the pyrethroid and the number in each batch that were dead or knocked down was recorded. Data is reported in Collett (1991, p. 75).

## Usage

```
budworm
```

## Format

This data frame contains 12 rows and 4 columns:

**sex:** sex of the budworm.

**dose:** dose of the insecticide trans-cypermethrin (in micro grams).

**ndead:** budworms killed in a trial.

**ntotal:** total number of budworms exposed per trial.

## Source

Collett, D. (1991) Modelling Binary Data, Chapman & Hall, London, Example 3.7

## References

Venables, W.N; Ripley, B.D.(1999) Modern Applied Statistics with S-Plus, Heidelberg, Springer, 3rd edition, chapter 7.2

## Examples

```
data(budworm)

## function to caclulate the empirical logits
empirical.logit<- function(nevent,ntotal) {
   y <- log((nevent + 0.5) / (ntotal - nevent + 0.5))
   y
}


# plot the empirical logits against log-dose

log.dose <- log(budworm$dose)
emp.logit <- empirical.logit(budworm$ndead, budworm$ntotal)
plot(log.dose, emp.logit, type='n', xlab='log-dose',ylab='emprirical logit')
title('budworm: emprirical logits of probability to die ')
male <- budworm$sex=='male'
female <- budworm$sex=='female'
lines(log.dose[male], emp.logit[male], type='b', lty=1, col=1)
lines(log.dose[female], emp.logit[female], type='b', lty=2, col=2)
legend(0.5, 2, legend=c('male', 'female'), lty=c(1,2), col=c(1,2))

## Not run:
* SAS example;
data budworm;
infile 'budworm.txt' firstobs=2;
input sex dose ndead ntotal;
run;

## End(Not run)
```

---

# data_cad *Coronary artery disease data*

---

## Description

A cross classified table with observational data from a Danish heart clinic. The response variable is CAD (coronary artery disease, some times called heart attack).

## Usage

```
data(cad1)
```

## Format

A data frame with 236 observations on the following 14 variables.

`Sex` Sex; a factor with levels `Female` `Male`

`AngPec` Angina pectoris (chest pain attacks); a factor with levels `Atypical` `None` `Typical`

`AMI` Acute myocardic infarct; a factor with levels `Definite` `NotCertain`

`QWave` A reading from an electrocardiogram; a factor with levels `No` `Yes`; Yes means pathological and is a sign of previous myocardial infarction.

`QWavecode` a factor with levels `Nonusable` `Usable`. An assesment of whether QWave is reliable.

`STcode` a factor with levels `Nonusable` `Usable`. An assesment of whether STchange is reliable.

`STchange` A reading from an electrocardiogram; a factor with levels `No` `Yes`. An STchange indicates a blockage of the coronary artery.

`SuffHeartF` Sufficient heart frequency; a factor with levels `No`, `Yes`

`Hypertrophi` a factor with levels `No`, `Yes`. Hypertrophy refers to an increased size of the heart muscle due to exercise.

`Hyperchol` a factor with levels `No` `Yes`. Hypercholesterolemia, also called high cholesterol, is the presence of high levels of cholesterol in the blood.

`Smoker` Is the patient a smoker; a factor with levels `No`, `Yes`.

`Inherit` Hereditary predispositions for CAD; a factor with levels `No`, `Yes`.

`Heartfail` Previous heart failures; a factor with levels `No` `Yes`

`CAD` Coronary Artery Disease; a factor with levels `No` `Yes`. CAD refers to a reduction of blood flow to the heart muscle (commonly known as a heart attack). The diagnosis made from biopsies.

## Details

Notice that data are collected at a heart clinic, so data do not represent the population, but are conditional on patients having ended up at the clinic.

- cad1: Complete dataset, 236 cases.
- cad2: Incomplete dataset, 67 cases. Information on (some of) the variables 'Hyperchol', 'Smoker' and 'Inherit' is missing.

## References

Hansen, J. F. (1980). The clinical diagnosis of ischaemic heart disease due to coronary artery disease. Danish Medical Bulletin

Højsgaard, Søren and Thiesson, Bo (1995). BIFROST - Block recursive models Induced From Relevant knowledge, Observations and Statistical Techniques. Computational Statistics and Data Analysis, vol. 19, p. 155-175 #'

## Examples

```
data(cad1)
## maybe str(cad1) ; plot(cad1) ...
```

## data_mathmark *Mathematics marks for students*

### Description

The mathmark data frame has 88 rows and 5 columns.

### Usage

```
data(mathmark)
```

### Format

This data frame contains the following columns: mechanics, vectors, algebra, analysis, statistics.

### Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

### References

David Edwards, An Introduction to Graphical Modelling, Second Edition, Springer Verlag, 2000

### Examples

```
data(mathmark)
```

## data_personality *Personality traits*

### Description

The peronality dataframe has 240 rows and 32 columns

### Usage

```
data(personality)
```

### Format

This dataframe has recordings on the following 32 variables: distant, talkatv, carelss, hardwrk, anxious, agreebl, tense, kind, opposng, relaxed, disorgn, outgoin, approvn, shy, discipl, harsh, persevr, friendl, worryin, respnsi, contrar, sociabl, lazy, coopera, quiet, organiz, criticl, lax, laidbck, withdrw, givinup, easygon

## Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

## References

Origin unclear

## Examples

```
data(personality)
str(personality)
```

---

descStat *Computing simple descriptive statistics of a numeric vector.*

---

## Description

Computing simple descriptive statistics of a numeric vector - not unlike what proc means of SAS does

## Usage

```
descStat(x, na.rm = TRUE)
```

## Arguments

| | |
|---|---|
| x | A numeric vector |
| na.rm | Should missing values be removed |

## Value

A vector with named elements.

## Author(s)

Gregor Gorjanc; <gregor.gorjanc@bf.uni-lj.si>

## See Also

summaryBy, summary_by

## Examples

```
x <- c(1, 2, 3, 4, NA, NaN)
descStat(x)
```

`dietox` *Growth curves of pigs in a 3x3 factorial experiment*

## Description

The `dietox` data frame has 861 rows and 7 columns.

## Usage

```
dietox
```

## Format

This data frame contains the following columns:

**Weight** Weight in Kg

**Feed** Cumulated feed intake in Kg

**Time** Time (in weeks) in the experiment

**Pig** Factor; id of each pig

**Evit** Factor; vitamin E dose; see 'details'.

**Cu** Factor, copper dose; see 'details'

**Start** Start weight in experiment, i.e. weight at week 1.

**Litter** Factor, id of litter of each pig

## Details

Data contains weight of slaughter pigs measured weekly for 12 weeks. Data also contains the start weight (i.e. the weight at week 1). The treatments are 3 different levels of Evit = vitamin E (dose: 0, 100, 200 mg dl-alpha-tocopheryl acetat /kg feed) in combination with 3 different levels of Cu=copper (dose: 0, 35, 175 mg/kg feed) in the feed. The cumulated feed intake is also recorded. The pigs are litter mates.

## Source

Lauridsen, C., Højsgaard, S.,Sørensen, M.T. C. (1999) Influence of Dietary Rapeseed Oli, Vitamin E, and Copper on Performance and Antioxidant and Oxidative Status of Pigs. J. Anim. Sci.77:906-916

## Examples

```
data(dietox)
head(dietox)
coplot(Weight ~ Time | Evit * Cu, data=dietox)
```

esticon *Contrasts for lm, glm, lme, and geeglm objects*

## Description

Computes linear functions (i.e. weighted sums) of the estimated regression parameters. Can also test the hypothesis, that such a function is equal to a specific value.

## Usage

```
esticon(obj, L, beta0, conf.int = TRUE, level = 0.95, joint.test = FALSE, ...)

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

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

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

## S3 method for class 'esticon_class'
vcov(object, ...)
```

## Arguments

| | |
|---|---|
| obj | Regression object (of type lm, glm, lme, geeglm). |
| L | Matrix (or vector) specifying linear functions of the regression parameters (one linear function per row). The number of columns must match the number of fitted regression parameters in the model. See 'details' below. |
| beta0 | A vector of numbers |
| conf.int | TRUE |
| level | The confidence level |
| joint.test | Logical value. If TRUE a 'joint' Wald test for the hypothesis L beta = beta0 is made. Default is that the 'row-wise' tests are made, i.e. (L beta)i=beta0i. If joint.test is TRUE, then no confidence interval etc. is calculated. |
| ... | Additional arguments; currently not used. |
| object | An esticon_class 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. |

## Details

Let the estimated parameters of the model be

$$\beta_1, \beta_2, \ldots, \beta_p$$

A linear function of the estimates is of the form

$$l = \lambda_1\beta_1 + \lambda_2\beta_2 + \cdots + \lambda_p\beta_p$$

where $\lambda_1, \lambda_2, \ldots, \lambda_p$ is specified by the user.

The esticon function calculates l, its standard error and by default also a 95 pct confidence interval. It is sometimes of interest to test the hypothesis $H_0 : l = \beta_0$ for some value $\beta_0$ given by the user. A test is provided for the hypothesis $H_0 : l = 0$ but other values of $\beta_0$ can be specified.

In general, one can specify r such linear functions at one time by specifying L to be an $r \times p$ matrix where each row consists of p numbers $\lambda_1, \lambda_2, \ldots, \lambda_p$. Default is then that $\beta_0$ is a p vector of 0s but other values can be given.

It is possible to test simultaneously that all specified linear functions are equal to the corresponding values in $\beta_0$.

For computing contrasts among levels of a single factor, 'contrast.lm' may be more convenient.

## Value

Returns a matrix with one row per linear function. Columns contain estimated coefficients, standard errors, t values, degrees of freedom, two-sided p-values, and the lower and upper endpoints of the 1-alpha confidence intervals.

## Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

## Examples

```
data(iris)
lm1  <- lm(Sepal.Length ~ Sepal.Width + Species + Sepal.Width : Species, data=iris)
## Note that the setosa parameters are set to zero
coef(lm1)

## Estimate the intercept for versicolor
lambda1 <- c(1, 0, 1, 0, 0, 0)
esticon(lm1, L=lambda1)

## Estimate the difference between versicolor and virgica intercept
## and test if the difference is 1
lambda2 <- c(0, 1, -1, 0, 0, 0)
esticon(lm1, L=lambda2, beta0=1)

## Do both estimates at one time
esticon(lm1, L=rbind(lambda1, lambda2), beta0=c(0, 1))
```

```
## Make a combined test for that the difference between versicolor and virgica intercept
## and difference between versicolor and virginica slope is zero:
lambda3 <- c(0, 0, 0, 0, 1, -1)
esticon(lm1, L=rbind(lambda2, lambda3), joint.test=TRUE)

# Example using esticon on coxph objects (thanks to Alessandro A. Leidi).
# Using dataset 'veteran' in the survival package
# from the Veterans' Administration Lung Cancer study

if (require(survival)){
data(veteran)
sapply(veteran, class)
levels(veteran$celltype)
attach(veteran)
veteran.s <- Surv(time, status)
coxmod <- coxph(veteran.s ~ age + celltype + trt, method='breslow')
summary(coxmod)

# compare a subject 50 years old with celltype 1
# to a subject 70 years old with celltype 2
# both subjects on the same treatment
AvB <- c(-20, -1, 0, 0, 0)

# compare a subject 40 years old with celltype 2 on treat=0
# to a subject 35 years old with celltype 3 on treat=1
CvB <- c(5, 1, -1, 0, -1)

est <- esticon(coxmod, L=rbind(AvB, CvB))
est
##exp(est[, c(2, 7, 8)])
}
```

---

`expr_to_fun` *Convert expression into function object.*

---

## Description

Convert expression into function object.

## Usage

```
expr_to_fun(expr_, order = NULL, vec_arg = FALSE)
```

## Arguments

| | |
|---|---|
| `expr_` | R expression. |
| `order` | desired order of function argument. |
| `vec_arg` | should the function take vector valued argument. |

## Examples

```
ee  <- expression(b1 + (b0 - b1)*exp(-k*x) + b2*x)
ff1 <- expr_to_fun(ee)
formals(ff1)

ff2 <- expr_to_fun(ee, vec_arg=TRUE)
formals(ff2)
formals(ff2)$length_parm
formals(ff2)$names_parm |> eval()

ee <- expression(matrix(c(x1+x2, x1-x2, x1^2+x2^2, x1^3+x2^3), nrow=2))
ff1 <- expr_to_fun(ee)
ff2 <- expr_to_fun(ee, vec_arg=TRUE)

formals(ff2)
formals(ff2)$length_parm
formals(ff2)$names_parm |> eval()
```

---

`fatacid` *Fish oil in pig food*

---

## Description

Fish oil in pig food

## Usage

```
fatacid
```

## Format

A dataframe.

## Details

A fish oil fatty acid `X14` has been added in different concentrations to the food for pigs in a study. Interest is in studying how much of the fatty acid can be found in the tissue. The concentrations of `x14` in the food are `verb+dose+={0.0, 4.4, 6.2, 9.3}`.

The pigs are fed with this food until their weight is 60 kg. From thereof and until they are slaughtered at 100kg, their food does not contain the fish oil. At 60kg (sample=1) and 100kg (sample=2) muscle biopsies are made and the concentration of x14 is determined. Measurements on the same pig are correlated, and pigs are additionally related through litters.

## References

Data courtesy of Charlotte Lauridsen, Department of Animal Science, Aarhus University, Denmark.

## fev *Forced expiratory volume in children*

### Description

Dataset to examine if respiratory function in children was influenced by smoking.

### Usage

```
fev
```

### Format

A data frame with 654 observations on the following 5 variables.

`Age` Age in years.

`FEV` Forced expiratory volume in liters per second.

`Ht` Height in centimeters.

`Gender` Gender.

`Smoke` Smoking status.

### References

I. Tager and S. Weiss and B. Rosner and F. Speizer (1979). Effect of Parental Cigarette Smoking on the Pulmonary Function of Children. American Journal of Epidemiology. 110:15-26

### Examples

```
data(fev)
summary(fev)
```

## firstlastobs *Locate the index of the first/last unique value*

### Description

Locate the index of the first/last unique value in i) a vector or of a variable in a data frame.

## Usage

```
lastobs(x, ...)

firstobs(x, ...)

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

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

## S3 method for class 'formula'
lastobs(formula, data = parent.frame(), ...)

## S3 method for class 'formula'
firstobs(formula, data = parent.frame(), ...)
```

## Arguments

| | |
|---|---|
| x | A vector |
| ... | Currently not used |
| formula | A formula (only the first term is used, see 'details'). |
| data | A data frame |

## Details

If writing ~a + b + c as formula, then only a is considered.

## Value

A vector.

## Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

## Examples

```
x <- c(rep(1, 5), rep(2, 3), rep(3, 7), rep(1, 4))
firstobs(x)
lastobs(x)
data(dietox)
firstobs(~Pig, data=dietox)
lastobs(~Pig, data=dietox)
```

formula_ops *Formula operations and coercion.*

## Description

Formula operations and coercion as a supplement to `update.formula()`

## Usage

```
formula_add_str(frm1, terms, op = "+")

formula_add(frm1, frm2)

formula_poly(chr1, n, noint = FALSE, y = NULL)

formula_nth(frm1, n)

formula_to_interaction_matrix(frm1)

formula_chr_to_form(rhs, lhs = character(0))

to_str(chr1, collapse = "+")

terms_labels(frm1)

simplify_rhs(object)

## S3 method for class 'formula'
simplify_rhs(object)

## S3 method for class 'character'
simplify_rhs(object)

as_rhs_frm(object)

as_lhs_frm(object)

as_rhs_chr(object, string = FALSE)

as_lhs_chr(object, string = FALSE)

unique_formula(list_of_formulas)
```

## Arguments

| | |
|---|---|
| `frm1, frm2` | Formulas to be coerced to character vectors. |
| `terms` | Character string. |

| | |
|---|---|
| op | Either "+" (default) or "-". |
| chr1 | Character vector to be coerced to formulas. |
| n | Positive integer. |
| noint | Boolean. |
| y | Response |
| rhs, lhs | right-hand-side and left-hand-side for formula (as characters) |
| collapse | Character to use as separator. |
| object | Character vector or formula. |
| string | Boolean. |
| list_of_formulas | list of formulas |

## Examples

```
formula_poly("z", 2)
formula_poly("z", 2, noint=TRUE)

as_rhs_chr(c("a", "b", "z"))
as_rhs_chr(c("a*b", "z"))

as_rhs_chr(y~a+b+z)
as_rhs_chr(y~a+b+z, string=TRUE)
as_rhs_chr(y~a+b+z)
as_rhs_chr(y~a*b+z)
as_rhs_chr(y~a*b+z, string=TRUE)

as_lhs_chr(y~a*b+z)
as_lhs_chr(log(y)~a*b+z)      ## Not what one might expect
as_lhs_chr(cbind(y, u)~a*b+z) ## Not what one might expect

formula_chr_to_form(c("a*b", "z"))
formula_chr_to_form(c("a*b", "z"), "y")
formula_chr_to_form(c("a*b", "z"), "log(y)")

formula_add(y~a*b+z, ~-1)
formula_add(y~a*b+z, ~a:b)

formula_add_str(y~x1 + x2, "x3")
formula_add_str(y~x1 + x2, "x1")
formula_add_str(y~x1 + x2, "x1", op="-")
```

---

# generate_data_list *Generate data list*

---

## Description

Generate data list

## Usage

```
generate_data_list(data., K, method = c("subgroups", "resample"))
```

## Arguments

| | |
|---|---|
| data. | A data frame |
| K | Number of folds |
| method | Method for generating data |

---

# get_formulas *Get formulas from model_stability_glm_class object*

---

## Description

Get formulas from model_stability_glm_class object

## Usage

```
get_formulas(object, unique = TRUE, text = FALSE)
```

## Arguments

| | |
|---|---|
| object | A model_stability_glm_class object |
| unique | If TRUE, return unique models |
| text | If TRUE, return text (rather than formula). |

---

# haldCement *Heat development in cement under hardening.*

---

## Description

Heat development in cement under hardening related to the chemical composition.

## Usage

```
haldCement
```

## Format

A data frame with 13 observations on the following 5 variables.

`x1` Percentage (weight) of `[3Ca0][Al2O3]`

`x2` Percentage (weight) of `[3Cao][SiO2]`

`x3` Percentage (weight) of `[4Ca0][Al2O3][Fe03]`

`x4` Percentage (weight) of `[2Cao][SiO2]`

`y` Heat development measured in calories per gram cement after 180 days

## References

Anders Hald (1949); Statistiske Metoder; Akademisk Forlag (in Danish), page 509.

## Examples

```
data(haldCement)

if( interactive() ){
pairs( haldCement )
}
m <- lm(y ~ x1 + x2 + x3 + x4, data=haldCement)
summary(m)

# Notice: The model explains practically all variation in data;
# yet none of the explanatory variables appear to be statistically
# significant.
```

---

# head_matrix *head and tail for matrices*

---

## Description

head and tail for matrices

## Usage

```
head2(x, n = 6, m = n)

tail2(x, n = 6, m = n)
```

## Arguments

| | |
|---|---|
| `x` | matrix |
| `n, m` | number of rows and columns |

## Examples

```
M <- matrix(1:20, nrow=4)
head2(M)
head2(M, 2)
```

---

`income` *Income data, years of educations and ethnicity*

---

## Description

Data on income, years of educations and ethnicity for a samle of adult Americans aged over 25. The year of sampling is not avalable in the source.

## Usage

```
income
```

## Format

This data frame contains:

**inc:** Income: Yearly income (thousands of dollars).

**educ:** Education: Number of years of education (12=high school graduate, 16=college graduate).

**race:** Racial-Ethnic group: "b" (black), "h" (hispanic) and "w" (white).

## Details

Variable names are as in the reference.

## References

Agresti, A. (2024) Statistical Methods for the Social Sciences, Global Edition (6th edition). ISBN-13: 9781292449197. Table 13.1

---

`interaction-plot` *Two-way interaction plot*

---

## Description

Plots the mean of the response for two-way combinations of factors, thereby illustrating possible interactions.

## Usage

```
interaction_plot(.data, .formula, interval = "conf.int")
```

### Arguments

| | |
|---|---|
| `.data` | A data frame |
| `.formula` | A formula of the form `y ~ x1 + x2` |
| `interval` | Either `conf.int`, `boxplot` or `none` |

### Note

This is a recent addition to the package and is subject to change.

### Examples

```
income$educf <- cut(income$educ, breaks=3)
income |> interaction_plot(inc ~ race + educf)
income |> interaction_plot(inc ~ race + educf, interval="conf.int")
income |> interaction_plot(inc ~ race + educf, interval="boxplot")
income |> interaction_plot(inc ~ race + educf, interval="none")
```

---

## `is_estimable` *Determines if contrasts are estimable.*

---

### Description

Determines if contrasts are estimable, that is, if the contrasts can be written as a linear function of the data.

### Usage

```
is_estimable(K, null.basis)
```

### Arguments

| | |
|---|---|
| `K` | A matrix. |
| `null.basis` | A basis for a null space (can be found with `null_basis()`). |

### Details

Consider the setting $E(Y) = Xb$. A linear function of $b$, say $l'b$ is estimable if and only if there exists an $r$ such that $r'X = l'$ or equivalently $l = X'r$. Hence $l$ must be in the column space of $X'$, i.e. in the orthogonal complement of the null space of $X$. Hence, with a basis $B$ for the null space, `is_estimable()` checks if each row $l$ of the matrix $K$ is perpendicular to each column basis vector in $B$.

### Value

A logical vector.

## Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

## References

http://web.mit.edu/18.06/www/Essays/newpaper_ver3.pdf

---

`linest` *Compute linear estimates*

---

## Description

Compute linear estimates, i.e. `L %*% beta` for a range of models. One example of linear estimates is population means (also known as LSMEANS).

## Usage

```
linest(object, L = NULL, level = 0.95, ...)

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

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

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

## Arguments

| | |
|---|---|
| `object` | Model object |
| `L` | Either `NULL` or a matrix with p columns where p is the number of parameters in the systematic effects in the model. If `NULL` then `L` is taken to be the p times p identity matrix |
| `level` | The level of the (asymptotic) confidence interval. |
| `...` | Additional arguments; currently not used. |
| `parm` | Specification of the parameters estimates for which confidence intervals are to be calculated. |
| `confint` | Should confidence interval appear in output. |

## Value

A dataframe with results from computing the contrasts.

## Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

## See Also

LSmeans, LE_matrix

## Examples

```
## Make balanced dataset
dat.bal <- expand.grid(list(AA=factor(1:2), BB=factor(1:3), CC=factor(1:3)))
dat.bal$y <- rnorm(nrow(dat.bal))

## Make unbalanced dataset
#   'BB' is nested within 'CC' so BB=1 is only found when CC=1
#   and BB=2,3 are found in each CC=2,3,4
dat.nst <- dat.bal
dat.nst$CC <-factor(c(1,1,2,2,2,2,1,1,3,3,3,3,1,1,4,4,4,4))

mod.bal  <- lm(y ~ AA + BB * CC, data=dat.bal)
mod.nst  <- lm(y ~ AA + BB : CC, data=dat.nst)

L <- LE_matrix(mod.nst, effect=c("BB", "CC"))
linest( mod.nst, L )
```

---

# linest-get *Auxillary functions for computing lsmeans, contrasts etc*

---

## Description

Auxillary functions for computing lsmeans, contrasts etc.

## Usage

```
get_xlevels(obj)

## Default S3 method:
get_xlevels(obj)

## S3 method for class 'mer'
get_xlevels(obj)

## S3 method for class 'merMod'
get_xlevels(obj)

get_contrasts(obj)
```

```
## Default S3 method:
get_contrasts(obj)

## S3 method for class 'merMod'
get_contrasts(obj)

set_xlevels(xlev, at)

get_vartypes(obj)

set_covariate_val(xlev, covariateVal)

get_X(obj, newdata, at = NULL)

## Default S3 method:
get_X(obj, newdata, at = NULL)

## S3 method for class 'merMod'
get_X(obj, newdata, at = NULL)
```

## Arguments

| | |
|---|---|
| obj | An R object |
| xlev | FIXME: to be described |
| at | FIXME: to be described |
| covariateVal | FIXME: to be described |
| newdata | FIXME: to be described |

---

`linest-matrix` *Linear estimates matrix*

---

## Description

Generate matrix specifying linear estimate.

## Usage

```
LE_matrix(object, effect = NULL, at = NULL)

## Default S3 method:
LE_matrix(object, effect = NULL, at = NULL)

aggregate_linest_list(linest_list)

get_linest_list(object, effect = NULL, at = NULL)
```

## Arguments

| | |
|---|---|
| object | Model object |
| effect | A vector of variables. For each configuration of these the estimate will be calculated. |
| at | Either NULL, a list or a dataframe. 1) If a list, then the list must consist of covariates (including levels of some factors) to be used in the calculations. 2) If a dataframe, the dataframe is split rowwise and the function is invoked on each row. |
| linest_list | Linear estimate list (as generated by get_linest_list). |

## Details

Check this

## See Also

LSmeans, linest

## Examples

```
## Two way anova:

data(warpbreaks)

## An additive model
m0 <- lm(breaks ~ wool + tension, data=warpbreaks)

## Estimate mean for each wool type, for tension="M":
K <- LE_matrix(m0, at=list(wool=c("A", "B"), tension="M"))
K

## Vanilla computation:
K %*% coef(m0)

## Alternative; also providing standard errors etc:
linest(m0, K)
esticon(m0, K)

## Estimate mean for each wool type when averaging over tension;
# two ways of doing this
K <- LE_matrix(m0, at=list(wool=c("A", "B")))
K
K <- LE_matrix(m0, effect="wool")
K
linest(m0, K)

## The linear estimate is sometimes called to "least squares mean"
# (LSmeans) or popupulation means.
# Same as
LSmeans(m0, effect="wool")
```

```
## Without mentioning 'effect' or 'at' an average across all
#predictors are calculated:
K <- LE_matrix(m0)
K
linest(m0, K)

## Because the design is balanced (9 observations per combination
#of wool and tension) this is the same as computing the average. If
#the design is not balanced, the two quantities are in general not
#the same.
mean(warpbreaks$breaks)

## Same as
LSmeans(m0)

## An interaction model
m1 <- lm(breaks ~ wool * tension, data=warpbreaks)

K <- LE_matrix(m1, at=list(wool=c("A", "B"), tension="M"))
K
linest(m1, K)
K <- LE_matrix(m1, at=list(wool=c("A", "B")))
K
linest(m1, K)
K <- LE_matrix(m1, effect="wool")
K
linest(m1, K)
LSmeans(m1, effect="wool")

K <- LE_matrix(m1)
K
linest(m1, K)
LSmeans(m1)
```

---

`ls-means` *Compute LS-means (aka population means or marginal means)*

---

## Description

LS-means (least squares means, also known as population means and as marginal means) for a range of model types.

## Usage

```
LSmeans(object, effect = NULL, at = NULL, level = 0.95, ...)

## Default S3 method:
LSmeans(object, effect = NULL, at = NULL, level = 0.95, ...)
```

```
## S3 method for class 'lmerMod'
LSmeans(object, effect = NULL, at = NULL, level = 0.95, adjust.df = TRUE, ...)

popMeans(object, effect = NULL, at = NULL, level = 0.95, ...)

## Default S3 method:
popMeans(object, effect = NULL, at = NULL, level = 0.95, ...)

## S3 method for class 'lmerMod'
popMeans(object, effect = NULL, at = NULL, level = 0.95, adjust.df = TRUE, ...)
```

## Arguments

| | |
|---|---|
| object | Model object |
| effect | A vector of variables. For each configuration of these the estimate will be calculated. |
| at | A list of values of covariates (including levels of some factors) to be used in the calculations |
| level | The level of the (asymptotic) confidence interval. |
| ... | Additional arguments; currently not used. |
| adjust.df | Should denominator degrees of freedom be adjusted? |

## Details

There are restrictions on the formulas allowed in the model object. For example having `y ~ log(x)` will cause an error. Instead one must define the variable `logx = log(x)` and do `y ~ logx`.

## Value

A dataframe with results from computing the contrasts.

## Warning

Notice that `LSmeans` and `LE_matrix` fails if the model formula contains an offset (as one would have in connection with e.g. Poisson regression.

## Note

`LSmeans` and `popMeans` are synonymous.

## Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

## See Also

`LE_matrix`, `linest`

## Examples

```
## Two way anova:

data(warpbreaks)

m0 <- lm(breaks ~ wool + tension, data=warpbreaks)
m1 <- lm(breaks ~ wool * tension, data=warpbreaks)
LSmeans(m0)
LSmeans(m1)

## same as:
K <- LE_matrix(m0);K
linest(m0, K)
K <- LE_matrix(m1);K
linest(m1, K)

LE_matrix(m0, effect="wool")
LSmeans(m0, effect="wool")

LE_matrix(m1, effect="wool")
LSmeans(m1, effect="wool")

LE_matrix(m0, effect=c("wool", "tension"))
LSmeans(m0, effect=c("wool", "tension"))

LE_matrix(m1, effect=c("wool", "tension"))
LSmeans(m1, effect=c("wool", "tension"))


## Regression; two parallel regression lines:

data(Puromycin)

m0 <- lm(rate ~ state + log(conc), data=Puromycin)
## Can not use LSmeans / LE_matrix here because of
## the log-transformation. Instead we must do:
Puromycin$lconc <- log( Puromycin$conc )
m1 <- lm(rate ~ state + lconc, data=Puromycin)

LE_matrix(m1)
LSmeans(m1)

LE_matrix(m1, effect="state")
LSmeans(m1, effect="state")

LE_matrix(m1, effect="state", at=list(lconc=3))
LSmeans(m1, effect="state", at=list(lconc=3))

## Non estimable contrasts

## ## Make balanced dataset
dat.bal <- expand.grid(list(AA=factor(1:2), BB=factor(1:3),
```

```
                            CC=factor(1:3)))
dat.bal$y <- rnorm(nrow(dat.bal))

## ## Make unbalanced dataset
#      'BB' is nested within 'CC' so BB=1 is only found when CC=1
#       and BB=2,3 are found in each CC=2,3,4
dat.nst <- dat.bal
dat.nst$CC <-factor(c(1, 1, 2, 2, 2, 2, 1, 1, 3, 3,
                      3, 3, 1, 1, 4, 4, 4, 4))

mod.bal  <- lm(y ~ AA + BB * CC,    data=dat.bal)
mod.nst  <- lm(y ~ AA + BB : CC, data=dat.nst)

LSmeans(mod.bal, effect=c("BB", "CC"))
LSmeans(mod.nst, effect=c("BB", "CC"))
LSmeans(mod.nst, at=list(BB=1, CC=1))

LSmeans(mod.nst, at=list(BB=1, CC=2))
## Above: NA's are correct; not an estimable function

if( require( lme4 )){
 warp.mm <- lmer(breaks ~ -1 + tension + (1|wool), data=warpbreaks)
 LSmeans(warp.mm, effect="tension")
 class(warp.mm)
 fixef(warp.mm)
 coef(summary(warp.mm))
 vcov(warp.mm)
 if (require(pbkrtest))
   vcovAdj(warp.mm)
}

LSmeans(warp.mm, effect="tension")
```

---

`math_teachers` *Height of math teachers*

---

## Description

Height of a sample of math teachers in Danish high schools collected at a continued education day at Mariager Fjord Gymnasium in 2019.

## Usage

```
math_teachers
```

## Format

:

**height** Height in centimeters

**sex** Male or female

## Examples

```
aggregate(height ~  sex, data=math_teachers, FUN=mean)
aggregate(height ~  sex, data=math_teachers, FUN=function(x) {c(mean=mean(x), sd=sd(x))})
```

---

`mb_summary` *Fast summary of microbenchmark object*

---

## Description

Fast summary of microbenchmark object. The default summary method from the microbenchmark package is fairly slow in producing a summary (due to a call to a function from the multcomp package.)

## Usage

```
mb_summary(object, unit, add.unit = TRUE, ...)

summary_mb(object, unit, add.unit = TRUE, ...)
```

## Arguments

| | |
|---|---|
| `object` | A microbenchmark object |
| `unit` | The time unit to be used |
| `add.unit` | Should time unit be added as column to resulting dataframe. |
| `...` | Additional arguments; currently not used. |

---

`milkman` *Milk yield data for manually milked cows.*

---

## Description

Milk yield data for cows milked manually twice a day (morning and evening).

## Usage

```
milkman
```

## Format

A data frame with 161836 observations on the following 12 variables.

`cowno` a numeric vector; cow identification

`lactno` a numeric vector; lactation number

`ampm` a numeric vector; milking time: 1: morning; 2: evening

`dfc` a numeric vector; days from calving

`my` a numeric vector; milk yield (kg)

`fatpct` a numeric vector; fat percentage

`protpct` a numeric vector; protein percentage

`lactpct` a numeric vector; lactose percentage

`scc` a numeric vector; somatic cell counts

`race` a factor with levels `RDM` `Holstein` `Jersey`

`ecmy` a numeric vector; energy corrected milk

`cowlact` Combination of cowno and lactno; necessary because the same cow may appear more than once in the dataset (in different lactations)

## Details

There are data for 222 cows. Some cows appear more than once in the dataset (in different lactations) and there are 288 different lactations.

## References

Friggens, N. C.; Ridder, C. and Løvendahl, P. (2007). On the Use of Milk Composition Measures to Predict the Energy Balance of Dairy Cows. J. Dairy Sci. 90:5453–5467 doi:10.3168/jds.2006-821.

This study was part of the Biosens project used data from the "Malkekoens energibalance og mobilisering" project; both were funded by the Danish Ministry of Food, Agriculture and Fisheries and the Danish Cattle Association.

## Examples

```
data(milkman)
```

## model_stability_glm *Model stability for glm objects*

### Description

Model stability for glm objects

### Usage

```
model_stability_glm(
  data.,
  model,
  n.searches = 10,
  method = c("subgroups", "resample"),
  ...
)
```

### Arguments

| | |
|---|---|
| data. | A data frame |
| model | A glm object |
| n.searches | Number of searches |
| method | Method for generating data |
| ... | Additional arguments to be passed to step |

## nir_milk *Near infra red light (NIR) measurements in milk*

### Description

Near infra red light (NIR) measurements are made at 152 wavelengths on 17 milk samples. While milk runs through a glass tube, infra red light is sent through the tube and the amount of light passing though the tube is measured at different wavelengths. Each milk sample was additionally analysed for fat, lactose, protein and dry matter.

### Usage

```
nir_milk

NIRmilk
```

## Format

Data comes in two formats:

nir_milk: A list with two components

- x Datafrane with infra red light amount at different wavelengths (column names are the wavelengths; just remove the leading X).
- y Datafrane with response variables fat, protein, lactose and dm (drymatter)

NIRmilk: This data frame contains 17 rows and 158 columns.

- The first column is the sample number.
- The columns named in the form `Xw` contains the transmittance (fraction of electromagnetic power) transmittance through the sample at wavelength `w`.
- The response variables are fat, protein, lactose and dm (dry matter).

An object of class `data.frame` with 17 rows and 158 columns.

## Examples

```
data(nir_milk)
data(NIRmilk)
```

---

# parseGroupFormula *Extract components from a formula with "conditioning bar"*

---

## Description

Extract components from a formula with the form `y ~ x1 + ... + xn | g1 + ... + gm`

## Usage

```
parseGroupFormula(form)
```

## Arguments

| | |
|---|---|
| `form` | A formula of the form `y ~ x1 + ... + xn \| g1 + ... + gm` |

## Value

If the formula is `y ~ x1 + x2 | g1 + g2` the result is

| | |
|---|---|
| `model` | `y ~ x1 + x2` |
| `groups` | `g1 + g2` |
| `groupFormula` | `~ g1 + g2` |

### Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

### Examples

```
gf1 <- parseGroupFormula(y ~ x1 + x2 | g1 + g2)
gf1

gf2 <- parseGroupFormula( ~ x1 + x2 | g1 + g2)
gf2
```

---

## pick_elements *Extract (pick) elements without using brackets*

---

### Description

Extract (pick) elements without using brackets so that elements can be picked out as part of a pipe workflow.

### Usage

```
pick1(x, which)

pick2(x, which)
```

### Arguments

| | |
|---|---|
| x | A list, data frame, or vector. |
| which | The index or name of the element(s) to extract. |

### Details

These two helper functions extract elements from lists, data frames, or vectors. They are simple wrappers for the standard bracket operators in R:

- `pick1()` uses single brackets (`[`) and returns a **subset**.
- `pick2()` uses double brackets (`[[`) and returns the **element itself**.

These are safer and more flexible than `$`, especially when used with the base R pipe (`|>`) or in functional programming.

### Value

- `pick1()` returns a subset of `x`.
- `pick2()` returns a single element from `x`.

## Examples

```
lst <- list(a = 1:3, b = 4:6)

# Without pipe
pick1(lst, "a")      # List with one element
pick2(lst, "a")      # Just the vector 1:3

# With base R pipe
lst |> pick1("a")
lst |> pick2("a")

df <- data.frame(x = 1:5, y = letters[1:5])

df |> pick1("y")     # Returns a data frame with column 'y'
df |> pick2("y")     # Returns column 'y' as a character vector
```

---

# pipe_arithmetic *Pipe-friendly arithmetic helpers*

---

## Description

A set of simple, vectorized, pipe-friendly arithmetic functions for transforming numeric data in pipelines. These helpers make common operations like multiplication, division, addition, subtraction, exponentiation, and reciprocals clearer when using the native pipe `|>`.

## Usage

```
reciprocal(x)

pow(x, p)

add(x, k)

subtract(x, k)

mult(x, k)

divide(x, k)
```

## Arguments

| | |
|---|---|
| x | A numeric vector or scalar. |
| p | A numeric scalar exponent (for pow). |
| k | A numeric scalar for addition, subtraction, multiplication, or division. |

## Details

All functions are vectorized and support numeric vectors, scalars, or compatible objects. They are designed to improve the readability of transformation pipelines.

## Value

A numeric vector or scalar resulting from the transformation.

## Examples

```
x <- c(1, 2, 3)

# Multiplication and division
x |> mult(10)
x |> divide(2)

# Addition and subtraction
x |> add(5)
x |> subtract(1)

# Reciprocal
x |> reciprocal()

# Power
x |> pow(2)

# Combined use in pipelines
x |>
  mult(2) |>
  add(3) |>
  reciprocal()
```

---

# plot_lm *Plot linear model object*

---

## Description

Plot linear model object

## Usage

```
plot_lm(lm_fit, format = "2x2", global_aes = NULL)
```

## Arguments

| | |
|---|---|
| lm_fit | An object of class 'lm' |
| format | The format of the plot (or a list of plots if format is "list") |
| global_aes | Currently no effect. |

## Examples

```
data(income)
m1 <- lm(inc ~ race + educ, data=income)
plot_lm(m1)
plot_lm(m1, "2x2")
plot_lm(m1, "1x4")
plot_lm(m1, "4x1")
plot_lm(m1, "list")
```

---

`potatoes` *Weight and size of 20 potatoes*

---

## Description

Weight and size of 20 potatoes. Weight in grams; size in millimeter. There are two sizes: `length` is the longest length and `width` is the shortest length across a potato.

## Usage

```
potatoes
```

## Format

A data frame with 20 observations on the following 3 variables.

`weight` a numeric vector

`length` a numeric vector

`width` a numeric vector

## Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

## Source

My own garden; autumn 2015.

## Examples

```
data(potatoes)
plot(potatoes)
```

prostate *Prostate Tumor Gene Expression Dataset*

## Description

This is the Prostate Tumor Gene Expression dataset used in Chung and Keles (2010).

## Usage

```
data(prostate)
```

## Format

A list with two components:

**x** Gene expression data. A matrix with 102 rows and 6033 columns.

**y** Class index. A vector with 102 elements.

## Details

The prostate dataset consists of 52 prostate tumor and 50 normal samples. Normal and tumor classes are coded in 0 and 1, respectively, in y vector. Matrix x is gene expression data and arrays were normalized, log transformed, and standardized to zero mean and unit variance across genes as described in Dettling (2004) and Dettling and Beuhlmann (2002). See Chung and Keles (2010) for more details.

## Source

Singh D, Febbo P, Ross K, Jackson D, Manola J, Ladd C, Tamayo P, Renshaw A, DAmico A, Richie J, Lander E, Loda M, Kantoff P, Golub T, and Sellers W (2002), "Gene expression correlates of clinical prostate cancer behavior", *Cancer Cell*, Vol. 1, pp. 203–209.

## References

Chung D and Keles S (2010), "Sparse partial least squares classification for high dimensional data", *Statistical Applications in Genetics and Molecular Biology*, Vol. 9, Article 17.

Dettling M (2004), "BagBoosting for tumor classification with gene expression data", *Bioinformatics*, Vol. 20, pp. 3583–3593.

Dettling M and Beuhlmann P (2002), "Supervised clustering of genes", *Genome Biology*, Vol. 3, pp. research0069.1–0069.15.

## Examples

```
data(prostate)
prostate$x[1:5,1:5]
prostate$y
```

## rbind_list *Bind list of data frames and add list names as a column*

### Description

Binds a named list of data frames (or tibbles) into a single data frame. Adds the list name as a new column (first column).

### Usage

```
rbind_list(lst, name = "name")
```

### Arguments

| | |
|---|---|
| lst | A named list of data frames or tibbles. |
| name | A character scalar: name of the column to hold the list names (default "name"). |

### Value

A data frame or tibble, depending on the class of the input.

### Examples

```
lst <- list(a = data.frame(x = 1:2), b = data.frame(x = 3:4))
rbind_list(lst)

lst <- split(iris, iris$Species)
rbind_list(lst)
```

## recodeVar *Recode values of a vector*

### Description

Recodes a vector with values, say 1,2 to a variable with values, say 'a', 'b'

### Usage

```
recodeVar(x, src, tgt, default = NULL, keep.na = TRUE)

recode_var(x, src, tgt, default = NULL, keep.na = TRUE)
```

## Arguments

| | |
|---|---|
| x | A vector; the variable to be recoded. |
| src | The source values: a subset of the present values of x |
| tgt | The target values: the corresponding new values of x |
| default | Default target value for those values of x not listed in src. When default=NULL, values of x which are not given in src will be kept in the output. |
| keep.na | If TRUE then NA's in x will be retained in the output |

## Value

A vector

## Warning

Care should be taken if x is a factor. A safe approach may be to convert x to a character vector using as.character.

## Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

## See Also

cut, factor, recodeVar

## Examples

```
x <- c("dec", "jan", "feb", "mar", "apr", "may")
src1 <- list(c("dec", "jan", "feb"), c("mar", "apr", "may"))
tgt1 <- list("winter", "spring")
recodeVar(x, src=src1, tgt=tgt1)
#[1] "winter" "winter" "winter" "spring" "spring" "spring"

x <- c(rep(1:3, 3))
#[1] 1 2 3 1 2 3 1 2 3

## Simple usage:
recodeVar(x, src=c(1, 2), tgt=c("A", "B"))
#[1] "A" "B" NA  "A" "B" NA  "A" "B" NA

## Here we need to use lists
recodeVar(x, src=list(c(1, 2)), tgt=list("A"))
#[1] "A" "A" NA  "A" "A" NA  "A" "A" NA
recodeVar(x, src=list(c(1, 2)), tgt=list("A"), default="L")
#[1] "A" "A" "L" "A" "A" "L" "A" "A" "L"
recodeVar(x, src=list(c(1, 2), 3), tgt=list("A", "B"), default="L")
#[1] "A" "A" "B" "A" "A" "B" "A" "A" "B"

## Dealing with NA's in x
x<-c(NA,rep(1:3, 3),NA)
```

```
#[1] NA  1  2  3  1  2  3  1  2  3 NA
recodeVar(x, src=list(c(1, 2)), tgt=list("A"))
#[1] NA  "A" "A" NA  "A" "A" NA  "A" "A" NA  NA
recodeVar(x, src=list(c(1, 2)), tgt=list("A"), default="L")
#[1] NA  "A" "A" "L" "A" "A" "L" "A" "A" "L" NA
recodeVar(x, src=list(c(1, 2)), tgt=list("A"), default="L", keep.na=FALSE)
#[1] "L" "A" "A" "L" "A" "A" "L" "A" "A" "L" "L"

x <- c("no", "yes", "not registered", "no", "yes", "no answer")
recodeVar(x, src = c("no", "yes"), tgt = c("0", "1"), default = NA)
```

---

`recover_pca_data` *Recover data from principal component analysis*

---

## Description

Recover data from principal component analysis based on the first (typically few) components.

## Usage

```
recover_pca_data(object, comp = 1)
```

## Arguments

| | |
|---|---|
| `object` | An object of class `prcomp`. |
| `comp` | The number of components to be used. Must be smaller than the number of variables. |

## Value

A dataframe

## Examples

```
crime <- doBy::crimeRate
rownames(crime) <- crime$state
crime$state <- NULL

o <- order(apply(scale(crime), 1, sum))
dat <- crime[o,]
head(dat)
tail(dat)
matplot(scale(dat), type="l")

pc1 <- prcomp(dat, scale. = TRUE)
summary(pc1)
rec2 <- recover_pca_data(pc1, 2)
```

```
pairs(rec2)

par(mfrow=c(1,2))
matplot(scale(dat), type="l")
matplot(scale(rec2), type="l")

j <- merge(dat, rec2, by=0)
pairs(j[,-1])
```

---

renameCol *Rename columns in a matrix or a dataframe.*

---

## Description

Rename columns in a matrix or a dataframe.

## Usage

```
renameCol(indata, src, tgt)
```

## Arguments

| | |
|---|---|
| indata | A dataframe or a matrix |
| src | Source: Vector of names of columns in indata to be renamed. Can also be a vector of column numbers. |
| tgt | Target: Vector with corresponding new names in the output. |

## Value

A dataframe if indata is a dataframe; a matrix in indata is a matrix.

## Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

## Examples

```
renameCol(CO2, 1:2, c("kk", "ll"))
renameCol(CO2, c("Plant", "Type"), c("kk", "ll"))

# These fail - as they should:
# renameCol(CO2, c("Plant", "Type", "conc"), c("kk", "ll"))
# renameCol(CO2, c("Plant", "Type", "Plant"), c("kk", "ll"))
```

## response *Get response variable from model*

### Description

Get response variable from model

### Usage

```
response(object)
```

### Arguments

| | |
|---|---|
| object | lm or glm object |

### Examples

```
data(cars)
lm1 <- lm(dist ~ speed + I(speed^2), data=cars)
lm1 |> response() |> head()
cars <- cars |> add_pred(lm1)
cars |> head()
cars <- cars |> add_resid(lm1)
cars
```

## response_plot *Plot the response variable against the predictor variables.*

### Description

Plot the response variable against the predictor variables.

### Usage

```
response_plot(
  data.,
  formula.,
  geoms = NULL,
  global_aes = NULL,
  plot = TRUE,
  nrow = NULL,
  ncol = NULL
)
```

### Arguments

| | |
|---|---|
| data. | A data frame containing the variables in the formula. |
| formula. | A formula of the form y ~ x1 + x2 + ... + xn, where y is the response variable and x1, x2, ..., xn are the predictor variables. A dot as right hand side is allowed. |
| geoms | A list of ggplot2 geoms to be added to the plot. |
| global_aes | A list of global aesthetics to be added to the plot. |
| plot | A logical value indicating whether the plot should be displayed. |
| nrow, ncol | Number of rows / columns in plot. |

### Value

A list of ggplot2 plots.

### Examples

```
library(ggplot2)
response_plot(iris, Sepal.Width ~ ., geoms=geom_point())
response_plot(iris, Sepal.Width ~ ., geoms=geom_point(), global_aes=list(color="Species"))
personality |> response_plot(easygon~., geoms=geom_point(), global_aes=NULL)
```

---

## scaleBy *Group-wise scaling of data*

---

### Description

Splits a data frame or matrix by grouping variables and scales numeric variables within each group.

### Usage

```
scaleBy(formula, data = parent.frame(), center = TRUE, scale = TRUE)

scale_by(data, formula, center = TRUE, scale = TRUE)
```

### Arguments

| | |
|---|---|
| formula | Grouping structure: a formula, character vector, or variables as as.quoted. |
| data | A data frame or matrix. |
| center | Logical; if TRUE, center the variables. |
| scale | Logical; if TRUE, scale the variables. |

### Value

A list of data frames or matrices (same class as input), one per group.

## Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

## See Also

summaryBy, transformBy, orderBy

## Examples

```
scale_by(iris, ~Species)
scale_by(iris, ~1)

## Combine result into one data frame:
a <- scale_by(iris, ~Species)
d <- do.call(rbind, a)

## Old interface
scaleBy(~Species, data = iris, center = TRUE, scale = FALSE)
scaleBy(~1, data = iris)
```

---

# scale_df *Scale numeric variables in a data frame*

---

## Description

Applies base::scale() to numeric, integer, or logical columns in a data frame. Non-numeric columns are left unchanged.

## Usage

```
scale_df(x, center = TRUE, scale = TRUE)
```

## Arguments

| | |
|---|---|
| x | A data frame or matrix. |
| center | Logical; if TRUE, center the variables. |
| scale | Logical; if TRUE, scale the variables. |

## Details

If x is not a data frame, base::scale() is applied directly.

## Value

An object of the same class as x.

## Examples

```
scale_df(iris)
```

---

`section_fun` *Section a function and set default values in function*

---

## Description

Section a functions domain by fixing certain arguments of a function call.

## Usage

```
section_fun(fun, nms, vls = NULL, method = "args")

section_fun_sub(fun, nms, vls = NULL, envir = parent.frame())

section_fun_env(fun, nms, vls = NULL)

get_section(object)

get_fun(object)
```

## Arguments

| | |
|---|---|
| `fun` | Function to be sectioned |
| `nms` | Either a named list of the form name=value where each name is the name of an argument of the function (in which case `vls` is ignored) or a character vector of names of arguments. |
| `vls` | A vector or list of values of the arguments |
| `method` | One of the following: 1) "args" (default); based on substituting fixed values into the function argument list as default values). For backward compatibility can also be "def". 2) "body" for substituting fixed values into the function body. For backward compatibility can also be "sub". 3) "env": (for environment); using an auxillary argument for storing sectioned values. |
| `envir` | Environment |
| `object` | An object from section_fun (a scaffold object). |

## Details

Let E be a subset of the cartesian product X x Y where X and Y are some sets. Consider a function f(x,y) defined on E. Then for any x in X, the section of E defined by x (denoted Ex) is the set of $y$s in Y such that (x, y) is in E. Correspondingly, the section of f(x,y) defined by x is the function $f_x$ defined on Ex given by $f_x(y)=f(x,y)$.

`section_fun` is a wrapper for calling `set_default` (default method), `section_fun_env` or `section_fun_sub`. Notice that creating a sectioned function with `section_fun_sub` can be time consuming.

## Value

A new function: The input function `fun` but with certain arguments fixed at specific values.

## Author(s)

Søren Højsgaard, <sorenh@math.aau.dk> based on code adapted from the curry package.

## See Also

`bquote_fun_list()`

## Examples

```
f  <- function(x, y){x + y}

f_ <- section_fun(f, list(y = 10),    method="args") ## "def"" is default
f_ <- section_fun(f, nms="y", vls=10, method="args") ## SAME AS ABOVE
f_
f_(x=1)

f_ <- section_fun(f, list(y = 10),    method="body") ##
f_ <- section_fun(f, nms="y", vls=10, method="body") ## SAME AS ABOVE
f_
f_(x=1)

f_ <- section_fun(f, list(y = 10),    method="env")
f_ <- section_fun(f, nms="y", vls=10, method="env") ## SAME AS ABOVE
f_
f_(x=1)
get_section(f_)
get_fun(f_)


## With more complicated values:
g <- function(A, B) {
  A + B
}
g_ <- section_fun(g, list(A = matrix(1:4, nrow=2)))
g_ <- section_fun(g, "A", list(matrix(1:4, nrow=2)))
g_(diag(1, 2))

g_ <- section_fun(g, list(A = matrix(1:4, nrow=2)))

## Using built in function
set.seed(123)
rnorm5 <- section_fun(rnorm, list(n=5))
rnorm5(0, 1)

set.seed(123)
rnorm(5)
```

set_default *Set default values in a functions arguments*

## Description

set_default() takes a function and returns a new version with updated default values for specified arguments.

## Usage

```
set_default(fun, nms, vls = NULL)
```

## Arguments

| | |
|---|---|
| fun | A function whose arguments will get new default values. |
| nms | Character vector of argument names, or something coercible to that (e.g. a call to v()). |
| vls | Optional vector or list of values to use as defaults. If nms is a named list, vls can be NULL. |

## Details

This is useful when you want to programmatically create specialized versions of a function with certain arguments preset to default values.

- The specified arguments will be moved to the end of the formal argument list in the returned function.
- You can supply arguments as a named list or as separate names and values.

## Value

A new function with updated default values in its formals.

## Examples

```
## Simple example
f1 <- function(x, y, z) { x + y + z }

## Add defaults for x and y
set_default(f1, list(x = 1, y = 2))
# function (z, x = 1, y = 2) { x + y + z }

## Same using separate vectors of names and values
set_default(f1, c("x", "y"), c(1, 2))

## Works with v() style if supported
# set_default(f1, v(x, y), c(1, 2))
```

```
## Another example with more arguments
f2 <- function(a, b, c, d) { a + b + c + d }
set_default(f2, list(b = 10, d = 5))
```

---

`set_list_set_matrix` *Matrix representatation of list of vectors and vice versa*

---

## Description

Matrix representatation of list of vectors and vice versa

## Usage

```
set_list2matrix(set_list, aggregate = FALSE)

matrix2set_list(set_matrix)
```

## Arguments

| | |
|---|---|
| `set_list` | list of vectors |
| `aggregate` | should the vectors be aggregated |
| `set_matrix` | matrix representatation |

## Examples

```
l <- list(c(1,2,3), c(3,2,4), c(3,2,4))
m1 <- set_list2matrix(l)
m1
matrix2set_list(m1)

m2 <- set_list2matrix(l, aggregate=TRUE)
m2
matrix2set_list(m2)
```

---

`shoes` *Wear of shoes*

---

## Description

Wear of soles of shoes of materials A and B for one foot each for of ten boys.

## Usage

```
shoes

shoes_long
```

## Format

This data frame contains:

**A:** Wear, material A

**B:** Wear, material B

**boy:** Id of boy

**footA:** The foot with material A

An object of class `data.frame` with 10 rows and 4 columns.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 20 rows and 4 columns.

## Details

The shoes data are measurements of the amount wear of the soles of shoes worn by 10 boys. The soles were made to two different synthetic materials, a standard material A and a cheaper material B.

## References

Box, Hunter, Hunter (2005) Statistics for Experimenters, 2nd edition Wiley, p. 81.

---

# split_byrow_bycol *Split matrix or dataframe into list*

---

## Description

Split matrix or dataframe into list by columns or by rows

## Usage

```
split_bycol(x, idx = NULL, as.list = FALSE)

split_byrow(x, idx = NULL)
```

## Arguments

| | |
|---|---|
| `x` | Matrix or dataframe. |
| `idx` | Index to split by. If NULL, split by columns or rows. |
| `as.list` | If TRUE, return list of dataframes. If FALSE, return list of matrices. |

## Examples

```
x <- mtcars[1:3, 1:6]
x  |> split_bycol()
x  |> split_bycol(as.list=TRUE)
x  |> split_bycol(as.list=FALSE)
x  |> split_bycol(idx=c(1,1,1,2,2,3,3,3))
## x  |> split_bycol(idx=c(1,1,7,2,2,3,3,3)) ## Gives error

x <- mtcars[1:6, 1:6]
x  |> split_byrow()
x  |> split_byrow(idx=c(1,1,2,2))

m <- as.matrix(x)
u <- x |> split_byrow(idx=c(1,1,2,2))
y <- m |> split_byrow(idx=c(1,1,2,2))
```

---

sub_seq *Find sub-sequences of identical elements in a vector.*

---

## Description

Find sub-sequences of identical elements in a vector.

## Usage

```
sub_seq(x, item = NULL)

subSeq(x, item = NULL)
```

## Arguments

| | |
|---|---|
| x | An atomic vector or a factor. |
| item | Optionally a specific value to look for in x. |

## Value

A dataframe.

## Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

## See Also

rle

### Examples

```
x <- c(1, 1, 1, 0, 0, 1, 1, 1, 2, 2, 2, 1, 2, 2, 2, 3)
sub_seq(x)
sub_seq(x, item=1)
```

---

## taylor *Taylor expansion (one dimension)*

---

### Description

Returns Taylor polynomial approximating a function fn(x)

### Usage

```
taylor(fn, x0, ord = 1)
```

### Arguments

| | |
|---|---|
| fn | A function of one variable and that variable must be named 'x'. |
| x0 | The point in which to to the Taylor expansion. |
| ord | The order of the Taylor expansion. |

### Value

function.

### Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

### Examples

```
fn <- function(x) log(x)
ord <- 2
x0 <- 2

xv  <- seq(.2, 5, .1)
plot(xv, fn(xv), type="l")
lines(xv, taylor(fn, x0=x0, ord=ord)(xv), lty=2)
abline(v=x0)

fn <- function(x)sin(x)
ord <- 4
x0 <- 0
xv <- seq(-2*pi, 2*pi, 0.1)
plot(xv, fn(xv), type="l")
lines(xv, taylor(fn, x0=x0, ord=ord)(xv), lty=2)
abline(v=x0)
```

## tidy-esticon *Tidy an esticon object*

### Description

Tidy summarizes information about the components of the object.

### Usage

```
## S3 method for class 'esticon_class'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

### Arguments

| | |
|---|---|
| x | A 'esticon_class' object (produced by esticon methods). |
| conf.int | Should confidence intervals be added. |
| conf.level | Desired confidence level. |
| ... | Additional arguments; currently not used. |

## tidy-linest *Tidy a linest object*

### Description

Tidy summarizes information about the components of the object.

### Usage

```
## S3 method for class 'linest_class'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

### Arguments

| | |
|---|---|
| x | A 'linest_class' object (produced by linest methods). |
| conf.int | Should confidence intervals be added. |
| conf.level | Desired confidence level. |
| ... | Additional arguments; currently not used. |

`time_since_event` *Calculate "time since event" in a vector*

## Description

Calculates the time since the nearest event in a sequence, optionally using a custom time scale.

## Usage

```
time_since_event(yvar, tvar = seq_along(yvar))

timeSinceEvent(...)
```

## Arguments

| | |
|---|---|
| `yvar` | A numeric or logical vector indicating events. |
| `tvar` | An optional numeric vector specifying time values. Defaults to the index. |
| `...` | Arguments pased on to time_since_event |

## Details

Events are coded as 1 (or TRUE). Non-events are anything else. The result includes absolute and signed distances to events.

## Value

A data frame with columns 'yvar', 'tvar', 'abs.tse' (absolute time since event), 'sign.tse' (signed time since event), and other helper columns.

## Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

## See Also

`subSeq`, `rle`

## Examples

```
## Example 1: Basic usage with default time index
y <- c(0, 0, 1, 0, 0, 1, 0)
tse <- time_since_event(y)
print(tse)

## Example 2: Custom (non-integer) time variable
y <- c(0, 0, 1, 0, 0, 0, 1, 0)
t <- seq(0.5, 3.5, length.out = length(y))
tse <- time_since_event(y, t)
```

```
print(tse)

## Example 3: Plotting the signed time since event
plot(sign.tse ~ tvar, data = tse, type = "b",
     main = "Signed time since event",
     xlab = "Time", ylab = "Signed time since event")
grid()
abline(h = 0, col = "red", lty = 2)
```

---

`transform_forecast` *Transform forecasts from model scale to data scale by simulation*

---

## Description

`transform_forecast()` takes a univariate time series forecast fitted on a transformed (model) scale and produces a new forecast object on the original data scale. This is done by simulating future paths on the model scale, transforming each path with a user-supplied function, and then computing pointwise means and prediction intervals on the transformed scale.

## Usage

```
transform_forecast(fc_object, trans_fun, nsim = 2000L, level = 95, y0 = 1)
```

## Arguments

| | |
|---|---|
| `fc_object` | A `"forecast"` object (from the **forecast** package) for a univariate time series, typically obtained via `forecast::forecast()`. |
| `trans_fun` | A function of the form `function(x, y0) ...` that takes a numeric vector `x` on the model scale and a scalar starting value `y0` on the output scale and returns a numeric vector of the same length on the output scale. |
| `nsim` | Integer; number of simulated future paths to use. Larger values give smoother prediction intervals but take longer to compute. |
| `level` | Numeric; prediction interval coverage in percent. |
| `y0` | Numeric; starting value on the output scale used to reconstruct the historical series from `fc_object$x`. For percentage change models it is often natural to set `y0 = 1` and interpret the resulting series as an index. |

## Details

The function assumes that `fc_object` is a `"forecast"` object as produced by the **forecast** package, and that `fc_object$model` supports `simulate()` with arguments `nsim` and `future`.

The transformation function `trans_fun` must have the form `trans_fun(x, y0)`, where `x` is a numeric vector representing a path on the model scale (for example log-values or percentage changes), and `y0` is a scalar "starting value" on the output scale. The function must return a numeric vector of the same length as `x` giving the corresponding path on the output scale.

Internally, `transform_forecast()` first reconstructs a historical series on the output scale by applying `trans_fun()` to `fc_object$x` with the supplied `y0`. It then simulates `nsim` future paths from `fc_object$model` on the model scale, transforms each path to the output scale using `trans_fun()` with `y0` equal to the last value of the reconstructed historical series, and finally computes the pointwise mean and prediction intervals (of nominal coverage `level`) across the simulated paths. The result is returned as a new `"forecast"` object with `x`, `mean`, `lower`, and `upper` on the output scale.

## Value

A `"forecast"` object similar to `fc_object`, but with the components `x`, `mean`, `lower`, and `upper` defined on the output (data) scale.

## See Also

`forecast`, `auto.arima`, `simulate`, `ts`.

## Examples

```
## Example 1: Log-transform of the Canadian lynx data
if (requireNamespace("forecast", quietly = TRUE)) {

  llynx   <- log(lynx)
  fit_log <- forecast::auto.arima(llynx)
  fc_log  <- forecast::forecast(fit_log, h = 20)
  forecast::autoplot(fc_log)
  ## transformation: log -> original scale
  trans_log <- function(z, y0) {
    exp(z)
  }
  fc_lynx <- transform_forecast(fc_log, trans_fun = trans_log,
                            nsim = 20, level = 95, y0 = 1)
  plot(fc_lynx)
  forecast::autoplot(fc_lynx) + ggplot2::theme_minimal()
}


## Not run:
if (requireNamespace("forecast", quietly = TRUE)) {
## Example 2 (variation): CO2 series, log-transform
  lco2    <- log(co2)
  fit_co2 <- forecast::auto.arima(lco2)
  fc_log_co2 <- forecast::forecast(fit_co2, h = 24)
  forecast::autoplot(fc_log_co2)
  trans_log <- function(z, y0) exp(z)

  fc_co2 <- transform_forecast(fc_log_co2, trans_fun = trans_log,
                           nsim = 20, level = 95, y0 = 1)
  forecast::autoplot(fc_co2) + ggplot2::theme_minimal()
}

# ## Example 3: Percentage change in income (uschange$Income)
if (requireNamespace("forecast", quietly = TRUE) &&
```

```
  requireNamespace("fpp2", quietly = TRUE)) {
 income <- uschange[, "Income"]  # quarterly percentage changes (%)

 ## transformation: pct change -> index with base 1
 trans_pct <- function(r, y0) {
   y0 * cumprod(1 + r / 100)
 }

 fit_pct <- forecast::auto.arima(income)
 fc_pct  <- forecast::forecast(fit_pct, h = 24)
 forecast::autoplot(fc_pct)
 fc_idx  <- transform_forecast(fc_pct, trans_fun = trans_pct,
                           nsim = 200, level = 95, y0 = 1)

 plot(fc_idx)
 forecast::autoplot(fc_idx) + ggplot2::theme_minimal()
}

## End(Not run)
```

---

`truncate0` *Truncate values in a matrix / vector to zero if they are below a certain threshold.*

---

## Description

Truncate values in a matrix / vector to zero if they are below a certain threshold.

## Usage

```
truncate0(x, tol = 0.6, sparse = TRUE)
```

## Arguments

| | |
|---|---|
| `x` | matrix / vector |
| `tol` | threshold |
| `sparse` | logical; if TRUE and x is a matrix, return a sparse matrix |

## v *Shorthand for vparse()*

### Description

A short and convenient alias for `vparse()`. Accepts unquoted names, character vectors, or a formula.

### Usage

```
v(...)
```

### Arguments

| | |
|---|---|
| `...` | Variable input in any accepted `vparse()` form |

### Value

A character vector of variable names

## vcheck *Check if variables exist in a data frame*

### Description

Check if variables exist in a data frame

### Usage

```
vcheck(df, ...)
```

### Arguments

| | |
|---|---|
| `df` | A data frame |
| `...` | Variables to check |

### Value

TRUE if all variables exist, otherwise error

## vmap *Apply a function to each parsed variable name*

### Description

Apply a function to each parsed variable name

### Usage

```
vmap(.vars, .f)
```

### Arguments

| | |
|---|---|
| .vars | Variables to parse |
| .f | Function to apply to each name |

### Value

A list of results

## vparse *Variable utilities: parse, select, check, map, rename*

### Description

These functions provide flexible tools for parsing and managing variable names from formulas, unquoted names, or character vectors. Demonstrated using CO2 dataset.

### Usage

```
vparse(...)
```

### Arguments

| | |
|---|---|
| ... | Variable input (unquoted, character vector, or formula) |

## vrename *Rename columns in a data frame*

### Description

Rename columns in a data frame

### Usage

```
vrename(df, rename_map)
```

### Arguments

| | |
|---|---|
| df | A data frame |
| rename_map | A named character vector (old_name = new_name) |

### Value

A data frame with renamed columns

## vselect *Select columns from a data frame using flexible input*

### Description

Select columns from a data frame using flexible input

### Usage

```
vselect(df, ...)
```

### Arguments

| | |
|---|---|
| df | A data frame |
| ... | Variable input (unquoted, character vector, or formula) |

### Value

A data frame with selected columns

`which.maxn` *Where are the n largest or n smallest elements in a numeric vector ?*

## Description

Determines the locations, i.e., indices of the n largest or n smallest elements of a numeric vector.

## Usage

```
which.maxn(x, n = 1)
```

## Arguments

| | |
|---|---|
| `x` | numeric vector |
| `n` | integer >= 1 |

## Value

A vector of length at most n with the indices of the n largest / smaller elements. NAs are discarded and that can cause the vector to be smaller than n.

## Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

## See Also

`which.max`, `which.min`

## Examples

```
x <- c(1:4, 0:5, 11, NA, NA)
ii <- which.minn(x, 5)

x <- c(1, rep(NA,10), 2)
ii <- which.minn(x, 5)
```

# Index