

# Package ‘doBy’

January 30, 2012

**Version** 4.5.1

**Title** doBy - Groupwise summary statistics, general linear contrasts, population means (least-squares-means), and other utilities

**Author** Søren Højsgaard <sorenh@mail.dk> and Ulrich Halekoh <Ulrich.Halekoh.agrsci.dk> with contributions from Jim Robison-Cox, Kevin Wright and Alessandro A. Leidi.

**Maintainer** Søren Højsgaard <sorenh@mail.dk>

**Description** doBy contains a variety of utilities including: 1) Facilities for groupwise computations of summary statistics and other facilities for working with grouped data. 2) General linear contrasts and LSMEANS (least-squares-means also known as population means), 3) Rscript2HTML for automatic generation of HTML file from R-script with a minimum of markup, 4) other utilities. doBy originally contained facilities for ‘doing something to data where data would be partitioned by some variables which define groupings of data’ - hence the name doBy.

**Encoding** latin1

**URL** <http://people.math.aau.dk/~sorenh/software/doBy/>

**ZipData** no

**License** GPL (>= 2)

**Imports** Matrix

**Depends** R (>= 2.10), survival, R2HTML, multcomp, lme4, snow, MASS

**Repository** CRAN

**Date/Publication** 2012-01-30 06:27:30

**R topics documented:**

|                          |           |
|--------------------------|-----------|
| beets . . . . .          | 2         |
| budworm . . . . .        | 3         |
| codstom . . . . .        | 4         |
| dietox . . . . .         | 6         |
| doBy . . . . .           | 7         |
| dose.LD50 . . . . .      | 8         |
| esticon . . . . .        | 9         |
| firstlastobs . . . . .   | 11        |
| lapplyBy . . . . .       | 12        |
| lmBy . . . . .           | 13        |
| milkman . . . . .        | 14        |
| orderBy . . . . .        | 15        |
| popMeans . . . . .       | 16        |
| recodeVar . . . . .      | 18        |
| renameCol . . . . .      | 19        |
| Rmarkup . . . . .        | 20        |
| sampleBy . . . . .       | 22        |
| scaleBy . . . . .        | 23        |
| splitBy . . . . .        | 24        |
| subSeq . . . . .         | 25        |
| subsetBy . . . . .       | 26        |
| summaryBy . . . . .      | 27        |
| timeSinceEvent . . . . . | 29        |
| transformBy . . . . .    | 30        |
| which.maxn . . . . .     | 31        |
| <b>Index</b>             | <b>33</b> |

---

|       |  |
|-------|--|
| beets | <i>Yield and sugar percentage in sugar beets from a split plot experiment.</i> |
|-------|--|

---

**Description**

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  $25m^2$  and the yield is recorded in kg. See 'details' for the experimental layout.

**Usage**

```
data(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 1-15  | 1 1 1 1 1             | 2 2 2 2 2 | 1 1 1 1 1 | Harvest time |
|            | 3 4 5 2 1             | 3 2 4 5 1 | 5 2 3 4 1 | Sowing time  |
|            | +-----+ -----+ -----+ |           |           |              |
| Plot 16-30 | 2 2 2 2 2             | 1 1 1 1 1 | 2 2 2 2 2 | Harvest time |
|            | 2 1 5 4 3             | 4 1 3 2 5 | 1 4 3 2 5 | Sowing time  |
|            | +-----+ -----+ -----+ |           |           |              |

**Examples**

```
data(beets)
## maybe str(beets) ; plot(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))
```

---

budworm

*Effect of Insecticide on survival of tobacco budworms*

---

**Description**

Number of killed budworms after exposure to an insecticide.

**Usage**

```
data(budworm)
```

**Format**

This data frame contains 12 rows and 4 columns:

**sex:** sex of the budworm

**dose:** dose of the insecticide trans-cypermethrin in  $\mu g$

**ndead:** budworms killed in a trial

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

**Details**

Mortality of the moth tobacco budworm 'Heliothis virescens' for 6 doses of the pyrethroid trans-cypermethrin differentiated with respect to sex.

**Source**

Collet, 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 calculate 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='empirical logit')
title('budworm: empirical 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))
```

---

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**

```
data(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 Leptoicl\_mac Mallo\_t\_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 commercial 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 regions. 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](mailto:chabotd@dfo-mpo.gc.ca)).

**Examples**

```

data(codstom)
str(codstom)
# removes multiple occurrences of same prey.type in stomachs
codstom1 <- summaryBy(preymass ~
  region+ship.type+ship.id+trip+set+fish.id+prey.type,
  data = codstom, id = ~fish.length,
  keep.names=TRUE, FUN = sum)

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

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

```

---

 dietox

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


---

**Description**

The dietox data frame has 861 rows and 7 columns.

Data contains weight of slaughter pigs measured weekly for 12 weeks. Data also contains the startweight (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 littermates.

**Usage**

```
data(dietox)
```

**Format**

This data frame contains the following columns:

**Weight** Weight  
**Feed** Cumulated feed intake  
**Time** Time (in weeks) in the experiment  
**Pig** Id of each pig  
**Evit** Vitamin E dose  
**Cu** Copper dose  
**Start** Start weight in experiment, i.e. weight at week 1.  
**Litter** Id of litter of each pig

**Source**

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

**Examples**

```
data(dietox)
str(dietox) ;
plot(dietox)
```

---

doBy

*Various utilities which includes functions for creating groupwise calculations etc.*

---

**Description**

The core doBy functions were developed to make it easy to split data into groups (defined by the levels of a set of factors) and performing some actions on each of these groups. Thus, these functions mimic what can be achieved using the BY statement in various SAS procedures.

In addition hereto the doBy package contains various other utilities.

**Details**

Functions summaryBy, splitBy, orderBy, sampleBy, transformBy are the core doBy functions

There is no need for a plotBy function – the xyplot function in the lattice package already fulfills these needs

The esticon function calculates linear functions of parameter estimates under various types of models.

There are various other utility functions in the package.

**Author(s)**

Søren Højsgaard, sorenh at mail dot dk

**See Also**

[summaryBy](#), [orderBy](#), [transformBy](#), [splitBy](#), [sampleBy](#)

**Examples**

```
data(dietox)

summaryBy(Weight+Feed~Evit+Cu+Time,      data=dietox, FUN=c(mean,var),
na.rm=TRUE, use="pair")

orderBy(~Time+Evit, data=dietox)

splitBy(formula = ~Evit+Cu, data = dietox)

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

---

dose.LD50

*Calculate LD50*

---

**Description**

Calculate the LD50 (the dose at which 50 pct of the subjects die) for a model of the form  $\text{logit}(p) = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p + \gamma d$  where none of the explanatory variables  $x_1 \dots x_p$  contains the dose  $d$ .

**Usage**

```
dose.LD50(x, lambda)
```

**Arguments**

`x`                    A glm object (for logistic regression)  
`lambda`                A vector of the same length as the number of parameters in `x`.

**Details**

`lambda` contains an NA at the entry corresponding to dose  $d$ . The other entries of `lambda` must be the values of the covariates  $x_1 \dots x_p$  at which the ld50 is to be calculated.

**Value**

A data frame

**Author(s)**

Søren Højsgaard

**Examples**

```
data(budworm)
m1 <- glm(ndeath/20 ~ sex + log(dose), data=budworm, weight=ntotal, family=binomial)
coef(m1)

dose.LD50(m1,c(1,1,NA))
dose.LD50(m1,c(1,0,NA))
```

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, cm, beta0, conf.int = TRUE, level=0.95, joint.test = FALSE,...)
```

**Arguments**

|            |  |
|------------|--|
| obj        | Regression object (of type lm, glm, lme, geeglm)   |
| cm         | Matrix 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 $L\beta = \beta_0$ is made. Default is that the 'row-wise' tests are made, i.e. $(L\beta)_i = \beta_{0i}$ . If joint.test is TRUE, then no confidence interval etc. is calculated. |
| ...        | Additional arguments; currently not used.  |

**Details**

Let the estimated parameters of the model be

*latex*

A linear function of the estimates is of the form

*latex*

where *latex* is specified by the user.

The *esticon* function calculates *c*, its standard error and by default also a 95 pct confidence interval. It is sometimes of interest to test the hypothesis *latex* for some value *latex* given by the user. A test is provided for the hypothesis *latex* but other values of *latex* can be specified.

In general, one can specify *r* such linear functions at one time by specifying *cm* to be an *latex* matrix where each row consists of *p* numbers *latex*. Default is then that *latex* is a *p* vector of 0s but other values can be given.

It is possible to test simulatneously that all specified linear functions are equal to the corresponding values in *latex*.

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.

### Note

'esticon' works on *geese/geeglm* objects from the *geepack* package (for Generalized Estimating Equations), on 'lm' and 'glm' objects, and on 'gls' objects.

### Author(s)

Søren Højsgaard, *sorenh at mail dot 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,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,lambda2,beta0=1)

## Do both estimates at one time
esticon(lm1,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,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

library(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)

esti<-esticon(coxmod,rbind(AvB,CvB))
esti
exp(esti[,c(2,7,8)])
```

---

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

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

## Arguments

x                    A vector

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

**Details**

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

**Value**

A vector.

**Author(s)**

Søren Højsgaard, `sorenh at mail dot 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)
```

---

lapplyBy

*Formula based version of lapply*


---

**Description**

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

**Usage**

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

**Arguments**

|         |  |
|---------|--|
| formula | A formula describing how data should be split  |
| data    | A dataframe  |
| FUN     | A function to be applied to each element in the splitted list, see 'Examples' below. |

**Value**

A list.

**Author(s)**

Søren Højsgaard, sorenh at mail dot dk

**See Also**

[orderBy](#), [summaryBy](#), [transformBy](#), [splitBy](#),

**Examples**

```
data(dietox)

## Calculate weekwise feed efficiency = weight gain / feed intake
dietox <- orderBy(~Pig+Time, data=dietox)
v<-lapplyBy(~Pig, data=dietox, function(d) c(NA, diff(d$Weight)/diff(d$Feed)))
dietox$FE <- unlist(v)

## Technically this is the same as
dietox <- orderBy(~Pig+Time, data=dietox)
wdata <- splitBy(~Pig, data=dietox)
v <- lapply(wdata, function(d) c(NA, diff(d$Weight)/diff(d$Feed)))
dietox$FE <- unlist(v)
```

---

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**

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

**Arguments**

|         |  |
|---------|--|
| formula | A linear model formula object of the form $y \sim x_1 + \dots + x_n \mid g_1 + \dots + g_m$ . In the formula object, $y$ represents the response, $x_1, \dots, x_n$ the covariates, and the grouping factors specifying the partitioning of the data according to which different lm fits should be performed. |
| data    | A dataframe  |
| id      | A formula describing variables from data which are to be available also in the output.   |
| ...     | Additional arguments passed on to <code>lm()</code> .  |

**Value**

A list of lm fits.

**Note**

This is a recent addition to the package; please report bugs.

**Author(s)**

Søren Højsgaard <sorenh at mail dot dk>

**Examples**

```
## To be written...
```

---

milkman

*Milk yield data for manually milked cows.*

---

**Description**

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

**Usage**

```
data(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.

## Examples

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

---

|         |  |
|---------|--|
| orderBy | <i>Ordering (sorting) rows of a data frame</i> |
|---------|--|

---

## 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

```
orderBy(formula, data)
```

## Arguments

|         |                                  |
|---------|----------------------------------|
| formula | The right hand side of a formula |
| data    | A data frame                     |

## 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 at mail dot dk` and Kevin Wright

## See Also

[summaryBy](#), [transformBy](#), [splitBy](#), [lapplyBy](#),

**Examples**

```
data(dietox)
orderBy(~Time+Evit, data=dietox)
## Sort decreasingly by Time
orderBy(~-Time+Evit, data=dietox)
```

popMeans

*Calculate population means (LSMEANS in SAS jargon)***Description**

Calculate population means (LSMEANS in SAS jargon).

**Usage**

```
popMeans(object, effect, at = NULL, only.at=TRUE, engine=esticon,...)
## Default S3 method:
popMeans(object, effect, at = NULL, only.at=TRUE, engine=esticon,...)
## S3 method for class 'lme'
popMeans(object, effect, at = NULL, only.at=TRUE, engine=esticon,...)
```

**Arguments**

|         |  |
|---------|--|
| object  | A model object; currently only available for 'lm' objects  |
| effect  | A vector of variables for which the population means should be calculated.   |
| at      | A list of covariate values   |
| only.at | If FALSE and at is not NULL then the contrast matrix is formed as if at was not specified. Then the values of at are set afterwards such that the contrast matrix will contain several identical rows. |
| engine  | The function for actually calculating the linear combination of the parameters. See 'details'.   |
| ...     | Additional arguments; currently not used.  |

**Details**

The function is lsMeans() is just an alias for popMeans().

The popMeans() function generates the relevant contrast matrix and then calls the engine on that matrix. Another possible engine is the glht function from the multcomp package

**Value**

A dataframe with a few additional attributes.

**Warning**

This function is a recent addition to the 'doBy' package and it is not thoroughly tested. Use with caution!

**Author(s)**

Søren Højsgaard, sorenh at mail dot dk

**See Also**

[esticon glht](#)

**Examples**

```

dat <- expand.grid(list(AA=factor(1:2), BB=factor(1:3), CC=factor(1:3)))
dat$y <- rnorm(nrow(dat))
dat$x <- rnorm(nrow(dat))
dat$x2 <- dat$x^2

## Examples

## 1) LSMEANS with factors only.
mod1 <- lm(y ~ AA + BB*CC + x, data=dat)
## Average over AA for each combination of (BB,CC); evaluate at x=mean(x)
popMeans(mod1, c("BB","CC"))
## Average over (AA,BB) for each value of CC; evaluate at x=mean(x)
popMeans(mod1, c("CC"))

## 2) The call to popMeans() below is equivalent to the following SAS code
## proc glm data=dat;
## class AA BB CC;
## model y = AA BB|CC x x*x;
## lsmeans CC BB*CC / stderr;
## run;
##
mod2 <- lm(y ~ AA + BB*CC + x + x2, data=dat)
popMeans(mod2, "CC")

## Notice the difference to:
mod3 <- lm(y ~ AA + BB*CC + x + I(x^2), data=dat)
popMeans(mod3, "CC")

## The difference arises because in the former case, x2 is evaluated at mean(x2) whereas
## in the latter case x is evaluated at mean(x)^2

## 3) Plug in particular values of covariates
## The call to popMeans() below is equivalent to the following SAS code
## proc glm data=dat;
## class AA BB CC;
## model y = AA BB|CC x x2;
## lsmeans CC BB*CC / at x=2 stderr;
## run;
popMeans(mod2, c("CC"), at=list(x=2))
## Above, x=2 is used while x2 is set to mean(x2)

## The call to popMeans() below is equivalent to the following SAS code
## proc glm data=dat;

```

```
## class AA BB CC;
## model y = AA BB|CC x x*x;
## lsmeans CC BB*CC / at x=2 stderr;
## run;
##
popMeans(mod3, c("CC"), at=list(x=2))
## Above, x=2 is used while I(x^2) is set to mean(x^2)=4. Notice that setting
## popMeans(mod3, c("CC"), at=list(x=2,"I(x^2)"=123))
## has no effect: I(x^2) is still 4 because x=2. Hence the following two results
## are identical

popMeans(mod2, c("CC"), at=list(x=2, x2=4))
popMeans(mod3, c("CC"), at=list(x=2))

## END
```

---

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)
```

### 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 at mail dot dk>

**See Also**

[cut](#), [factor](#), [recode](#)

**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"
```

---

renameCol

*Rename columns in a matrix or a dataframe.*

---

**Description**

Rename columns in a matrix or a dataframe.

**Usage**

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

**Arguments**

|                     |   |
|---------------------|---|
| <code>indata</code> | A dataframe or a matrix   |
| <code>src</code>    | Source: Vector of names of columns in 'indata' to be renamed. Can also be a vector of column numbers. |
| <code>tgt</code>    | 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 at mail dot dk>

**See Also**

[rename.vars](#)

**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"))
```

---

Rmarkup

*Automatic Generation of Reports (as HTML documents)*

---

**Description**

HTMLreport provides a simple framework for mixing text and R code for automatic report generation. The basic idea is to replace the R code with its output, such that the final document only contains the text and the output of the statistical analysis.

**Usage**

```
Rmarkup(srcfile, driver = RweaveHTMLreport(), destdir=".",
postfix="REPORT", cssfile=NULL, cleanup=TRUE,
parms=list(height=400,width=600), details=0,...)
```

**Arguments**

|         |   |
|---------|---|
| srcfile | A text file with R code and descriptive text. The file name extension can be anything except .html as this is the name of the output file |
| driver  | The actual workhorse  |
| destdir | Specification of where to put the report.   |
| postfix | A string to append to the filename for the HTML file.   |
| cssfile | An optional cssfile; must be located in destdir to take effect.   |
| cleanup | If set to FALSE then the temporary files are not removed.   |
| parms   | Control height and width of graphics.   |
| details | Set to a positive value to see intermediate results; mainly for debugging purposes.   |
| ...     | Further arguments, currently not used.  |

**Value**

None

**Note**

HTMLreport is a recent addition to the doBy package and it has not been thoroughly tested.

**Author(s)**

Søren Højsgaard <sorenh at mail dot dk>

**See Also**

[Sweave](#)

**Examples**

```
tf <- system.file("HTMLreport", "PuromycinAnalysis-report.R",
  package = "doBy")

## Create report in working directory
Rmarkup(tf)
## Creates report in specified directory (which must exist).
## Rmarkup(tf, path=".REPORT/")
```

---

`sampleBy`*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**

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

**Arguments**

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

**Author(s)**

Søren Højsgaard, `sorenh at mail dot dk`

**See Also**

[orderBy](#), [summaryBy](#), [transformBy](#), [splitBy](#),

**Examples**

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

---

scaleBy *Groupwise scaling and centering of numeric columns in a dataframe*

---

### Description

Groupwise scaling and centering of numeric columns in a dataframe

### Usage

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

### Arguments

|         |   |
|---------|---|
| formula | A formula. A dot (.) is allowed on both left and right hand side of formula. See 'details' for the meaning of this. |
| data    | A dataframe   |
| center  | If TRUE then data is centered to have mean zero   |
| scale.  | If TRUE then data is scaled to have variance one  |

### Details

A typical formula is  $y_1 + y_2 \sim f_1 + f_2$  where  $y_1$  and  $y_2$  are numeric variables and  $f_1$  and  $f_2$  can be of any type. For each cross-combination of the values of  $f_1$  and  $f_2$ , the variables ( $y_1, y_2$ ) are centered/scaled.

It is valid to write  $\sim f_1 + f_2$ . In this case the variables to be centered/scaled are taken to be all numeric variables in the dataframe except those that are listed on the right hand side of the formula.

It is valid to write  $y_1 + y_2 \sim \dots$ . In this case the stratification is taken to be by all non-numeric variables. If there are no non-numeric variables, then no stratification is made and a 'global' centering/scaling is made.

It is valid to write  $\sim \dots$ . In this case the variables to be centered/scaled are taken to be all numeric variables in the dataframe. The stratification is made by all non-numeric variables. If there are no non-numeric variables, then no stratification is made and a 'global' centering/scaling is made.

### Value

A dataframe

### Note

The workhorse is the `funBy` function which, alas, is not yet documented.

### Author(s)

Søren Højsgaard, `sorenh at mail dot dk`

**See Also**

[lapplyBy](#), [orderBy](#),  
[splitBy](#), [summaryBy](#), [transformBy](#),

**Examples**

```
data(dietox)

# "Remove" the effect of time by centering data within each time point.
dietox2 <- scaleBy(Weight~Time, data=dietox, scale=FALSE)

## Not run:
library(lattice)
xyplot(Weight~Time|Evit+Cu, groups=Pig, data=dietox)
xyplot(Weight~Time|Evit+Cu, groups=Pig, data=dietox2)

## End(Not run)
```

---

splitBy

*Split a data frame*


---

**Description**

Split a dataframe according to the levels of variables in the dataframe. The variables to split by can be given as a formula or as a character vector.

**Usage**

```
splitBy(formula, data = parent.frame(), drop=TRUE, return.matrix=FALSE)
## S3 method for class 'splitByData'
print(x, ...)
```

**Arguments**

|               |   |
|---------------|---|
| formula       | The right hand side of a formula (or a character vector)                        |
| data          | A data frame  |
| drop          | Logical indicating if levels that do not occur should be dropped                |
| return.matrix | Should the returned list consist of dataframes or matrices, see 'details' below |
| x             | A splitByData object (basically a list).  |
| ...           | Additional arguments, currently not used.                                       |

**Details**

The function transform the dataframe 'data' into a numerical matrix (using the 'asNumericMatrix' function from the Hmisc package) and makes the splitting operation on this. If return.matrix is TRUE, then these matrices are returned, otherwise the matrices are turned into dataframes and then these are returned.

**Value**

A list of dataframes of matrices

**Author(s)**

Søren Højsgaard, sorenh at mail dot dk

**See Also**

[orderBy](#), [summaryBy](#), [transformBy](#),  
[lapplyBy](#),

**Examples**

```
data(dietox)
splitBy(formula = ~Evit+Cu, data = dietox)
```

---

subSeq

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

---

**Description**

Find sub-sequences of identical elements in a vector.

**Usage**

```
subSeq(x, item = NULL)
```

**Arguments**

x                    An atomic vector.  
item                 Optionally a specific value to look for in 'x'.

**Value**

A dataframe.

**Author(s)**

Søren Højsgaard <sorenh at mail dot dk>

**See Also**[rle](#)**Examples**

```
x <- c(1,1,1,0,0,1,1,1,2,2,2,1,2,2,2,3)
(ans <- subSeq(x))
ans$value
# Notice: Same results below
subSeq(x,item=1)
subSeq(x,item="1")

x <- as.character(c(1,1,1,0,0,1,1,1,2,2,2,1,2,2,2,3))
(ans <- subSeq(x))
ans$value
# Notice: Same results below
subSeq(x,item="1")
subSeq(x,item=1)
```

subsetBy

*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**

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

**Arguments**

|         |   |
|---------|---|
| formula | A formula to split by   |
| subset  | logical expression indicating elements or rows to keep: missing values are taken as false.                  |
| data    | A data frame  |
| 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 at mail dot dk

**See Also**

See Also [splitBy](#)

**Examples**

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

---

summaryBy

---

*Function to calculate groupwise summary statistics*


---

**Description**

Function to calculate groupwise summary statistics, much like the summary procedure of SAS

**Usage**

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

**Arguments**

|                |   |
|----------------|---|
| formula        | A formula object, see examples below  |
| data           | A data frame  |
| id             | A formula specifying variables which data are not grouped by but which should appear in the output. See examples below.                                   |
| FUN            | A list of functions to be applied, see examples below.  |
| keep.names     | If TRUE and if there is only ONE function in FUN, then the variables in the output will have the same name as the variables in the input, see 'examples'. |
| p2d            | Should parentheses in output variable names be replaced by dots?  |
| order          | Should the resulting dataframe be ordered according to the variables on the right hand side of the formula? (using <a href="#">orderBy</a> )              |
| full.dimension | If TRUE then rows of summary statistics are repeated such that the result will have the same number of rows as the input dataset.                         |
| ...            | Additional arguments to FUN. This could for example be NA actions.  |

**Details**

Extra arguments ('...') are passed onto the functions in FUN. Hence care must be taken that all functions in FUN accept these arguments - OR one can explicitly write a functions which get around this. This can particularly be an issue in connection with handling NAs. See examples below.

Some code for this function has been suggested by Jim Robison-Cox.

**Value**

A data frame

**Author(s)**

Søren Højsgaard, sorenh at mail dot dk

**See Also**

[ave](#), [lapplyBy](#), [orderBy](#), [scaleBy](#), [splitBy](#),  
[transformBy](#),

**Examples**

```
data(dietox)
dietox12 <- subset(dietox,Time==12)

summaryBy(Weight+Feed~Evit+Cu, data=dietox12,
  FUN=c(mean,var,length))

summaryBy(Weight+Feed~Evit+Cu+Time, data=subset(dietox,Time>1),
  FUN=c(mean,var,length))

## Calculations on transformed data:

summaryBy(log(Weight)+Feed~Evit+Cu, data=dietox12)

## Calculations on all numerical variables (not mentioned elsewhere):

summaryBy(.~Evit+Cu, data=dietox12,
  id=~Litter, FUN=mean)

## There are missing values in the 'airquality' data, so we remove these
## before calculating mean and variance with 'na.rm=TRUE'. However the
## length function does not accept any such argument. Hence we get
## around this by defining our own summary function in which length is
## not supplied with this argument while mean and var are:

sumfun <- function(x, ...){
  c(m=mean(x, ...), v=var(x, ...), l=length(x))
}
summaryBy(Ozone+Solar.R~Month, data=airquality, FUN=sumfun, na.rm=TRUE)

## Using '.' on the right hand side of a formula means to stratify by
## all variables not used elsewhere:

data(warpbreaks)
summaryBy(breaks ~ wool+tension, warpbreaks)
summaryBy(breaks ~., warpbreaks)
summaryBy(.~ wool+tension, warpbreaks)
```

```
## Keep the names of the variables (works only if FUN only returns one
## value):

summaryBy(Ozone+Wind~Month, data=airquality, FUN=c(mean), na.rm=TRUE,
  keep.names=TRUE)

## Using full.dimension=TRUE

## Consider:
summaryBy(breaks~wool, data=warpbreaks)
## Rows of result are replicated below
summaryBy(breaks~wool, data=warpbreaks, full.dimension=TRUE)
## Notice: Previous result is effectively the same as
with(warpbreaks, ave(breaks, wool))
## A possible application of full.dimension=TRUE is if we want to
## standardize (center and scale) data within groups:
ss <- summaryBy(breaks~wool, data=warpbreaks, full.dimension=TRUE, FUN=c(mean,sd))
(warpbreaks$breaks-ss$breaks.mean)/ss$breaks.sd
```

---

|                |  |
|----------------|--|
| timeSinceEvent | <i>Calculate "time since event" in a vector.</i> |
|----------------|--|

---

### Description

Events are coded as 1 in numeric vector (and non-events are coded with values different from 1). `timeSinceEvent` will give the time since event (with and without sign). In a logical vector, events are coded as TRUE and all non-events as FALSE.

### Usage

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

### Arguments

|      |   |
|------|---|
| yvar | A numerical or logical vector specifying the events |
| tvar | An optional vector specifying time                  |

### Value

A dataframe with columns 'yvar', 'tvar', 'abs.tse' (absolute time since nearest event), 'sign.tse' (signed time since nearest event) and 'run' (indicator of the time window around each event).

### Note

NA's in yvar are converted to zeros.

**Author(s)**

Søren Højsgaard <sorenh at mail dot dk>

**See Also**

[subSeq, rle](#)

**Examples**

```
## Events:
yvar <- c(0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0)

## Plot results:
tse<- timeSinceEvent(yvar)
plot(sign.tse~tvar, data=tse, type="b")
grid()
rug(tse$tvar[tse$yvar==1], col=4,lwd=4)
points(scale(tse$run), col=tse$run,lwd=2)
lines(abs.tse+.2~tvar, data=tse, type="b",col=3)

## Find times for which time since an event is at most 1:
tse$tvar[tse$abs<=1]

yvar <- c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
)
tvar <- c(207, 208, 208, 208, 209, 209, 209, 209, 210, 210, 211, 211,
211, 212, 213, 213, 214, 214, 215, 216, 216, 216, 216, 217, 217,
217, 218, 218, 219, 219, 219, 219, 220, 220, 221, 221, 221, 221,
222, 222, 222)

timeSinceEvent(yvar, tvar)
```

---

transformBy

*Function to make groupwise transformations*


---

**Description**

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

**Usage**

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

**Arguments**

|         |   |
|---------|---|
| formula | A formula with only a right hand side, see examples below |
| data    | A data frame  |
| ...     | 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 at mail dot dk

**See Also**

[orderBy](#), [summaryBy](#), [splitBy](#), [doby.xtabs](#),

**Examples**

```
data(dietox)
transformBy(~Pig, data=dietox, minW=min(Weight), maxW=max(Weight),
  gain=sum(range(Weight)*c(-1,1)))
```

---

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

**Arguments**

|   |                  |
|---|------------------|
| x | numeric vector   |
| n | integer $\geq 1$ |

**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@mail dot dk](mailto:sorenh@mail dot 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

## \*Topic **datasets**

beets, 2  
budworm, 3  
codstom, 4  
dietox, 6  
milkman, 14

## \*Topic **models**

dose.LD50, 8  
lmBy, 13  
popMeans, 16

## \*Topic **univar**

summaryBy, 27  
transformBy, 30

## \*Topic **utilites**

popMeans, 16

## \*Topic **utilities**

doBy, 7  
esticon, 9  
firstlastobs, 11  
lapplyBy, 12  
orderBy, 15  
recodeVar, 18  
Rmarkup, 20  
sampleBy, 22  
scaleBy, 23  
splitBy, 24  
subSeq, 25  
subsetBy, 26  
timeSinceEvent, 29  
which.maxn, 31

## \*Topic **utilities**

renameCol, 19

ave, 28

beets, 2

budworm, 3

codstom, 4

coef.lmBy (lmBy), 13

cut, 19

dietox, 6

doBy, 7

doby.xtabs, 31

dose.LD50, 8

esticon, 9, 17

factor, 19

firstlastobs, 11

firstobs (firstlastobs), 11

fitted.lmBy (lmBy), 13

getBy (lmBy), 13

glht, 17

lapplyBy, 12, 15, 24, 25, 28

lastobs (firstlastobs), 11

lmBy, 13

milkman, 14

orderBy, 8, 13, 15, 22, 24, 25, 27, 28, 31

popMeans, 16

print.lmBy (lmBy), 13

print.splitByData (splitBy), 24

recode, 19

recodeVar, 18

rename.vars, 20

renameCol, 19

residuals.lmBy (lmBy), 13

rle, 26, 30

Rmarkup, 20

sampleBy, 8, 22

scaleBy, 23, 28

splitBy, 8, 13, 15, 22, 24, 24, 27, 28, 31

subSeq, 25, 30

subsetBy, 26  
summary.conMeans (popMeans), 16  
summaryBy, 8, 13, 15, 22, 24, 25, 27, 31  
Sweave, 21  
  
timeSinceEvent, 29  
transformBy, 8, 13, 15, 22, 24, 25, 28, 30  
  
which.max, 32  
which.maxn, 31  
which.min, 32  
which.minn (which.maxn), 31