Package: BMisc (via r-universe)

November 20, 2026

Title Miscellaneous Functions for Panel Data, Quantiles, and Printing Results

Version 1.4.11

Description These are miscellaneous functions for working with panel data, quantiles, and printing results. For panel data, the package includes functions for making a panel data balanced (that is, dropping missing individuals that have missing observations in any time period), converting id numbers to row numbers, and to treat repeated cross sections as panel data under the assumption of rank invariance. For quantiles, there are functions to make distribution functions from a set of data points (this is particularly useful when a distribution function is created in several steps), to combine distribution functions based on some external weights, and to invert distribution functions. Finally, there are several other miscellaneous functions for obtaining weighted means, weighted distribution functions, and weighted quantiles; to generate summary statistics and their differences for two groups; and to add or drop covariates from formulas. Additional utilities support staggered treatment adoption settings, including functions for identifying treatment groups, recovering pre-treatment outcomes and covariate averages, and computing lagged outcomes and first differences in panel data.

Depends R (>= 4.1.0)

Imports data.table, Rcpp

License GPL-3

Suggests testthat (>= 3.0.0), plm, tibble, caret

Encoding UTF-8

Config/testthat/edition 3

LinkingTo Rcpp, RcppArmadillo

URL https://bcallaway11.github.io/BMisc/

BugReports https://github.com/bcallaway11/BMisc/issues

Config/roxygen2/version 8.1.1

NeedsCompilation yes

Author Brantly Callaway [aut, cre], Grant McDermott [ctb], Kyle Butts [ctb]

Maintainer Brantly Callaway <brantly.callaway@uga.edu>

Repository https://cran.r-universe.dev

Date/Publication 2026-11-20 18:40:02 UTC

RemoteUrl https://github.com/cran/BMisc

RemoteRef HEAD

RemoteSha 8a04d72a020ec97ade5489d47e3921982666bb79

Contents

add_cov_to_formula . . . 3
block_boot_sample . . . 4
check_staggered . . . 4
checkfun . . . 5
combine_ecdfs . . . 6
compare_binary . . . 7
cs2panel . . . 8
drop_collinear . . . 8
drop_cov_from_formula . . . 9
element_wise_mult . . . 10
get_first_difference . . . 10
get_group . . . 11
get_lagYi . . . 12
get_list_element . . . 13
get_principal_components . . . 13
get_Yi1 . . . 14
get_Yibar . . . 15
get_Yibar_pre . . . 16
get_YiGmin1 . . . 17
get_Yit . . . 18
ids2rownum . . . 19
invert_ecdf . . . 19
lhs_vars . . . 20
make_balanced_panel . . . 21
make_dist . . . 21
multiplier_bootstrap . . . 22
mv_mult . . . 23
orig2t . . . 24
panel2cs . . . 24
panel2cs2 . . . 25
quantile.BMisc_dist . . . 26
rhs . . . 27

rhs_vars 27
set_balanced_panel 28
sort_panel 29
source_all 29
subsample 30
t2orig 30
time_invariant_to_panel 31
toformula 32
TorF 33
weighted_checkfun 33
weighted_combine_list 34
weighted_ecdf 35
weighted_mean 35
weighted_quantile 36

Index **37**

`add_cov_to_formula` *Add a Covariate to a Formula*

Description

`add_cov_to_formula` adds some covariates to a formula; covs should be a list of variable names

Usage

```
add_cov_to_formula(covs, formula)
```

Arguments

`covs`	should be a list of variable names
`formula`	which formula to add covariates to

Value

formula

Examples

```
ff <- y ~ x
add_cov_to_formula(list("w", "z"), ff)

ff <- ~x
add_cov_to_formula("z", ff)
```

block_boot_sample *Block Bootstrap*

Description

make draws of all observations with the same id in a panel data context. This is useful for bootstrapping with panel data.

Usage

```
block_boot_sample(data, idname)
```

Arguments

data	data.frame from which you want to bootstrap
idname	column in data which contains an individual identifier

Value

data.frame bootstrapped from the original dataset; this data.frame will contain new ids

Examples

```
data("LaborSupply", package = "plm")
bbs <- block_boot_sample(LaborSupply, "id")
nrow(bbs)
head(bbs$id)
```

check_staggered *check_staggered*

Description

A function to check whether treatment is staggered in a panel data set; that is, whether treatment is absorbing, so that no unit ever reverts from treated back to untreated.

Usage

```
check_staggered(df, idname, treatname, tname = NULL)
```

Arguments

df	the data.frame used in the function
idname	name of column that holds the unit id
treatname	name of column with the treatment indicator
tname	name of column that holds the time period. If supplied, each unit's rows are sorted by it before checking. If NULL (the default), the rows are assumed to already be in time order.

Value

a logical indicating whether treatment is staggered

Examples

```
n <- 50
id <- rep(seq_len(n), each = 4)
t <- rep(1:4, n)
g <- rep(sample(c(0, 2, 3), n, replace = TRUE), each = 4)
treat <- as.integer(t >= g & g > 0)
dta <- data.frame(id = id, t = t, treat = treat)
check_staggered(dta, idname = "id", treatname = "treat")

## treatment that switches back off is not staggered
dta_rev <- data.frame(
  id = c(1, 1, 2, 2), t = c(1, 2, 1, 2), treat = c(0, 1, 1, 0)
)
check_staggered(dta_rev, idname = "id", treatname = "treat")
```

---

checkfun *Check Function*

Description

The check function used for optimizing to get quantiles

Usage

```
checkfun(a, tau)
```

Arguments

a	vector to compute quantiles for
tau	between 0 and 1, ex. .5 implies get the median

Value

numeric value

Examples

```
x <- rnorm(100)
x[which.min(checkfun(x, 0.5))] ## should be around 0
```

`combine_ecdfs` *Combine Distribution Functions*

Description

Combines distribution functions into their mixture $F(y) = \sum_c w_c F_c(y)$, with weights given by `weights`.

By default (`y.seq = NULL`) the mixture is exact: when every distribution function is a step function (as from `stats::ecdf()`, `weighted_ecdf()` or `make_dist()`), so is the mixture, with a jump at each support point y_{cj} of each F_c equal to w_c times the jump of F_c there. These are stacked and sorted, and their cumulative sums give the mixture at every support point. `quantile()` of the result is then exact as well.

With `y.seq`, the mixture is instead evaluated only at the points `y.seq` (as in BMisc 1.4.10 and earlier), and its quantiles are rounded to those points; this also works for distribution functions that are not step functions.

Usage

```
combine_ecdfs(y.seq = NULL, ecdflist, weights = NULL, ...)
```

Arguments

`y.seq`	`NULL` (the default) for the exact mixture, or a sequence of y values at which to evaluate it
`ecdflist`	list of ecdf objects (distribution functions) to combine
`weights`	a vector of weights to put on each distribution function, which should sum to 1; if weights are not provided then equal weight is given to each distribution function. Distribution functions with weight 0 are skipped in the exact mixture.
`...`	additional arguments that can be past to BMisc::make_dist

Value

ecdf (a `BMisc_dist` object, see `make_dist()`)

Examples

```
x <- rnorm(100)
y <- rnorm(100, 1, 1)
Fx <- ecdf(x)
Fy <- ecdf(y)
both <- combine_ecdfs(ecdflist = list(Fx, Fy))
```

```
quantile(both, 0.5) # exact median of the mixture
plot(Fx, col = "green")
plot(Fy, col = "blue", add = TRUE)
plot(both, add = TRUE)

# evaluated on a grid only
both_grid <- combine_ecdfs(seq(-2, 3, 0.1), list(Fx, Fy))
```

`compare_binary` *Compare Variables across Groups*

Description

`compare_binary` takes in a variable e.g. union and runs bivariate regression of x on treatment (for summary statistics)

Usage

```
compare_binary(
  x,
  on,
  dta,
  w = rep(1, nrow(dta)),
  report = c("diff", "levels", "both")
)
```

Arguments

`x`	variables to run regression on
`on`	binary variable
`dta`	the data to use
`w`	weights
`report`	which type of report to make; diff is the difference between the two variables by group

Value

matrix of results

Examples

```
dta <- data.frame(x = rnorm(100), treat = rep(c(0, 1), 50))
compare_binary("x", "treat", dta, report = "diff")
```

cs2panel *Cross Section to Panel*

Description

Turn repeated cross sections data into panel data by imposing rank invariance; does not require that the inputs have the same length

Usage

```
cs2panel(cs1, cs2, yname)
```

Arguments

cs1	data frame, the first cross section
cs2	data frame, the second cross section
yname	the name of the variable to calculate difference for (should be the same in each dataset)

Value

the change in outcomes over time

Examples

```
cs1 <- data.frame(y = rnorm(100))
cs2 <- data.frame(y = rnorm(100, mean = 1))
dy <- cs2panel(cs1, cs2, "y")
mean(dy) ## approx 1
```

drop_collinear *drop_collinear*

Description

A function to check for multicollinearity and drop collinear terms from a matrix

Usage

```
drop_collinear(matrix)
```

Arguments

matrix	a matrix for which the function will remove collinear columns

Value

a matrix with collinear columns removed

Examples

```
if (requireNamespace("caret", quietly = TRUE)) {
  X <- cbind(1:5, 2 * (1:5), rnorm(5))
  colnames(X) <- c("x1", "x2", "x3")
  drop_collinear(X) ## x2 dropped as collinear with x1
}
```

---

drop_cov_from_formula *Drop a Covariate from a Formula*

Description

drop_cov_from_formula adds drops some covariates from a formula; covs should be a list of variable names

Usage

```
drop_cov_from_formula(covs, formula)
```

Arguments

covs	should be a list of variable names
formula	the formula to drop covariates from

Value

formula

Examples

```
ff <- y ~ x + w + z
drop_cov_from_formula(list("w", "z"), ff)

drop_cov_from_formula("z", ff)
```

element_wise_mult *element_wise_mult*

Description

This is a function that takes in two matrices of dimension nxB and nxk and returns a Bxk matrix that comes from element-wise multiplication of every column in the first matrix times the entire second matrix and the averaging over the n-dimension. It is equivalent (but faster than) the following R code: `sapply(1:biters, function(b) sqrt(n)*colMeans(Umat[,b]*inf.func))` . This function is particularly useful for fast computations using the multiplier bootstrap.

Usage

```
element_wise_mult(U, inf_func)
```

Arguments

U	nxB matrix (e.g., these could be a matrix of Rademachar weights for B bootstrap iterations using the multiplier bootstrap
inf_func	nxk matrix of (e.g., these could be a matrix containing the influence function for different parameter estimates)

Value

a Bxk matrix

get_first_difference *get_first_difference*

Description

A function that calculates the first difference in a panel data setting. If the data.frame that is passed in has nxT rows, the resulting vector will also have nxT elements with one element for each unit set to be NA.

Usage

```
get_first_difference(df, idname, yname, tname)
```

Arguments

df	the data.frame used in the function
idname	name of column that holds the unit id
yname	name of column containing the outcome (or other variable) for which to calculate its outcome in the immediate pre-treatment period
tname	name of column that holds the time period

Value

a vector with one element per row of df: the change in yname from the unit's previous period (ordered by tname), NA for each unit's first period

Examples

```
n <- 50
id <- rep(seq_len(n), each = 4)
t <- rep(1:4, n)
y <- rnorm(n * 4)
dta <- data.frame(id = id, t = t, y = y)
dy <- get_first_difference(dta, idname = "id", yname = "y", tname = "t")
```

---

get_group *get_group*

Description

A function to calculate a unit's group in a panel data setting with a binary treatment and staggered treatment adoption and where there is a column in the data indicating whether or not a unit is treated

Usage

```
get_group(df, idname, tname, treatname)
```

Arguments

df	the data.frame used in the function
idname	name of column that holds the unit id
tname	name of column that holds the time period
treatname	name of column with the treatment indicator

Value

a numeric vector with one element per row of df: the unit's group, i.e. the first period in which it is treated, or 0 for units that are never treated

Examples

```
n <- 50
id <- rep(seq_len(n), each = 4)
t <- rep(1:4, n)
g <- rep(sample(c(0, 2, 3), n, replace = TRUE), each = 4)
treat <- as.integer(t >= g & g > 0)
dta <- data.frame(id = id, t = t, treat = treat)
dta$group <- get_group(dta, idname = "id", tname = "t", treatname = "treat")
head(unique(dta[, c("id", "group")]))
```

get_lagYi *get_lagYi*

Description

A function that calculates lagged outcomes in a panel data setting. If the data.frame that is passed in has nxT rows, the resulting vector will also have nxT elements with one element for each unit set to be NA

Usage

```
get_lagYi(df, idname, yname, tname, nlags = 1)
```

Arguments

df	the data.frame used in the function
idname	name of column that holds the unit id
yname	name of column containing the outcome (or other variable) for which to calculate its outcome in the immediate pre-treatment period
tname	name of column that holds the time period
nlags	The number of periods to lag. The default is 1, which computes the lag from the previous period.

Value

a vector with one element per row of df: the value of yname nlags periods earlier for the same unit (ordered by tname), NA for each unit's first nlags periods

Examples

```
n <- 50
id <- rep(seq_len(n), each = 4)
t <- rep(1:4, n)
y <- rnorm(n * 4)
dta <- data.frame(id = id, t = t, y = y)
dta$lag_y <- get_lagYi(dta, idname = "id", yname = "y", tname = "t")
```

get_list_element *Return Particular Element from Each Element in a List*

Description

a function to take a list and get a particular part out of each element in the list

Usage

```
get_list_element(listolists, whichone = 1)
```

Arguments

listolists	a list
whichone	which item to get out of each list (can be numeric or name)

Value

list of all the elements 'whichone' from each list

Examples

```
len <- 100 # number elements in list
lis <- lapply(1:len, function(l) list(x = (-l), y = l^2)) # create list
get_list_element(lis, "x")[1] # should be equal to -1
get_list_element(lis, 1)[1] # should be equal to -1
```

get_principal_components *get_principal_components*

Description

A function to calculate unit-specific principal components, given panel data

Usage

```
get_principal_components(
  xformula,
  data,
  idname,
  tname,
  n_components = NULL,
  ret_wide = FALSE,
  ret_id = FALSE
)
```

Arguments

xformula	a formula specifying the variables to use in the principal component analysis
data	a data.frame containing the panel data
idname	the name of the column containing the unit id
tname	the name of the column containing the time period
n_components	the number of principal components to retain, the default is NULL which will result in all principal components being retained
ret_wide	whether to return the data in wide format (where the number of rows is equal to n = length(unique(data[[idname]])) or long format (where the number of rows is equal to nT = nrow(data)). The default is FALSE, so that long data is returned by default.
ret_id	whether to return the id column in the output data.frame. The default is FALSE.

Value

a data.frame containing the original data with the principal components appended

Examples

```
n <- 20
id <- rep(seq_len(n), each = 4)
t <- rep(1:4, n)
x1 <- rnorm(n * 4)
dta <- data.frame(id = id, t = t, x1 = x1)
pcs <- get_principal_components(~x1, dta, idname = "id", tname = "t")
dim(pcs)
```

---

get_Yi1 *get_Yi1*

Description

A function to calculate outcomes for units in the first time period that is available in a panel data setting (this function can also be used to recover covariates, etc. in the first period).

Usage

```
get_Yi1(df, idname, yname, tname, gname)
```

Arguments

df	the data.frame used in the function
idname	name of column that holds the unit id
yname	name of column containing the outcome (or other variable) for which to calculate its outcome in the immediate pre-treatment period
tname	name of column that holds the time period
gname	name of column containing the unit's group

Value

a vector with one element per row of `df`: the unit's value of `yname` in the first period in which it is observed

Examples

```
n <- 50
id <- rep(seq_len(n), each = 4)
t <- rep(1:4, n)
y <- rnorm(n * 4)
g <- rep(sample(c(0, 2, 3), n, replace = TRUE), each = 4)
dta <- data.frame(id = id, t = t, y = y, group = g)
dta$Yi1 <- get_Yi1(dta, idname = "id", yname = "y", tname = "t", gname = "group")
```

`get_Yibar` *get_Yibar*

Description

A function to calculate the average outcome across all time periods separately for each unit in a panel data setting (this function can also be used to recover covariates, etc.).

Usage

```
get_Yibar(df, idname, yname)
```

Arguments

`df`	the data.frame used in the function
`idname`	name of column that holds the unit id
`yname`	name of column containing the outcome (or other variable) for which to calculate its outcome in the immediate pre-treatment period

Value

a numeric vector with one element per row of `df`: the unit's average of `yname` over all of its periods

Examples

```
n <- 50
id <- rep(seq_len(n), each = 4)
t <- rep(1:4, n)
y <- rnorm(n * 4)
dta <- data.frame(id = id, t = t, y = y)
dta$Yibar <- get_Yibar(dta, idname = "id", yname = "y")
```

get_Yibar_pre *get_Yibar_pre*

Description

A function to calculate average outcomes for units in their pre-treatment periods (this function can also be used to recover pre-treatment averages of covariates, etc.). For units that do not participate in the treatment (and therefore have group==0), the function calculates their overall average outcome.

Usage

```
get_Yibar_pre(df, idname, yname, tname, gname)
```

Arguments

df	the data.frame used in the function
idname	name of column that holds the unit id
yname	name of column containing the outcome (or other variable) for which to calculate its outcome in the immediate pre-treatment period
tname	name of column that holds the time period
gname	name of column containing the unit's group

Value

a numeric vector with one element per row of df: the unit's average of yname over periods before its group (over all periods for never-treated units, NaN for treated units with no pre-treatment periods)

Examples

```
n <- 50
id <- rep(seq_len(n), each = 4)
t <- rep(1:4, n)
y <- rnorm(n * 4)
g <- rep(sample(c(0, 2, 3), n, replace = TRUE), each = 4)
dta <- data.frame(id = id, t = t, y = y, group = g)
dta$Yibarpre <- get_Yibar_pre(dta, idname = "id", yname = "y",
                              tname = "t", gname = "group")
```

get_YiGmin1 *get_YiGmin1*

Description

A function to calculate outcomes for units in the period right before they become treated (this function can also be used to recover covariates, etc. in the period right before a unit becomes treated). For units that do not participate in the treatment (and therefore have group==0), they are assigned their outcome in the last period.

Usage

```
get_YiGmin1(df, idname, yname, tname, gname)
```

Arguments

df	the data.frame used in the function
idname	name of column that holds the unit id
yname	name of column containing the outcome (or other variable) for which to calculate its outcome in the immediate pre-treatment period
tname	name of column that holds the time period
gname	name of column containing the unit's group

Value

a vector with one element per row of df: the unit's value of yname in period group - 1 (in its last period if group is 0), or NA if the unit is not observed in that period

Examples

```
n <- 50
id <- rep(seq_len(n), each = 4)
t <- rep(1:4, n)
y <- rnorm(n * 4)
g <- rep(sample(c(0, 2, 3), n, replace = TRUE), each = 4)
dta <- data.frame(id = id, t = t, y = y, group = g)
dta$YiGmin1 <- get_YiGmin1(dta, idname = "id", yname = "y",
                           tname = "t", gname = "group")
head(unique(dta[, c("id", "group", "YiGmin1")]))
```

get_Yit *get_Yit*

Description

A function to calculate outcomes for units in a particular time period `tp` in a panel data setting (this function can also be used to recover covariates, etc. in the first period).

Usage

```
get_Yit(df, tp, idname, yname, tname)
```

Arguments

`df`	the data.frame used in the function
`tp`	The time period for which to get the outcome
`idname`	name of column that holds the unit id
`yname`	name of column containing the outcome (or other variable) for which to calculate its outcome in the immediate pre-treatment period
`tname`	name of column that holds the time period

Value

a vector of outcomes in period t, the vector will have the length nT (i.e., this is returned for each element in the panel, not for a particular period); `NA` for units not observed in period `tp`

Examples

```
n <- 50
id <- rep(seq_len(n), each = 4)
t <- rep(1:4, n)
y <- rnorm(n * 4)
dta <- data.frame(id = id, t = t, y = y)
Yit2 <- get_Yit(dta, tp = 2, idname = "id", yname = "y", tname = "t")
length(Yit2) ## n * 4
```

ids2rownum *Convert Vector of ids into Vector of Row Numbers*

Description

ids2rownum takes a vector of ids and converts it to the right row number in the dataset; ids should be unique in the dataset that is, don't pass the function panel data with multiple same ids

Usage

```
ids2rownum(ids, data, idname)
```

Arguments

ids	vector of ids
data	data frame
idname	unique id

Value

vector of row numbers

Examples

```
ids <- seq(1, 1000, length.out = 100)
ids <- ids[order(runif(100))]
df <- data.frame(id = ids)
ids2rownum(df$id, df, "id")
```

invert_ecdf *Invert Ecdf*

Description

take an ecdf object and invert it to get a step-quantile function

Usage

```
invert_ecdf(df)
```

Arguments

df	an ecdf object

Value

stepfun object that contains the quantiles of the df

Examples

```
y <- rnorm(100)
F <- ecdf(y)
Finv <- invert_ecdf(F)
Finv(0.5) ## approximate median
```

---

lhs_vars *Left-hand Side Variables*

Description

Take a formula and return a vector of the variables on the left hand side, it will return NULL for a one sided formula

Usage

```
lhs_vars(formula)
```

Arguments

formula	a formula

Value

vector of variable names

Examples

```
ff <- yvar ~ x1 + x2
lhs_vars(ff)
```

make_balanced_panel *Balance a Panel Data Set*

Description

This function drops observations from data.frame that are not part of balanced panel data set. `data` is copied before dropping any rows, so the object passed in is left unmodified; see `set_balanced_panel` for an in-place alternative that avoids copying large inputs.

Usage

```
make_balanced_panel(data, idname, tname)
```

Arguments

`data`	data.frame (or data.table) used in function
`idname`	unique id
`tname`	time period name

Value

a balanced panel, with the same class (`data.frame` or `data.table`) as `data`

Examples

```
id <- rep(seq(1, 100), each = 2) # individual ids for setting up a two period panel
t <- rep(seq(1, 2), 100) # time periods
y <- rnorm(200) # outcomes
dta <- data.frame(id = id, t = t, y = y) # make into data frame
dta <- dta[-7, ] # drop the 7th row from the dataset (which creates an unbalanced panel)
dta <- make_balanced_panel(dta, idname = "id", tname = "t")
```

make_dist *Make a Distribution Function*

Description

turn vectors of a values and their distribution function values into an ecdf. Vectors should be the same length and both increasing.

Usage

```
make_dist(
  x,
  Fx,
  sorted = FALSE,
  rearrange = FALSE,
  force01 = FALSE,
  method = "constant"
)
```

Arguments

x	vector of values
Fx	vector of the distribution function values
sorted	boolean indicating whether or not x is already sorted; computation is somewhat faster if already sorted
rearrange	boolean indicating whether or not should monotize distribution function
force01	boolean indicating whether or not to force the values of the distribution function (i.e. Fx) to be between 0 and 1
method	which method to pass to approxfun to approximate the distribution function. Default is "constant"; other possible choice is "linear". "constant" returns a step function, just like an empirical cdf; "linear" linearly interpolates between neighboring points.

Value

ecdf

Examples

```
y <- rnorm(100)
y <- y[order(y)]
u <- runif(100)
u <- u[order(u)]
F <- make_dist(y, u)
```

---

multiplier_bootstrap *multiplier_bootstrap*

Description

A function that takes in an influence function (an nxk matrix) and the number of bootstrap iterations and returns a Bxk matrix of bootstrap results. This function uses Rademacher weights.

Usage

```
multiplier_bootstrap(inf_func, biters, method = c("auto", "gemm", "loop"))
```

Arguments

inf_func	nxk matrix of (e.g., these could be a matrix containing the influence function for different parameter estimates)
biters	the number of bootstrap iterations
method	one of "auto" (the default), "gemm" to always use the blocked matrix multiplication, or "loop" to always accumulate one draw at a time.

Details

The influence function is read once, in blocks of rows, and each block is combined with all biters draws at once. With an optimized BLAS (Accelerate, OpenBLAS, MKL, BLIS) that is a single matrix multiplication per block, which is much faster. With R's reference BLAS it is not, and the draws are accumulated one at a time within each block instead. method = "auto" picks between the two from extSoftVersion()[["BLAS"]]. Both give the same draws for a given seed (up to floating-point rounding), so the choice only affects speed.

Value

a Bxk matrix

---

mv_mult *Matrix-Vector Multiplication*

Description

This function multiplies a matrix by a vector and returns a numeric vector.

Usage

```
mv_mult(A, v)
```

Arguments

A	an nxk matrix.
v	a vector (can be stored as numeric or as a kx1 matrix)

Value

A numeric vector resulting from the multiplication of the matrix by the vector.

Examples

```
A <- matrix(1:9, nrow = 3, ncol = 3)
v <- c(2, 4, 6)
mv_mult(A, v)
```

orig2t *orig2t*

Description

A helper function to switch from original time periods to "new" time periods (which are just time periods going from 1 to total number of available periods). This allows for periods not being exactly spaced apart by 1.

Usage

```
orig2t(orig, original_time.periods)
```

Arguments

orig	a vector of original time periods to convert to new time periods.
original_time.periods	vector containing all original time periods.

Value

new time period converted from original time period; NA for values of orig that are not among original_time.periods (or 0).

Examples

```
original_time.periods <- c(2001, 2003, 2005, 2007)
orig2t(c(2001, 2005), original_time.periods) ## returns c(1, 3)
```

panel2cs *Panel Data to Repeated Cross Sections*

Description

panel2cs takes a 2 period dataset and turns it into a cross sectional dataset. The data includes the change in time varying variables between the time periods. The default functionality is to keep all the variables from period 1 and add all the variables listed by name in timevars from period 2 to those.

Usage

```
panel2cs(data, timevars, idname, tname)
```

Arguments

data	data.frame used in function
timevars	vector of names of variables to keep
idname	unique id
tname	time period name

Value

data.frame

Examples

```
id <- rep(seq(1, 50), 2)
t <- rep(seq(1, 2), each = 50)
y <- rnorm(100)
dta <- data.frame(id = id, t = t, y = y)
out <- panel2cs(dta, timevars = "y", idname = "id", tname = "t")
nrow(out)
```

---

panel2cs2 *Panel Data to Repeated Cross Sections*

Description

panel2cs2 takes a 2 period dataset and turns it into a cross sectional dataset; i.e., long to wide. This function considers a particular case where there is some outcome whose value can change over time. It returns the dataset from the first period with the outcome in the second period and the change in outcomes over time appended to it

Usage

```
panel2cs2(data, yname, idname, tname, balance_panel = TRUE)
```

Arguments

data	data.frame used in function
yname	name of outcome variable that can change over time
idname	unique id
tname	time period name
balance_panel	whether to ensure that panel is balanced. Default is TRUE, but code runs somewhat faster if this is set to be FALSE.

Value

data from first period with .y0 (outcome in first period), .y1 (outcome in second period), and .dy (change in outcomes over time) appended to it

Examples

```
id <- rep(seq(1, 50), 2)
t <- rep(seq(1, 2), each = 50)
y <- rnorm(100)
dta <- data.frame(id = id, t = t, y = y)
out <- panel2cs2(dta, yname = "y", idname = "id", tname = "t")
head(out[, c("id", ".y0", ".y1", ".dy")])
```

`quantile.BMisc_dist` *Quantiles of a Distribution Function*

Description

Quantiles of a distribution function created by `make_dist` (and so by `weighted_ecdf` and `combine_ecdfs`), computed by inverting it directly: for a step function, the smallest support point y with $F(y) \geq p$ (the generalized inverse, `type = 1` in `quantile`); for `method = "linear"`, the inverse of the piecewise-linear F.

Sample-quantile types other than 1 interpolate between order statistics of a sample, which a weighted distribution function doesn't have, so `type` is ignored (with a warning if it isn't 1).

Usage

```
## S3 method for class 'BMisc_dist'
quantile(x, probs = seq(0, 1, 0.25), names = TRUE, ...)
```

Arguments

`x`	a distribution function from `make_dist`.
`probs`	numeric vector of probabilities in $[0, 1]$.
`names`	logical; if `TRUE`, name the result by `probs`, as `quantile` does.
`...`	ignored, except `type` (see Description).

Value

numeric vector of quantiles.

Examples

```
F <- weighted_ecdf(c(0, 1), weights = c(0.4, 0.6))
quantile(F, 0.5) # 1
```

rhs *Right-hand Side of Formula*

Description

Take a formula and return the right hand side of the formula

Usage

```
rhs(formula)
```

Arguments

formula	a formula

Value

a one sided formula

Examples

```
ff <- yvar ~ x1 + x2
rhs(ff)
```

rhs_vars *Right-hand Side Variables*

Description

Take a formula and return a vector of the variables on the right hand side

Usage

```
rhs_vars(formula)
```

Arguments

formula	a formula

Value

vector of variable names

Examples

```
ff <- yvar ~ x1 + x2
rhs_vars(ff)

ff <- y ~ x1 + I(x1^2)
rhs_vars(ff)
```

`set_balanced_panel` *set_balanced_panel*

Description

In-place version of `make_balanced_panel` for large panels where copying `data` is undesirable. Converts `data` to a data.table by reference (mutating the caller's object into a data.table as a side effect, if it is not one already) instead of copying it. The balanced result must still be captured from the return value, the same as any other function in this package (filtering out unbalanced units cannot itself happen without allocating the smaller, filtered result somewhere).

Usage

```
set_balanced_panel(data, idname, tname)
```

Arguments

`data`	data.frame (or data.table) used in function
`idname`	unique id
`tname`	time period name

Value

a balanced data.table

Examples

```
id <- rep(seq(1, 100), each = 2)
t <- rep(seq(1, 2), 100)
y <- rnorm(200)
dta <- data.frame(id = id, t = t, y = y)
dta <- dta[-7, ]
dta <- set_balanced_panel(dta, idname = "id", tname = "t") # dta is now a data.table
```

sort_panel *sort_panel*

Description

Sorts a panel data set by unit id and then time period. `data` is copied before sorting, so the object passed in is left unmodified. For large panels where copying is undesirable, sort in place instead with `data.table::setorderv(data, c(idname, tname))`.

Usage

```
sort_panel(data, idname, tname)
```

Arguments

`data`	data.frame (or data.table) used in function
`idname`	unique id
`tname`	time period name

Value

`data` sorted by `(idname, tname)`, with the same class (`data.frame` or `data.table`) as `data`

Examples

```
id <- rep(sample(1:5), each = 2) # units in a shuffled, unsorted order
t <- rep(c(2, 1), 5) # each unit's own periods out of time order
dta <- data.frame(id = id, t = t, y = rnorm(10))
dta <- sort_panel(dta, idname = "id", tname = "t")
```

source_all *source_all*

Description

Source all the files in a folder

Usage

```
source_all(fldr)
```

Arguments

`fldr`	path to a folder

subsample *Subsample of Observations from Panel Data*

Description

returns a subsample of a panel data set; in particular drops all observations that are not in `keepids`. If it is not set, randomly keeps `nkeep` observations.

Usage

```
subsample(dta, idname, tname, keepids = NULL, nkeep = NULL)
```

Arguments

`dta`	a data.frame which is a balanced panel
`idname`	the name of the id variable
`tname`	the name of the time variable
`keepids`	which ids to keep
`nkeep`	how many ids to keep (only used if `keepids` is not set); the default is the number of unique ids

Value

a data.frame that contains a subsample of `dta`

Examples

```
data("LaborSupply", package = "plm")
nrow(LaborSupply)
unique(LaborSupply$year)
ss <- subsample(LaborSupply, "id", "year", nkeep = 100)
nrow(ss)
```

t2orig *t2orig*

Description

A helper function to switch from "new" t values to original t values. This allows for periods not being exactly spaced apart by 1.

Usage

```
t2orig(t, original_time.periods)
```

Arguments

t	a vector of time periods to convert back to original time periods.
original_time.periods	vector containing all original time periods.

Value

original time period converted from new time period; NA for values of t that do not correspond to a time period.

Examples

```
original_time.periods <- c(2001, 2003, 2005, 2007)
t2orig(1:4, original_time.periods) ## returns c(2001, 2003, 2005, 2007)
```

---

time_invariant_to_panel

time_invariant_to_panel

Description

This function takes a time-invariant variable and repeats it for each period in a panel data set.

Usage

```
time_invariant_to_panel(x, df, idname, balanced_panel = TRUE)
```

Arguments

x	a vector of length equal to the number of unique ids in df.
df	the data.frame used in the function
idname	name of column that holds the unit id
balanced_panel	a logical indicating whether the panel is balanced. If TRUE, the function will optimize the repetition process. Default is TRUE.

Value

a vector of length equal to the number of rows in df.

Examples

```
n <- 50
id <- rep(seq_len(n), each = 4)
t <- rep(1:4, n)
dta <- data.frame(id = id, t = t)
x_unit <- rnorm(n)
x_panel <- time_invariant_to_panel(x_unit, dta, idname = "id")
length(x_panel) ## n * 4
```

`toformula` *Variable Names to Formula*

Description

take a name for a y variable and a vector of names for x variables and turn them into a formula

Usage

```
toformula(yname, xnames)
```

Arguments

`yname`	the name of the y variable
`xnames`	vector of names for x variables

Value

a formula

Examples

```
toformula("yvar", c("x1", "x2"))

## should return yvar ~ 1
toformula("yvar", rhs_vars(~1))
```

TorF *TorF*

Description

A function to replace NA's with FALSE in vector of logicals

Usage

```
TorF(cond, use_isTRUE = FALSE)
```

Arguments

cond	a vector of conditions to check
use_isTRUE	whether or not to use a vectorized version of isTRUE. This is generally slower but covers more cases.

Value

logical vector

Examples

```
TorF(c(TRUE, NA, FALSE)) ## NA becomes FALSE
```

weighted_checkfun *Weighted Check Function*

Description

Weights the check function

Usage

```
weighted_checkfun(q, cvec, tau, weights)
```

Arguments

q	the value to check
cvec	vector of data to compute quantiles for
tau	between 0 and 1, ex. .5 implies get the median
weights	the weights, weighted.checkfun normalizes the weights to sum to 1.

Value

numeric

Examples

```
x <- rnorm(100)
weighted_checkfun(0, x, tau = 0.5, weights = rep(1, 100))
```

`weighted_combine_list` *weighted_combine_list*

Description

A function that takes in either a list of vectors or matrices and computes a weighted average of them, where the weights are applied to every element in the list.

Usage

```
weighted_combine_list(l, w, normalize_weights = TRUE)
```

Arguments

`l`	a list that contains either vectors or matrices of the same dimension that are to be combined
`w`	a vector of weights, the weights should have the same number of elements as `length(l)`
`normalize_weights`	whether or not to force the weights to sum to 1, default is true

Value

matrix or vector corresponding to the weighted average of all of the elements in `l`

Examples

```
l <- list(c(1, 2, 3), c(4, 5, 6))
weighted_combine_list(l, w = c(0.5, 0.5)) ## returns c(2.5, 3.5, 4.5)
```

weighted_ecdf *Weighted Distribution Function*

Description

Get a distribution function from a vector of values after applying some weights

Usage

```
weighted_ecdf(y, y.seq = NULL, weights = NULL, norm = TRUE)
```

Arguments

y	a vector to compute the mean for
y.seq	an optional vector of values to compute the distribution function for; the default is to use all unique values of y
weights	the vector of weights, can be NULL, then will just return mean
norm	normalize the weights so that they have mean of 1, default is to normalize

Value

ecdf

Examples

```
y <- rnorm(100)
w <- runif(100)
F <- weighted_ecdf(y, weights = w)
F(0) ## approx 0.5
```

weighted_mean *Weighted Mean*

Description

Get the mean applying some weights

Usage

```
weighted_mean(y, weights = NULL, norm = TRUE)
```

Arguments

y	a vector to compute the mean for
weights	the vector of weights, can be NULL, then will just return mean
norm	normalize the weights so that they have mean of 1, default is to normalize

Value

the weighted mean

Examples

```
y <- rnorm(100)
w <- runif(100)
weighted_mean(y, weights = w)
```

---

weighted_quantile *weighted_quantile*

Description

function to recover quantiles of a vector with weights

Usage

```
weighted_quantile(tau, cvec, weights = NULL, norm = TRUE)
```

Arguments

tau	a vector of values between 0 and 1
cvec	a vector to compute quantiles for
weights	the weights, weighted.checkfun normalizes the weights to sum to 1.
norm	normalize the weights so that they have mean of 1, default is to normalize

Value

vector of quantiles

Examples

```
y <- rnorm(100)
w <- runif(100)
weighted_quantile(c(0.25, 0.5, 0.75), y, weights = w)
```

Index

add_cov_to_formula, 3

block_boot_sample, 4

check_staggered, 4
checkfun, 5
combine_ecdfs, 6, *26*
compare_binary, 7
cs2panel, 8

drop_collinear, 8
drop_cov_from_formula, 9

element_wise_mult, 10

get_first_difference, 10
get_group, 11
get_lagYi, 12
get_list_element, 13
get_principal_components, 13
get_Yi1, 14
get_Yibar, 15
get_Yibar_pre, 16
get_YiGmin1, 17
get_Yit, 18

ids2rownum, 19
invert_ecdf, 19

lhs_vars, 20

make_balanced_panel, 21, *28*
make_dist, 21, *26*
make_dist(), *6*
multiplier_bootstrap, 22
mv_mult, 23

orig2t, 24

panel2cs, 24
panel2cs2, 25

quantile, *26*
quantile.BMisc_dist, 26

rhs, 27
rhs_vars, 27

set_balanced_panel, *21*, 28
sort_panel, 29
source_all, 29
stats::ecdf(), *6*
subsample, 30

t2orig, 30
time_invariant_to_panel, 31
toformula, 32
TorF, 33

weighted_checkfun, 33
weighted_combine_list, 34
weighted_ecdf, *26*, 35
weighted_ecdf(), *6*
weighted_mean, 35
weighted_quantile, 36