

# Package: BMisc (via r-universe)

June 11, 2026

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

**Version** 1.4.9

**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, dplyr, Rcpp, tidyr

**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** 8.0.0  
**NeedsCompilation** yes  
**Author** Brantly Callaway [aut, cre]  
**Maintainer** Brantly Callaway <brantly.callaway@uga.edu>  
**Repository** https://cran.r-universe.dev  
**Date/Publication** 2026-06-11 11:51:39 UTC  
**RemoteUrl** https://github.com/cran/BMisc  
**RemoteRef** HEAD  
**RemoteSha** 1231f8c407c3e633e7eb927690493a216c96d02f

## Contents

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

`add_cov_to_formula` 3

<code>subsample</code> . . . . .	25
<code>t2orig</code> . . . . .	26
<code>time_invariant_to_panel</code> . . . . .	27
<code>toformula</code> . . . . .	27
<code>TorF</code> . . . . .	28
<code>weighted_checkfun</code> . . . . .	29
<code>weighted_combine_list</code> . . . . .	29
<code>weighted_ecdf</code> . . . . .	30
<code>weighted_mean</code> . . . . .	31
<code>weighted_quantile</code> . . . . .	31

**Index** 33

---

`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	<i>Block Bootstrap</i>
-------------------	------------------------

---

**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	<i>check_staggered</i>
-----------------	------------------------

---

**Description**

A function to check if treatment is staggered in a panel data set.

**Usage**

```
check_staggered(df, idname, treatname)
```

**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

**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")
```

---

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	<i>Combine Two Distribution Functions</i>
---------------	---

---

**Description**

Combines two distribution functions with given weights by ‘weights’

**Usage**

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

**Arguments**

y.seq	sequence of possible y values
ecdflist	list of ecdf objects (distribution functions) to combine
weights	a vector of weights to put on each distribution function; if weights are not provided then equal weight is given to each distribution function
...	additional arguments that can be past to BMisc::make_dist

**Value**

ecdf

**Examples**

```
x <- rnorm(100)
y <- rnorm(100, 1, 1)
Fx <- ecdf(x)
Fy <- ecdf(y)
both <- combine_ecdfs(seq(-2, 3, 0.1), list(Fx, Fy))
plot(Fx, col = "green")
plot(Fy, col = "blue", add = TRUE)
plot(both, add = TRUE)
```

---

compare_binary	<i>Compare Variables across Groups</i>
----------------	--

---

**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  $n \times B$  and  $n \times k$  and returns a  $B \times k$  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 Rademacher weights for B bootstrap iterations using the multiplier bootstrap)
inf_func	nxB matrix of (e.g., these could be a matrix containing the influence function for different parameter estimates)

**Value**

a BxB 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 nxB rows, the resulting vector will also have nxB 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

**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	<i>get_group</i>
-----------	------------------

---

**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

**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	<i>get_lagYi</i>
-----------	------------------

---

**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.

**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(listlists, whichone = 1)
```

**Arguments**

listlists	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 = (-1), y = 1^2)) # 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 $n = \text{length}(\text{unique}(\text{data}[[\text{idname}]])$ ) or long format (where the number of rows is equal to $nT = \text{nrow}(\text{data})$ ). 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

### 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)
```



---

<code>get_YiGmin1</code>	<i>get_YiGmin1</i>
--------------------------	--------------------

---

**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**

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

**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")]))
```

---

<code>get_Yit</code>	<i>get_Yit</i>
----------------------	----------------

---

**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)

**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.

**Usage**

```
make_balanced_panel(data, idname, tname, return_data.table = FALSE)
```

**Arguments**

data                data.frame used in function

idname             unique id

tname              time period name

return\_data.table    if TRUE, make\_balanced\_panel will return a data.table rather than a data.frame.  
Default is FALSE.

**Value**

data.frame that is a balanced panel

**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**

<code>x</code>	vector of values
<code>Fx</code>	vector of the distribution function values
<code>sorted</code>	boolean indicating whether or not <code>x</code> is already sorted; computation is somewhat faster if already sorted
<code>rearrange</code>	boolean indicating whether or not should monotize distribution function
<code>force01</code>	boolean indicating whether or not to force the values of the distribution function (i.e. <code>Fx</code> ) to be between 0 and 1
<code>method</code>	which method to pass to <code>approxfun</code> 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  $n \times k$  matrix) and the number of bootstrap iterations and returns a  $B \times k$  matrix of bootstrap results. This function uses Rademehar weights.

**Usage**

```
multiplier_bootstrap(inf_func, biters)
```

**Arguments**

inf_func	$n \times k$ matrix of (e.g., these could be a matrix containing the influence function for different parameter estimates)
biters	the number of bootstrap iterations

**Value**

a  $B \times k$  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 $n \times k$ matrix.
v	a vector (can be stored as numeric or as a $k \times 1$ 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	<i>orig2t</i>
--------	---------------

---

**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

**Examples**

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

---

panel2cs	<i>Panel Data to Repeated Cross Sections</i>
----------	--

---

**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")])
```

---

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)
```

---

source_all	<i>source_all</i>
------------	-------------------

---

**Description**

Source all the files in a folder

**Usage**

```
source_all(fldr)
```

**Arguments**

fldr                path to a folder

---

subsample	<i>Subsample of Observations from Panel Data</i>
-----------	--

---

**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

**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	<i>Weighted Mean</i>
---------------	----------------------

---

**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	<i>weighted_quantile</i>
-------------------	--------------------------

---

**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  
[compare\\_binary](#), 6  
[cs2panel](#), 7

[drop\\_collinear](#), 8  
[drop\\_cov\\_from\\_formula](#), 9

[element\\_wise\\_mult](#), 9

[get\\_first\\_difference](#), 10  
[get\\_group](#), 11  
[get\\_lagYi](#), 11  
[get\\_list\\_element](#), 12  
[get\\_principal\\_components](#), 13  
[get\\_Yi1](#), 14  
[get\\_Yibar](#), 14  
[get\\_Yibar\\_pre](#), 15  
[get\\_YiGmin1](#), 16  
[get\\_Yit](#), 16

[ids2rownum](#), 17  
[invert\\_ecdf](#), 18

[lhs\\_vars](#), 18

[make\\_balanced\\_panel](#), 19  
[make\\_dist](#), 20  
[multiplier\\_bootstrap](#), 21  
[mv\\_mult](#), 21

[orig2t](#), 22

[panel2cs](#), 22  
[panel2cs2](#), 23

[rhs](#), 24

[rhs\\_vars](#), 24

[source\\_all](#), 25  
[subsample](#), 25

[t2orig](#), 26  
[time\\_invariant\\_to\\_panel](#), 27  
[toformula](#), 27  
[TorF](#), 28

[weighted\\_checkfun](#), 29  
[weighted\\_combine\\_list](#), 29  
[weighted\\_ecdf](#), 30  
[weighted\\_mean](#), 31  
[weighted\\_quantile](#), 31