Package 'propertee'

November 13, 2026

Version 1.0.7

Title Standardization-Based Effect Estimation with Optional Prior Covariance Adjustment

Description The Prognostic Regression Offsets with Propagation of ERrors (for Treatment Effect Estimation) package facilitates direct adjustment for experiments and observational studies that is compatible with a range of study designs and covariance adjustment strategies. It uses explicit specification of clusters, blocks and treatment allocations to furnish probability of assignment-based weights targeting any of several average treatment effect parameters, and for standard error calculations reflecting these design parameters. For covariance adjustment of its Hajek and (one-way) fixed effects estimates, it enables offsetting the outcome against predictions from a dedicated covariance model, with standard error calculations propagating error as appropriate from the covariance model.

License MIT + file LICENSE

License_is_FOSS yes

License_restricts_use no

Encoding UTF-8

LazyData true

Suggests knitr, rmarkdown, testthat (>= 3.0.0), multcomp, MASS

Config/testthat/edition 3

Imports stats, methods, sandwich

Enhances robustbase

Depends R (>= 4.1.0)

VignetteBuilder knitr

URL https://github.com/benbhansen-stats/propertee

BugReports https://github.com/benbhansen-stats/propertee/issues

Collate 'StudySpecification.R' 'StudySpecificationAccessors.R'
'SandwichLayer.R' 'block_center_residuals.R'
'SandwichLayerVariance.R' 'StudySpecificationConverters.R'
'StudySpecificationStructure.R' 'StudySpecificationUtilities.R'
'WeightedStudySpecification.R' 'areg.center.R' 'confint_lm.R'
'teeMod.R' 'as.lmitt.R' 'as_data_frame.R' 'assigned.R'
'bread.R' 'c_weightedStudySpecification.R' 'cov_adj.R' 'data.R'
'dichotomy.R' 'expand.model.frame_tee.R' 'get_cov_adj.R'
'get_data_from_model.R' 'get_spec.R' 'glmrobMethods.R'
'lmitt.R' 'lmrob_methods.R' 'merge_preserve_order.R'
'propertee_package.R' 'specification_table.R'
'summary.StudySpecification.R' 'summary.teeMod.R' 'update_by.R'
'validWeights.R' 'weights_internal.R' 'weights_exported.R'

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Josh Errickson [aut],
Josh Wasserman [cre, aut],
Mark Fredrickson [ctb],
Tim Lycurgus [ctb],
Adam Sales [ctb],
Xinhe Wang [ctb],
Ben Hansen [aut]

Maintainer Josh Wasserman <jwass@umich.edu>

Repository CRAN

Date/Publication 2026-11-13 09:10:15 UTC

Contents

+,WeightedStudySpecification,numeric-method . . . 3
as.lmitt . . . 4
as.SandwichLayer . . . 5
assigned . . . 6
as_rct_spec . . . 7
bread.teeMod . . . 8
c,WeightedStudySpecification-method . . . 9
confint.teeMod . . . 10
cov_adj . . . 11
default . . . 13
estfun.glmrob . . . 13
estfun.lmrob . . . 14
estfun.teeMod . . . 15
ett . . . 16
get_structure . . . 18
GV_data . . . 19
has_binary_treatment . . . 20
identical_StudySpecifications . . . 20

identify_small_blocks 21
lmitt 21
lsoSynth 24
michigan_school_pairs 25
rct_spec 26
schooldata 28
show,PreSandwichLayer-method 29
show,StudySpecification-method 29
show,teeMod-method 30
show,WeightedStudySpecification-method 30
simdata 31
specification_data_concordance 31
specification_table 32
STARplus 34
subset,PreSandwichLayer-method 36
subset,WeightedStudySpecification-method 37
summary.StudySpecification 37
summary.teeMod 38
treatment 39
unit_of_assignment 42
var_table 43
vcov.teeMod 44
weights,WeightedStudySpecification-method 45

Index **46**

`+,WeightedStudySpecification,numeric-method`
`WeightedStudySpecification` *Operations*

Description

Algebraic operators on `WeightedStudySpecification` objects and numeric vectors. `WeightedStudySpecifications` do not support addition or subtraction.

Usage

```
## S4 method for signature 'WeightedStudySpecification,numeric'
e1 + e2

## S4 method for signature 'numeric,WeightedStudySpecification'
e1 + e2

## S4 method for signature 'WeightedStudySpecification,numeric'
e1 - e2

## S4 method for signature 'numeric,WeightedStudySpecification'
e1 - e2
```

```
## S4 method for signature 'WeightedStudySpecification,numeric'
e1 * e2

## S4 method for signature 'numeric,WeightedStudySpecification'
e1 * e2

## S4 method for signature 'WeightedStudySpecification,numeric'
e1 / e2

## S4 method for signature 'numeric,WeightedStudySpecification'
e1 / e2
```

Arguments

e1, e2	WeightedStudySpecification or numeric objects

Details

These are primarily used to either combine weights via multiplication, or to invert weights. Addition and subtraction are not supported and will produce errors.

Value

a WeightedStudySpecification object

---

as.lmitt *Convert* lm *object into* teeMod

Description

Converts the output of lm() into a teeMod object, for standard errors that account for block and cluster information carried with the lm's weights, and/or an offset incorporating predictions of the outcome from a separate model.

Usage

```
as.lmitt(x, specification = NULL)

as.teeMod(x, specification = NULL)
```

Arguments

x	lm object with weights containing a WeightedStudySpecification, or an offset from cov_adj().

specification — Optional, explicitly specify the StudySpecification to be used. If the StudySpecification is specified elsewhere in x (e.g. passed as an argument to any of ate(), ett(), cov_adj() or assigned()) it will be found automatically and does not need to be passed here as well. If different StudySpecification objects are passed (either through the lm in weights or covariance adjustment, or through this argument), an error will be produced.

Details

The formula with which x was created must include a treatment identifier (e.g. assigned()). If a model-based offset is incorporated, the model's predictions would have to have been extracted using cov_adj() as opposed to predict{} in order for teeMod standard error calculations to reflect propagation of error from these predictions. This mechanism only supports treatment main effects: to estimate interactions of treatment assignment with a moderator variable, use lmitt() instead of lm() followed by as.lmitt().

Value

teeMod object

---

as.SandwichLayer — *Convert a PreSandwichLayer to a SandwichLayer with a StudySpecification object*

Description

as.SandwichLayer() uses the StudySpecification object passed to the specification argument to populate the slots in a SandwichLayer object that a PreSandwichLayer does not have sufficient information for.

Usage

```
as.SandwichLayer(x, specification, by = NULL, Q_data = NULL)
```

Arguments

x — a PreSandwichLayer object

specification — a StudySpecification object

by — optional; a string or named vector of unique identifier columns in the data used to create specification and the data used to fit the covariance adjustment model. Default is NULL, in which case unit of assignment columns are used for identification (even if they do not uniquely identify units of observation). If a named vector is provided, names should represent variables in the data used to create specification, while values should represent variables in the covariance adjustment data.

Q_data — dataframe of direct adjustment sample, which is needed to generate the keys slot of the SandwichLayer object. Defaults to NULL, in which case if by is NULL, the data used to create specification is used, and if by is not NULL, appropriate data further up the call stack (passed as arguments to cov_adj() or lmitt.formula(), for example) is used.

Value

a SandwichLayer object

---

assigned *Obtain Treatment from StudySpecification*

Description

When passing a lm object to lmitt(), extract and use the treatment variable specified in the StudySpecification.

Usage

```
assigned(specification = NULL, data = NULL, dichotomy = NULL)

adopters(specification = NULL, data = NULL, dichotomy = NULL)

a.(specification = NULL, data = NULL, dichotomy = NULL)

z.(specification = NULL, data = NULL, dichotomy = NULL)
```

Arguments

specification — Optional StudySpecification. If the StudySpecification can't be identified in the model (usually because neither weights (ate() or ett()) nor a covariate adjustment model (cov_adj()) are found), the StudySpecification can be passed diretly.

data — Optional data set. By default assigned() will attempt to identify the appropriate data, if this fails (or you want to overwrite it), you can pass the data here.

dichotomy — optional; a formula defining the dichotomy of the treatment variable if it isn't already 0/1. See details for more information. If ett() or ate() is called within a lmitt() call that specifies a dichotomy argument, that dichotomy will be used if the argument here has not been specified.

Details

When passing a lm object to lmitt(), the treatment variable in the formula passed to lm() needs to be identifiable. Rather than placing the treatment variable directly in the formula, use one of these functions, to allow lmitt() to identify the treatment variable.

To keep the formula in the `lm()` call concise, instead of passing `specification` and `data` arguments to these functions, one can pass a `WeightedStudySpecification` object to the `weights` argument of the `lm()` call or a `SandwichLayer` object to the `offset` argument.

Alternatively, you can pass the `specification` and `data` arguments.

While `assigned()` can be used in any situation, it is most useful for scenarios where the treatment variable is non-binary and the `StudySpecification` contains a `Dichotomy`. For example, say `q` is a 3-level ordinal treatment variable, and the binary comparison of interest is captured in `dichotomy = q == 3 ~ q < 3`. If you were to fit a model including `q` as a predictor, e.g. `lm(y ~ q, ...)`, `lm` would treat `q` as the full ordinal variable. On the other hand, by calling `lm(y ~ assigned(), weights = ate(spec), ...)`, `assigned()` will generate the appropriate binary variable to allow estimation of treatment effects.

If called outside of a model call and without a `data` argument, this will extract the treatment from the `specification`. If this is the goal, the `treatment()` function is better suited for this purpose.

Value

The treatment variable to be placed in the regression formula.

Examples

```
data(simdata)
spec <- obs_spec(z ~ uoa(uoa1, uoa2), data = simdata)
mod <- lm(y ~ assigned(), data = simdata, weights = ate(spec))
lmittmod <- lmitt(mod)
summary(lmittmod, vcov.type = "CR0")
```

---

as_rct_spec *Convert* StudySpecification *between types*

Description

Convert a `StudySpecification` between a observational study, a randomized control trial, and a regression discontinuity (created from `obs_spec`, `rct_spec` and `rd_spec` respectively).

Usage

```
as_rct_spec(StudySpecification, ..., loseforcing = FALSE)

as_obs_spec(StudySpecification, ..., loseforcing = FALSE)

as_rd_spec(StudySpecification, data, ..., forcing)
```

Arguments

StudySpecification	a StudySpecification to convert
...	Ignored.
loseforcing	converting from RD to another StudySpecification type will error to avoid losing the forcing variable. Setting loseforcing = TRUE allows the conversion to automatically drop the forcing variable. Default FALSE.
data	converting to an RD requires adding a forcing variable, which requires access to the original data.
forcing	converting to an RD requires adding a forcing variable. This should be entered as a formula which would be passed to update, e.g. forcing = . ~ . + forcing(forcevar).

Value

StudySpecification of the updated type

Examples

```
spec <- rct_spec(z ~ unit_of_assignment(uoa1, uoa2), data = simdata)
spec
as_obs_spec(spec)
as_rd_spec(spec, simdata, forcing = ~ . + forcing(force))
spec2 <- rd_spec(o ~ uoa(uoa1, uoa2) + forcing(force), data = simdata)
spec2
# as_rct_spec(spec2) # this will produce an error
as_rct_spec(spec2, loseforcing = TRUE)
```

---

bread.teeMod *Extract bread matrix from a* teeMod *model fit*

Description

An S3method for sandwich::bread that extracts the bread of the direct adjustment model sandwich covariance matrix.

Usage

```
## S3 method for class 'teeMod'
bread(x, values_from = x, ...)
```

Arguments

x	a fitted teeMod model
values_from	optional, a fitted model object. Defaults to x.
...	arguments passed to methods

Details

This function is a thin wrapper around `.get_tilde_a22_inverse()`.

Value

A variance-covariance matrix with row and column entries for the estimated coefficients in `x`, the marginal mean outcome in the control condition, the marginal mean `offset` in the control condition (if an `offset` is provided), and if a moderator variable is specified in the formula for `x`, the mean interaction in the control condition of the outcome and `offset` with the moderator variable

---

`c,WeightedStudySpecification-method` *Concatenate weights*

Description

Given several variations of weights generated from a single `StudySpecification`, combine into a single weight.

Usage

```
## S4 method for signature 'WeightedStudySpecification'
c(x, ..., warn_dichotomy_not_equal = FALSE)
```

Arguments

`x`	.. a `WeightedStudySpecification` object, typically created from `ate()` or `ett()`
`...`	any number of additional `WeightedStudySpecification` objects with equivalent `StudySpecification` to `x` and eachother
`warn_dichotomy_not_equal`	if `FALSE` (default), `WeightedStudySpecifications` are considered equivalent even if their `dichotomy` differs. If `TRUE`, a warning is produced.

Details

Concatenating `WeightedStudySpecification` objects with `c()` requires both individual `WeightedStudySpecification` objects to come from the same `StudySpecification` and have the same target (e.g all created with `ate()` or all created with `ett()`, no mixing-and-matching). All arguments to `c()` must be `WeightedStudySpecification`.

`WeightedStudySpecification` objects may be concatenated together even without having the same `@dichotomy` slot. This procedure only prompts a warning for differing dichotomies if the argument `warn_dichotomy_not_equal` is set to `TRUE`.

Value

A numeric `vector` with the weights concatenated in the input order.

Examples

```
data(simdata)
spec <- rct_spec(z ~ unit_of_assignment(uoa1, uoa2), data = simdata)
w1 <- ate(spec, data = simdata[1:30,])
w2 <- ate(spec, data = simdata[31:40,])
w3 <- ate(spec, data = simdata[41:50,])
c_w <- c(w1, w2, w3)
c(length(w1), length(w2), length(w3), length(c_w))

spec <- rct_spec(dose ~ unit_of_assignment(uoa1, uoa2), data = simdata)
w1 <- ate(spec, data = simdata[1:10, ], dichotomy = dose >= 300 ~ .)
w2 <- ate(spec, data = simdata[11:30, ], dichotomy = dose >= 200 ~ .)
w3 <- ate(spec, data = simdata[31:50, ], dichotomy = dose >= 100 ~ .)
c_w <- c(w1, w2, w3)
```

---

confint.teeMod *Confidence intervals with standard errors provided by* vcov.teeMod()

Description

An S3method for stats::confint that uses standard errors computed using vcov.teeMod(). Additional arguments passed to this function, such as cluster and type, specify the arguments of the vcov.teeMod() call.

Usage

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

Arguments

object	a fitted teeMod model
parm	a specification of which parameters are to be given confidence intervals, either a vector of numbers or a vector of names. If missing, all parameters are considered.
level	the confidence level required.
...	additional arguments to pass to vcov.teeMod()

Details

Rather than call stats::confint.lm(), confint.teeMod() calls .confint_lm(), a function internal to the propertee package that ensures additional arguments in the ... of the confint.teeMod() call are passed to the internal vcov() call.

Value

A matrix (or vector) with columns giving lower and upper confidence limits for each parameter. These will be labelled as (1-level)/2 and 1 - (1-level)/2 in % (by default 2.5% and 97.5%)

cov_adj *Prepare prognostic regression offset for* `lmitt()` *or* `lm()`

Description

`cov_adj()` takes a fitted model object and predicts outcomes under the counterfactual of being in the control condition. Using these predictions as offsets in a linear model with just the treatment variable as a regressor estimates the treatment effect with covariance adjustments as provided by the previously fitted model. In the special case that this covariance model was fit using only (quasi-)experimental control group observations, the method is known as Peters-Belson regression. `cov_adj()` assists the process by reporting its predictions on the scale of the response (in contrast to `predict.glm()`) and by retaining certain artifacts from covariance model fitting. These artifacts are used by `vcov_tee()`, and by `stats::vcov()` as applied to a `teeMod` object, so that the covariance model's contribution to sampling variability is reflected in treatment coefficient standard errors. `cov_adj()` can be used with `lm()`, but is better supported in combination with propertee's `lm`-wrapper, `lmitt()`.

Usage

```
cov_adj(
  model,
  newdata = NULL,
  specification = NULL,
  by = NULL,
  set_to_reference = NULL
)
```

Arguments

`model`	any model that inherits from a `glm`, `lm`, or `robustbase::lmrob` object.
`newdata`	a dataframe; optional. See Details for how `newdata` is found in the call stack if none is provided.
`specification`	a `StudySpecification` object; optional. If not provided, it can be retrieved from the call stack if passed to suitable calls.
`by`	named vector; optional.
`set_to_reference`	named list; optional.

Details

If `specification` is provided or can be found in the call stack, then prior to generating adjustments, `cov_adj()` will identify the treatment variable and replace all values with a reference level. If the treatment has logical type, this reference level is `FALSE`, and if it has numeric type, this is the smallest non-negative value (which means 0 for 0/1 binary). (Factor treatments are not currently supported.)

More broadly, the set_to_reference argument can be used to set columns to values specified by the user prior to generating predictions. The argument takes a named list where the names specify the columns whose values should be replaced, and the entries specify the replacement values. If an entry is given as default(), internal routines will determine the reference level (the first level of a factor/factorized character variable or the minimum of a numeric/logical variable).

If newdata is provided, the values of the output vector represent adjustments for the outcomes in newdata. If cov_adj() is called in the offset argument of lmitt(), predictions will correspond to rows in the data argument of lmitt(). Otherwise, adjustments are made based on the data to which model is fit.

The by argument specifies columns for merging datasets. The names of the argument specify columns in the dataframe for building specification, while the entries specify columns in the dataframe for fitting model.

Value

A SandwichLayer if specification is not NULL or a StudySpecification object is found in the call stack, otherwise a PreSandwichLayer object

Examples

```
data("STARplus")

##' A prognostic model fitted to experimental + non-experimental controls
y0hat_read <- lm(read_yr1 ~ gender*dob +dobNA + race,
                 data = STARplus,
                 subset = cond_at_entry!="small")

STARspec <- rct_spec(cond_at_entry ~ unit_of_assignment(stdntid) +
                         block(grade_at_entry, school_at_entry),
                     subset=!is.na(grade_at_entry),# excludes non-experimentals
                     data = STARplus)
ett_wts    <- ett(STARspec, data = STARplus,
                  dichotomy= cond_at_entry =="small" ~.)

ett_read <- lm(read_yr1 ~ assigned(dichotomy= cond_at_entry =="small" ~.),
               offset = cov_adj(y0hat_read),
               data = STARplus,
               weights = ett_wts)
coef(ett_read)
ett_read |> as.lmitt() # brings in control-group means of outcome, predictions

ate_read <- lmitt(read_yr1 ~ 1, STARspec, STARplus,
                  dichotomy= cond_at_entry =="small" ~.,
                  offset = cov_adj(y0hat_read),
                  weights = "ate")
show(ate_read)
vcov(ate_read, type = "HC0", cov_adj_rcorrect = "HC0") |> unname()

ate_read_loc <-
```

```
    lmitt(read_yr1 ~ race, STARspec, STARplus,
          dichotomy= cond_at_entry =="small" ~.,
          offset = cov_adj(y0hat_read, newdata = STARplus),
          weights = "ate")
show(ate_read_loc)
```

`default` *Instruct* `cov_adj()` *to find a default reference value for columns in a covariance adjustment model*

Description

Instruct `cov_adj()` to find a default reference value for columns in a covariance adjustment model

Usage

```
default()
```

Details

When set as the entry of a list passed to the `set_to_reference` argument of `cov_adj()`, `cov_adj()` uses the following replacement values depending on the column type:

- factor/character: minimum level
- numeric: minimum value
- logical: FALSE

Value

empty list that inherits from class `set_to_reference_default`

`estfun.glmrob` *Extract empirical estimating equations from a* `glmbrob` *model fit*

Description

Extract empirical estimating equations from a `glmbrob` model fit

Extract bread matrix from an `lmrob()` fit

Usage

```
## S3 method for class 'glmrob'
estfun(x, ...)

## S3 method for class 'glmrob'
bread(x, ...)
```

Arguments

x	a fitted lmrob object
...	arguments passed to methods

Value

matrix, estimating functions evaluated at data points and fitted parameters

matrix, inverse Hessian of loss as evaluated at fitted parameters

---

estfun.lmrob *Generate matrix of estimating equations for* lmrob() *fit*

Description

Generate matrix of estimating equations for lmrob() fit

Extract bread matrix from an lmrob() fit

Usage

```
## S3 method for class 'lmrob'
estfun(x, ...)

## S3 method for class 'lmrob'
bread(x, ...)
```

Arguments

x	An lmrob object produced using an MM/SM estimator chain
...	Additional arguments to be passed to bread

Details

This is part of a workaround for an issue in the robustbase code affecting sandwich covariance estimation. The issue in question is issue #6471, robustbase project on R-Forge. This function contributes to providing sandwich estimates of covariance-adjusted standard errors for robust linear covariance adjustment models.

This is part of a workaround for an issue in the robustbase code affecting sandwich covariance estimation. The issue in question is issue #6471, robustbase project on R-Forge. This function contributes to providing sandwich estimates of covariance-adjusted standard errors for robust linear covariance adjustment models.

Value

A $n\times$(p+1) matrix where the first column corresponds to the scale estimate and the remaining p colums correspond to the coefficients

A $p\times$(p+1) matrix where the first column corresponds to the scale estimate and the remaining p colums correspond to the coefficients

Author(s)

Ben B. Hansen

`estfun.teeMod` *Extract empirical estimating equations from a* `teeMod` *model fit*

Description

An S3method for `sandwich::estfun` for producing a matrix of contributions to the direct adjustment estimating equations.

Usage

```
## S3 method for class 'teeMod'
estfun(x, values_from = x, ...)
```

Arguments

`x`	a fitted `teeMod` model
`values_from`	optional, a fitted model object. Defaults to `x`.
`...`	arguments passed to methods, most importantly those that define the bias corrections for the residuals of `x` and, if applicable, a `fitted_covariance_model` stored in its offset

Details

If a prior covariance adjustment model has been passed to the `offset` argument of the `teeMod` model using `cov_adj()`, `estfun.teeMod()` incorporates contributions to the estimating equations of the covariance adjustment model.

The covariance adjustment sample may not fully overlap with the direct adjustment sample, in which case `estfun.teeMod()` returns a matrix with the same number of rows as the number of unique units of observation used to fit the two models. Uniqueness is determined by matching units of assignment used to fit the covariance adjustment model to units of assignment in the `teeMod` model's `StudySpecification` slot; units of observation within units of assignment that do not match are additional units that add to the row count.

The working residuals (and, if applicable, working weights) from `values_from` will be used in the output matrix in place of those from `x`. In other words, the contributions to the the empirical estimating equations remain the same but are evaluated at parameter estimates from `values_from` rather than `x`.

The by argument in `cov_adj()` can provide a column or a pair of columns (a named vector where the name specifies a column in the direct adjustment sample and the value a column in the covariance adjustment sample) that uniquely specifies units of observation in each sample. This information can be used to align each unit of observation's contributions to the two sets of estimating

equations. If no by argument is provided and units of observation cannot be uniquely specified, contributions are aligned up to the unit of assignment level. If standard errors are clustered no finer than that, they will provide the same result as if each unit of observation's contributions were aligned exactly.

This method incorporates bias corrections made to the residuals of `x` and, if applicable, the covariance model stored in its `offset`. When its crossproduct is taken (perhaps after suitable summing across rows within clusters), it provides a heteroskedasticity- (or cluster-) robust estimate of the meat matrix of the variance-covariance of the parameter estimates in `x`.

Value

An $n \times k$ matrix of empirical estimating equations for `x`. `k` includes the model intercept, main effects of treatment and moderator variables, any moderator effects, and marginal and conditional means of the outcome (and `offset`, if provided) in the control condition. See Details for definition of n.

`ett` *Generate Direct Adjusted Weights for Treatment Effect Estimation*

Description

These should primarily be used inside models. See Details.

Usage

```
ett(specification = NULL, dichotomy = NULL, by = NULL, data = NULL)

att(specification = NULL, dichotomy = NULL, by = NULL, data = NULL)

ate(specification = NULL, dichotomy = NULL, by = NULL, data = NULL)

etc(specification = NULL, dichotomy = NULL, by = NULL, data = NULL)

atc(specification = NULL, dichotomy = NULL, by = NULL, data = NULL)

ato(specification = NULL, dichotomy = NULL, by = NULL, data = NULL)

olw(specification = NULL, dichotomy = NULL, by = NULL, data = NULL)

owt(specification = NULL, dichotomy = NULL, by = NULL, data = NULL)

pwt(specification = NULL, dichotomy = NULL, by = NULL, data = NULL)
```

Arguments

specification	optional; a StudySpecification object created by one of rct_spec(), rd_spec(), or obs_spec().
dichotomy	optional; a formula defining the dichotomy of the treatment variable if it isn't already 0/1. See details for more information. If ett() or ate() is called within a lmitt() call that specifies a dichotomy argument, that dichotomy will be used if the argument here has not been specified.
by	optional; named vector or list connecting names of unit of assignment/ variables in specification to unit of assignment/unitid/cluster variables in data. Names represent variables in the StudySpecification; values represent variables in the data. Only needed if variable names differ.
data	optional; the data for the analysis to be performed on. May be excluded if these functions are included as the weights argument of a model.

Details

These functions should primarily be used in the weight argument of lmitt() orlm(). All arguments are optional if used within those functions. If used on their own, specification and data must be provided.

- ate - Average treatment effect. Aliases: ate().
- ett - Effect of treatment on the treated. Aliases: ett(), att().
- etc - Effect of treatment on controls. Aliases: etc(), atc().
- ato - Overlap-weighted average effect. Aliases: ato(), olw, owt, pwt.

In a StudySpecification with blocks, the weights are generated as a function of the ratio of the number of treated units in a block versus the total number of units in a block.

In any blocks where that ratio is 0 or 1 (that is, all units in the block have the same treatment status), the weights will be 0. In effect this removes from the target population any block in which there is no basis for estimating either means under treatment or means under control.

If block is missing for a given observation, a weight of 0 is applied.

A dichotomy is specified by a formula consisting of a conditional statement on both the left-hand side (identifying treatment levels associated with "treatment") and the right hand side (identifying treatment levels associated with "control"). For example, if your treatment variable was called dose and doses above 250 are considered treatment, you might write:

```
ate(..., dichotomy = dose > 250 ~ dose <= 250
```

The period (.) can be used to assign all other units of assignment. For example, we could have written the same treatment regime as either

```
etc(..., dichotomy = dose > 250 ~ .
```

or

```
olw(..., dichotomy = . ~ dose <= 250
```

The dichotomy formula supports Relational Operators (see Comparison), Logical Operators (see Logic), and %in% (see match()).

The conditionals need not assign all values of treatment to control or treatment, for example, `dose > 300 ~ dose < 200` does not assign `200 <= dose <= 300` to either treatment or control. This would be equivalent to manually generating a binary variable with `NA` whenever `dose` is between 200 and 300. Standard errors will reflect the sizes of the comparison groups specified by the `dichotomy`.

Tim Lycurgus contributed code for the computation of weights. The 'overlap weight' concept is due to Li, Morgan and Zaslavsky (2018), although the current implementation differs from that discussed in their paper in that it avoids estimated propensity scores.

Value

a `WeightedStudySpecification` object, which is a vector of numeric weights

References

Li, Fan, Kari Lock Morgan, and Alan M. Zaslavsky. "Balancing covariates via propensity score weighting." Journal of the American Statistical Association 113, no. 521 (2018): 390-400.

Examples

```
data(simdata)
spec <- rct_spec(z ~ unit_of_assignment(uoa1, uoa2), data = simdata)
summary(lmitt(y ~ 1, data = simdata, specification = spec, weights = ate()), vcov.type = "CR0")
```

---

get_structure `StudySpecification` *Structure Information*

Description

Obtaining a `data.frame` which encodes the specification information.

Usage

```
get_structure(specification)

## S4 method for signature 'StudySpecificationStructure'
show(object)
```

Arguments

`specification`	a `StudySpecification` object
`object`	a `StudySpecificationStructure` object, typically the output of `get_structure`

Value

A `StudySpecificationStructure` object containing the structure of the `specification` as a `data.frame`.

Examples

```
data(simdata)
spec <- rct_spec(z ~ uoa(uoa1, uoa2) + block(bid), data = simdata)
get_structure(spec)
```

GV_data *Cluster-randomized experiment data on voter turnout in cable system markets*

Description

This dataset is a toy example derived from a cluster-randomized field experiment that evaluates the effect of "Rock the Vote" TV advertisements on voter turnout rate. The original study included 23,869 first-time voters across 85 cable television markets in 12 states. These markets were grouped into matched sets based on their past voter turnout rates and then randomly assigned to either a treatment or control condition. This toy dataset is constructed by randomly sampling 10% of individuals from selected cable television markets in the original dataset.

Usage

```
GV_data
```

Format

A `data.frame` with 248 rows and 7 columns.

- age Age of participant
- vote_04 Outcome variable indicating whether participant voted
- tv_company Cable system serving participant's residential area
- treatment Binary variable denoting treatment assignment
- pairs A numeric indicator for the strata or matched pair group to which a cable system belongs (1-3)
- population_size Total population size of residential area served by cable system
- sample_size Number of individuals sampled from the cable system cluster

Details

The original dataset was drawn from a randomized controlled trial in which 85 cable system areas were first grouped into 40 matched sets based on historical voter turnout. Within each matched set, one cable system area was randomly assigned to the treatment condition, while the others served as controls.

This toy dataset includes a subset of the original replication data, specifically individuals from matched sets 1–3, which encompass 7 of the 85 cable system areas. Within these selected clusters, a 10% random sample of individuals was taken.

The fuller Green-Vavreck dataset that this derives from bears a Creative Commons BY-NC-ND license (v3.0) and is housed in Yale University's Institution for Social and Policy Studies (ID: D005).

Source

https://dataverse.yale.edu/dataset.xhtml?persistentId=doi:10.60600/YU/OYIPLJ

References

Green, Donald P. & Lynn Vavreck (2008) "Analysis of Cluster-Randomized Experiments: A Comparison of Alternative Estimation Approaches." Political Analysis 16(2):138-152.

---

has_binary_treatment *Check whether treatment stored in a* StudySpecification *object is binary*

Description

Check whether treatment stored in a StudySpecification object is binary

Usage

```
has_binary_treatment(spec)
```

Arguments

spec	StudySpecification object

Value

logical vector of length 1

---

identical_StudySpecifications *Test equality of two* StudySpecification *objects*

Description

Check whether two StudySpecification objects are identical.

Usage

```
identical_StudySpecifications(x, y)
```

Arguments

x	A StudySpecification object.
y	A StudySpecification object.

Value

Logical, are x and y identical?

identify_small_blocks *Identify fine strata*

Description

Identify blocks in a StudySpecification with exactly one treated or one control unit of assignment.

Usage

```
identify_small_blocks(spec)
```

Arguments

spec	A StudySpecification object.

Value

Logical vector with length given by the number of blocks in StudySpecification

lmitt *Linear Model for Intention To Treat*

Description

Generates a linear model object to estimate a treatment effect, with proper estimation of variances accounting for the study specification.

Usage

```
lmitt(obj, specification, data, ...)

## S3 method for class 'formula'
lmitt(
  obj,
  specification,
  data,
  absorb = FALSE,
  offset = NULL,
  weights = NULL,
  ...
)

## S3 method for class 'lm'
lmitt(obj, specification = NULL, ...)
```

Arguments

obj	A formula or a lm object. See Details.
specification	The StudySpecification to be used. Alternatively, a formula creating a specification. (Of the type of that would be passed as the first argument to rd_spec(), rct_spec(), or obs_spec(), with the difference that cluster(), uoa() and unit_of_assignment() terms can be omitted when each row of data represents a distinct unit of assignment.) If the formula includes a forcing() element, an RD specification is created. Otherwise an observational specification is created. An RCT specification must be created manually using rct_spec().
data	A data.frame such as would be passed into lm().
...	Additional arguments passed to lm() and other functions. An example of the latter is dichotomy=, a formula passed to assigned() and, as appropriate, ate(), att(), atc() or ato(). It is used to dichotomize a non-binary treatment variable in specification. See the Details section of the ate() help page for examples.
absorb	If TRUE, fixed effects are included for blocks identified in the StudySpecification. Excluded in FALSE. Default is FALSE. The estimates of these fixed effects are suppressed from the returned object.
offset	Offset of the kind which would be passed into lm(). Ideally, this should be the output of cov_adj().
weights	Which weights should be generated? Options are "ate" or "ett". Alternatively, the output of a manually run ate() or ett() can be used.

Details

The first argument to lmitt() should be a formula specifying the outcome on the left hand side. The right hand side of the formula can be any of the following:

- 1: Estimates a main treatment effect.
- a subgroup variable: Estimates a treatment effect within each level of your subgrouping variable.
- a continuous moderator: Estimates a main treatment effect as well as a treatment by moderator interaction. The moderator is not automatically centered.

Alternatively, obj can be a pre-created lm object. No modification is made to the formula of the object. See the help for as.lmitt() for details of this conversion.

The lmitt() function's subset= argument governs the subsetting of data prior to model fitting, just as with lm(). Functions such as rct_spec() that create StudySpecifications also take an optional subset= argument, but its role differs from that of the subset= argument of lm() or lmitt(). The subset= argument when creating a StudySpecification restricts the data used to generate the StudySpecification, but has no direct impact on the future lm() or lmitt() calls using that StudySpecification. (It can have an indirect impact by excluding particular units from receiving a treatment assignment or weight. When treatment assignments or weights are reconstructed from the StudySpecification, these units will receive NAs, and will be excluded from the lm() or lmitt() fit under typical na.action settings.)

To avoid variable name collision, the treatment variable defined in the `specification` will have a "`.`" appended to it. For example, if you request a main treatment effect (with a formula of `~ 1`) with a treatment variable named "txt", you can obtain its estimate from the returned `teeMod` object via `$coefficients["txt."]`.

`lmitt()` will produce a message if the `StudySpecification` designates treatment assignment by block but the blocking structure appears not to be reflected in the `weights`, nor in a block fixed effect adjustment (via `absorb=TRUE`). While not an error, this is at odds with intended uses of `propertee`, so `lmitt()` flags it as a potential oversight on the part of the analyst. To disable this message, run `options("propertee_message_on_unused_blocks" = FALSE)`.

`lmitt()` returns objects of class '`teeMod`', for Treatment Effect Estimate Model, extending the lm class to add a summary of the response distribution under control (the coefficients of a controls-only regression of the response on an intercept and any moderator variable). `teeMod` objects also record the underlying `StudySpecification` and information about any externally fitted models `mod` that may have been used for covariance adjustment by passing `offset=cov_adj(mod)`. In the latter case, responses are offsetted by predictions from `mod` prior to treatment effect estimation, but estimates of the response variable distribution under control are calculated without reference to `mod`.

The response distribution under control is also characterized when treatment effects are estimated with block fixed effects, i.e. for `lmitt()` with a `formula` first argument with option `absorb=TRUE`. Here as otherwise, the supplementary coefficients describe a regression of the response on an intercept and moderator variables, to which only control observations contribute; but in this case the weights are modified for this supplementary regression. The treatment effect estimates adjusted for block fixed effects can be seen to coincide with estimates calculated without block effect but with weights multiplied by an additional factor specific to the combination of block and treatment condition. For block s containing units with weights w_i and binary treatment assignments z_i, define $\hat{\pi}_s$ by $\hat{\pi}_s \sum_s w_i = \sum_s z_i w_i$. If $\hat{\pi}_s$ is 0 or 1, the block doesn't contribute to effect estimation and the additional weighting factor is 0; if $0 < \hat{\pi}_s < 1$, the additional weighting factor is $1 - \hat{\pi}_s$ for treatment group members and $\hat{\pi}_s$ for controls. When estimating a main effect only or a main effect with continuous moderator, supplementary coefficients under option `absorb=TRUE` reflect regressions with additional weighting factor equal to 0 or $\hat{\pi}_s$, respectively, for treatment or control group members of block s. With a categorical moderator and `absorb=TRUE`, this additional weighting factor determining supplementary coefficients is calculated separately for each level ℓ of the moderator variable, with the sums defining $\hat{\pi}_{s\ell}$ restricted not only to block s but also to observations with moderator equal to ℓ.

Value

`teeMod` object (see Details)

Examples

```
data(simdata)
spec <- rct_spec(z ~ cluster(uoa1, uoa2), data = simdata)
mod1 <- lmitt(y ~ 1, data = simdata, specification = spec, weights = "ate")
mod2 <- lmitt(y ~ as.factor(o), data = simdata, specification = spec, weights = "ate")
### observational study with treatment z assigned row-wise within blocks:
mod3 <- lmitt(y ~ 1, data=simdata, specification=z ~ block(bid), weights="att")
### regression discontinuity study with units of assignment
### given by combinations of uoa1, uoa2:
mod4 <- lmitt(y ~ 1, data = simdata,
```

```
specification = z ~ uoa(uoa1, uoa2) + forcing(force))
```

`lsoSynth` *Synthethic Regression Discontinuity Data*

Description

The data for this example were randomly simulated using the **synthpop** package in R based on data originally collected by Lindo, Sanders, and Oreopoulos (2010).

Usage

```
lsoSynth
```

Format

A `data.frame` with 40,403 rows and 11 columns.

- `R`
- `lhsgrade_pct`
- `nextGPA`
- `probation_year1`
- `totcredits_year1`
- `male`
- `loc_campus1`
- `loc_campus2`
- `bpl_north_america`
- `english`
- `age_at_entry`

Details

See the "Regression Discontinuity StudySpecifications" vignette on the `propertee` website for more details on the original data, a link to the code used to generate this synthethic data, and a detailed example.

michigan_school_pairs *Intervention data from a pair-matched study of schools in Michigan*

Description

Michigan high schools, with a plausible cluster RCT

Usage

```
michigan_school_pairs
```

Format

A `data.frame` with 14 rows and 13 columns.

- schoolid school id
- blk block
- z treatment variable
- MALE_G11_PERC percentage of G11 male students
- FEMALE_G11_PERC percentage of G11 female students
- AM_G11_PERC percentage of G11 American Indian/Alaska Native students
- ASIAN_G11_PERC percentage of G11 Asian students
- HISP_G11_PERC percentage of G11 Hispanic students
- BLACK_G11_PERC percentage of G11 Black students
- WHITE_G11_PERC percentage of G11 White students
- PACIFIC_G11_PERC percentage of G11 Hawaiian Native/Pacific Islander students
- TR_G11_PERC percentage of G11 Two or More Races students
- G11 Number of G11 students

Details

Grade 11 demographics for all Michigan high schools in 2013, with mock block and treatment assignments for 14 high schools within a large county in the metro Detroit area. These schools were selected for this demonstration based on their similarity to the 14 high schools from an adjacent Michigan county that participated in the Pane et al (2013) study. As a result, they serve as an example of what one might expect to find as the state-specific school-level subsample in a multi-state paired cluster randomized trial featuring random assignment at the school level.

The mock experimental schools were selected by optimal matching of experimental schools to adjacent county schools, with substitute schools grouped into the same pairs or triples ('fine strata') as were their experimental counterparts. The original pairs and triples had been selected to reduce variation in baseline variables predictive of outcomes, and the blocking structure the substitute sample inherits may be expected to do this as well. The treatment/control distinction is also inherited from the experimental sample, but there is of course no treatment effect within the mock experiment.

The selection of mock experimental schools was based on both demographic and student achievement variables, but the present data frame includes only the demographic variables (as sourced from the Common Core of Data [CCD; U.S. Department of Education]). School average outcomes in student test scores are available separately, from Michigan's Center for Education Performance Information. See the vignette 'Real-data demonstration with a finely stratified cluster RCT and a broader administrative database', available on the package website.

References

Pane, John F., et al. "Effectiveness of cognitive tutor algebra I at scale." *Educational Evaluation and Policy Analysis* 36.2 (2014): 127-144.

U.S. Department of Education. Public Elementary/Secondary School Universe Survey Data, v.2a. Institute of Education Sciences, National Center for Education Statistics.

Examples

```
data(michigan_school_pairs)
mi_spec <- rct_spec(z ~ uoa(schoolid)+block(blk),
data=michigan_school_pairs)
mi_spec
table(is.na(michigan_school_pairs$blk))
specification_table(mi_spec, "block", "treatment")
```

---

rct_spec *Generates a* StudySpecification *object with the given specifications.*

Description

Generate a randomized control treatment StudySpecification (rct_spec()), or an observational StudySpecification (obs_spec()), or a regression discontinuity StudySpecification (rd_spec()).

Usage

```
rct_spec(formula, data, subset = NULL, na.fail = TRUE)

rd_spec(formula, data, subset = NULL, na.fail = TRUE)

obs_spec(formula, data, subset = NULL, na.fail = TRUE)

rct_specification(formula, data, subset = NULL, na.fail = TRUE)

rd_specification(formula, data, subset = NULL, na.fail = TRUE)

obs_specification(formula, data, subset = NULL, na.fail = TRUE)

obsstudy_spec(formula, data, subset = NULL, na.fail = TRUE)

obsstudy_specification(formula, data, subset = NULL, na.fail = TRUE)
```

Arguments

formula	a formula defining the StudySpecification components. See Details for specification.
data	the data set from which to build the StudySpecification. Note that this data need not be the same as used to estimate the treatment effect; rather the data passed should contain information about the units of treatment assignment (as opposed to the units of analysis).
subset	optional, subset the data before creating the StudySpecification object
na.fail	If TRUE (default), any missing data found in the variables specified in formula (excluding treatment) will trigger an error. If FALSE, non-complete cases will be dropped before the creation of the StudySpecification

Details

The formula should include exactly one unit_of_assignment() to identify the units of assignment (one or more variables). (uoa, cluster, or unitid are synonyms for unit_of_assignment; the choice of which has no impact on the analysis. See below for a limited exception in which the unit_of_assignment specification may be omitted.) If defining an rd_spec, the formula must also include a forcing() entry. The formula may optionally include a block() as well. Each of these can take in multiple variables, e.g. to pass both a household ID and individual ID as unit of assignment, use uoa(hhid, iid) and not uoa(hhid) + uoa(iid).

The treatment variable passed into the left-hand side of formula can either be logical, numeric, or character. If it is anything else, it attempts conversion to one of those types (for example, factor and ordered are converted to numeric if the levels are numeric, otherwise to character). If the treatment is not logical or numeric with only values 0 and 1, in order to generate weights with ate() or ett(), the dichotomy argument must be used in those functions to identify the treatment and control groups. See ett() for more details on specifying a dichotomy.

There are a few aliases for each version.

If the formula excludes a unit_of_assignment(), data merges are performed on row order. Such formulas can also be passed as the specification argument to lmitt(), and that is their primary intended use case. It is recommended that each formula argument passed to *_specification() include a unit_of_assignment(), uoa() or cluster() term identifying the key variable(s) with which StudySpecification data is to be merged with analysis data. Exceptions to this rule will be met with a warning. To disable the warning, run options("propertee_warn_on_no_unit_of_assignment" = FALSE).

The units of assignment, blocks, and forcing variables must be numeric or character. If they are otherwise, an attempt is made to cast them into character.

Value

a StudySpecification object of the requested type for use in further analysis.

Examples

```
data(simdata)
spec <- rct_spec(z ~ unit_of_assignment(uoa1, uoa2) + block(bid),
                 data = simdata)
```

```
data(schooldata)
spec <- obs_spec(treatment ~ unit_of_assignment(schoolid) + block(state),
                 data = schooldata)
```

`schooldata` *Student data*

Description

An example of data sets stored at two levels.

Usage

```
schooldata

studentdata
```

Format

Two `data.frames`, one with school-level data (`schooldata`) including treatment assignment and a second with student-level data (`studentdata`). `schoolata`:

- schoolid Unique school ID variable.
- treatment Was this school in the intervention group?
- state State which the school is in.
- pct_disadvantage Percent of student body flagged as "disadvantaged".

`studentdata`:

- id Unique student ID.
- schoolid Unique school ID variable.
- grade Student's grade, 3-5.
- gpa Student GPA in prior year.
- math Standarized math score (out of 100).

An object of class `data.frame` with 8713 rows and 5 columns.

Details

In this hypothetical data, schools were randomly assignment to treatment status, but the unit of analysis is students. Thus the two data sets, one encoding school information (including treatment status) and one encoding student information (which does not include treatment status).

Examples

```
soec <- obs_spec(treatment ~ uoa(schoolid), data = schooldata)

# Treatment effect
mod1 <- lmitt(math ~ 1, specification = soec, data = studentdata)

# Treatment effect by grade
mod2 <- lmitt(math ~ as.factor(grade), specification = soec, data = studentdata)
```

---

show,PreSandwichLayer-method

Show a `PreSandwichLayer` *or* `SandwichLayer`

Description

Display information about a `PreSandwichLayer` or `SandwichLayer` object

Usage

```
## S4 method for signature 'PreSandwichLayer'
show(object)
```

Arguments

`object`	`PreSandwichLayer` or `SandwichLayer` object

Value

an invisible copy of `object`

---

show,StudySpecification-method

Show a `StudySpecification`

Description

Display information about a `StudySpecification` object

Usage

```
## S4 method for signature 'StudySpecification'
show(object)
```

Arguments

`object`	`StudySpecification` object, usually a result of a call to `rct_spec()`, `obs_spec()`, or `rd_spec()`.

Value

object, invisibly.

---

show,teeMod-method *Show a* teeMod

Description

Display information about a teeMod object

Usage

```
## S4 method for signature 'teeMod'
show(object)
```

Arguments

object	teeMod object, usually a result of a call to lmitt().

Value

object, invisibly.

---

show,WeightedStudySpecification-method *Show a* WeightedStudySpecification

---

Description

Prints out the weights from a WeightedStudySpecification

Usage

```
## S4 method for signature 'WeightedStudySpecification'
show(object)
```

Arguments

object	a WeightedStudySpecification object

Value

an invisible copy of object

simdata *Simulated data*

Description

Simulated data to use with the **propertee** package with unit of assignment level treatment assignment

Usage

```
simdata
```

Format

A data.frame with 100 rows and 7 columns.

- `uoa1` First level unit of assignment ID
- `uoa2` Second level unit of assignment ID
- `bid` Block ID
- `force` Forcing variable
- `z` Binary treatment indicator
- `o` 4-level ordered treatment variable
- `dose` Dose treatment variable
- `x` Some predictor
- `y` Some outcome

specification_data_concordance *Check for variable agreement within units of assignment*

Description

Useful for debugging purposes to ensure that there is concordance between variables in the `StudySpecification` and data.

Usage

```
specification_data_concordance(
  specification,
  data,
  by = NULL,
  warn_on_nonexistence = TRUE
)
```

Arguments

specification	a StudySpecification object
data	a new data set, presumably not the same used to create specification.
by	optional; named vector or list connecting names of variables in specification to variables in data. Names represent variables in specification; values represent variables in data. Only needed if variable names differ.
warn_on_nonexistence	default TRUE. If a variable does not exist in data, should this be flagged? If FALSE, silently move on if a variable doesn't exist in data.

Details

Consider the following scenario: A StudySpecification is generated from some dataset, "data1", which includes a block variable "b1". Within each unique unit of assignment/unitid/cluster of "data1", it must be the case that "b1" is constant. (Otherwise the creation of the StudySpecification will fail.)

Next, a model is fit which includes weights generated from the StudySpecification, but on dataset "data2". In "data2", the block variable "b1" also exists, but due to some issue with data cleaning, does not agree with "b1" in "data1".

This could cause errors, either directly (via actual error messages) or simply produce nonsense results. specification_data_concordance() is specificationed to help debug these scenarios by providing information on whether variables in both the data used in the creation of specification ("data1" in the above example) and some new dataset, data, ("data2" in the above example) have any inconsistencies.

Value

invisibly TRUE if no warnings are produced, FALSE if any warnings are produced.

---

specification_table *Table of elements from a* StudySpecification

---

Description

Produces a table (1-dimensional, or 2-dimensional if y is specified) of the elements of the StudySpecification.

Usage

```
specification_table(
  specification,
  x,
  y = NULL,
  sort = FALSE,
  decreasing = TRUE,
  use_var_names = FALSE,
```

```
  ...
)

stable(
  specification,
  x,
  y = NULL,
  sort = FALSE,
  decreasing = TRUE,
  use_var_names = FALSE,
  ...
)
```

Arguments

specification	A StudySpecification object
x	One of "treatment", "unit of assignment", (synonym "uoa"), "block". Abbreviations are accepted. "unit of assignment" can be replaced by "unitid" or "cluster" if the StudySpecification was created with that element.
y	Optionally, another string similar to x. A 1-dimensional table is produced if y is left at its default, NULL.
sort	Ignored if y is not NULL. If FALSE (default), one-way table is sorted according to "names" of levels. If set to TRUE, one-way table is sorted according to values.
decreasing	If sort is TRUE, choose whether to sort descending (TRUE, default) or ascending (FALSE).
use_var_names	If TRUE, name dimensions of table returned by variable names. If FALSE (default), name by their function (e.g. "treatment" or "blocks"). Passing the dnn argument in ... (an argument of table()) overrides whatever is requested here.
...	additional arguments table()

Value

A table of the requested variables.

Examples

```
data(simdata)
spec <- obs_spec(z ~ unit_of_assignment(uoa1, uoa2) + block(bid),
                 data = simdata)
specification_table(spec, "treatment")
specification_table(spec, "treatment", "block", sort = TRUE, use_var_names = TRUE)
```

STARplus *STAR participants plus nonexperimental controls*

Description

Data from Tennessee's Project STAR study. This data frame describes student participants in the Project STAR (Student-Teacher Achievement Ratio) field experiment conducted in Tennessee, USA beginning in the mid-1980s, as well as an external control group consisting of the contemporaneous cohort of students attending a matched sample of Tennessee schools that did not participate in the STAR experiment. Variables are as described in Project STAR data documentation (see references), with five exceptions. Three `*_at_entry` variables were constructed as follows: `grade_at_entry` indicates the grade of student's first participation, while `school_at_entry` and `cond_at_entry` reflect the school ID and classroom type corresponding to the student's grade at entry to the study. Additionally, `read_yr1` and `math_yr1` capture a student's scaled scores on the Scholastic Assessment Test (SAT) administered to them during their `grade_of_entry`, i.e. their earliest available post-treatment SAT measurements.

Usage

```
STARplus
```

Format

A `data.frame` with 13,382 rows and 56 columns.

- stdnid Student ID
- gender Student gender
- race Student race
- birthmonth Student month of birth
- birthday Student day of birth
- birthyear Student year of birth
- read_yr1 SAT reading scaled score from grade at which student entered the study
- math_yr1 SAT math scaled score from grade at which student entered the study
- gktreadss Kindergarten reading scaled score (RCT participants only)
- gktmathss Kindergarten math scaled score (RCT participants only)
- gktlistss Kindergarten listening scaled score (RCT participants only)
- gkwordskillss Kindergarten word study skills scaled score (RCT participants only)
- g1schid Grade 1 School ID
- g1tchid Grade 1 Teacher ID
- g1classsize Class size of Grade 1
- g1treadss Grade 1 SAT reading scaled score
- g1tmathss Grade 1 SAT math scaled score

- g1tlistss Grade 1 total listening scale score in SAT
- g1wordskillss Grade 1 word study skills scale score in SAT
- g1readbsraw Grade 1 reading raw score in Basic Skills First (BSF) tests
- g1mathbsraw Grade 1 math raw score in BSF
- g1readbsobjpct Grade 1 reading percent objectives mastered in BSF tests
- g1mathbsobjpct Grade 1 math percent objectives mastered in BSF tests
- g2schid Grade 2 School ID
- g2tchid Grade 2 Teacher ID
- g2classsize Class size of Grade 2
- g2treadss Grade 2 total reading scale score in SAT
- g2tmathss Grade 2 total math scale score in SAT
- g2tlistss Grade 2 total listening scale score in SAT
- g2wordskillss Grade 2 word study skills scale score in SAT
- g2readbsraw Grade 2 reading raw score in BSF tests
- g2mathbsraw Grade 2 math raw score in BSF test
- g2readbsobjpct Grade 2 reading percent objectives mastered in BSF tests
- g3schid Grade 3 School ID
- g3tchid Grade 3 Teacher ID
- g3classsize Class size of Grade 3
- g3treadss Grade 3 total reading scale score in SAT
- g3tmathss Grade 3 total math scale score in SAT
- g3langss Grade 3 total language scale score in SAT
- g3tlistss Grade 3 total listening scale score in SAT
- g3socialsciss Grade 3 social science scale score in SAT
- g3spellss Grade 3 spelling scale score in SAT
- g3vocabss Grade 3 vocabulary scale score in SAT
- g3mathcomputss Grade 3 math computation scale score in SAT
- g3mathnumconcss Grade 3 concept of numbers scale score in SAT
- g3mathapplss Grade 3 math applications scale score in SAT
- g3wordskillss Grade 3 word study skills scale score in SAT
- g3readbsraw Grade 3 reading raw score in BSF tests
- g3mathbsraw Grade 3 math raw score in BSF tests
- g3readbsobjpct Grade 3 reading percent objectives mastered in BSF tests
- g3mathbsobjpct Grade 3 math percent objectives mastered in BSF tests
- dob Date of birth (with NAs imputed RCT participant median)
- dobNA Dat of birth not recorded
- grade_at_entry Grade at which each student first entered the study
- school_at_entry School ID corresponding to the student's grade at entry into the study
- cond_at_entry Classroom type corresponding to the student's grade at entry into the study

Details

Note: This dataset bears a Creative Commons Zero license (v1.0).

Source

doi:10.7910/DVN/SIWH9F

References

C.M. Achilles; Helen Pate Bain; Fred Bellott; Jayne Boyd-Zaharias; Jeremy Finn; John Folger; John Johnston; Elizabeth Word, 2008, "Tennessee's Student Teacher Achievement Ratio (STAR) project", Harvard Dataverse, V1, https://doi.org/10.7910/DVN/SIWH9F UNF:3:Ji2Q+9HCCZAbw3csOdMNdA

---

subset,PreSandwichLayer-method

`PreSandwichLayer` *and* `SandwichLayer` *subsetting*

Description

Return subset of a `PreSandwichLayer` or `SandwichLayer` which meets conditions.

Usage

```
## S4 method for signature 'PreSandwichLayer'
subset(x, subset)

## S4 method for signature 'PreSandwichLayer'
x[i]
```

Arguments

`x`	`PreSandwichLayer` or `SandwichLayer` object
`subset`	Logical vector identifying values to keep or drop
`i`	indices specifying elements to extract or replace. See `help("[")` for further details.

Value

`x` subset by `subset` or `i`

subset,WeightedStudySpecification-method
WeightedStudySpecification *subsetting*

Description

Provides functionality to subset the weights of a WeightedStudySpecification object.

Usage

```
## S4 method for signature 'WeightedStudySpecification'
subset(x, subset)

## S4 method for signature 'WeightedStudySpecification'
x[i]
```

Arguments

x	WeightedStudySpecification object
subset	Logical vector identifying values to keep or drop
i	indices specifying elements to extract or replace. See help("[") for further details.

Value

A WeightedStudySpecification object which is a subsetted version of x.

summary.StudySpecification
Summarizing StudySpecification *objects*

Description

summary() method for class StudySpecification.

Usage

```
## S3 method for class 'StudySpecification'
summary(object, ..., treatment_binary = TRUE)

## S3 method for class 'summary.StudySpecification'
print(x, ..., max_unit_print = 3)
```

Arguments

object	StudySpecification object, usually a result of a call to rct_spec(), obs_spec(), or rd_spec().
...	Ignored
treatment_binary	Should the treatment be dichotomized if object contains a dichotomy? Ignored if object does not contain a dichotomy.
x	summary.StudySpecification object, usually as a result of a call to summary.StudySpecification()
max_unit_print	Maximum number of treatment levels to print in treatment table

Value

The StudySpecification or summary.StudySpecificationobject, invisibly

---

summary.teeMod *Summarizing* teeMod *objects*

Description

summary() method for class teeMod

Usage

```
## S3 method for class 'teeMod'
summary(object, vcov.type = "HC0", ...)

## S3 method for class 'summary.teeMod'
print(
  x,
  digits = max(3L, getOption("digits") - 3L),
  signif.stars = getOption("show.signif.stars"),
  ...
)
```

Arguments

object	teeMod object
vcov.type	A string indicating the desired variance estimator. See vcov_tee() for details on accepted types.
...	Additional arguments to vcov_tee(), such as the desired finite sample heteroskedasticity-robust standard error adjustment.
x	summary.teeMod object
digits	the number of significant digits to use when printing.
signif.stars	logical. If 'TRUE', 'significance stars' are printed for each coefficient.

Details

If a teeMod object is fit with a SandwichLayer offset, then the usual stats::summary.lm() output is enhanced by the use of covariance-adjusted sandwich standard errors, with t-test values recalculated to reflect the new standard errors.

Value

object of class summary.teeMod

---

treatment *Accessors and Replacers for* StudySpecification *objects*

Description

Allows access to the elements which define a StudySpecification, enabling their extraction or replacement.

Usage

```
treatment(x, newdata = NULL, dichotomy = NULL, by = NULL, ...)

## S4 method for signature 'StudySpecification'
treatment(x, newdata = NULL, dichotomy = NULL, by = NULL, ...)

treatment(x) <- value

## S4 replacement method for signature 'StudySpecification'
treatment(x) <- value

units_of_assignment(x, newdata = NULL, by = NULL)

## S4 method for signature 'StudySpecification'
units_of_assignment(x, newdata = NULL, by = NULL)

units_of_assignment(x) <- value

## S4 replacement method for signature 'StudySpecification'
units_of_assignment(x) <- value

clusters(x, newdata = NULL, by = NULL)

## S4 method for signature 'StudySpecification'
clusters(x, newdata = NULL, by = NULL)

clusters(x) <- value

## S4 replacement method for signature 'StudySpecification'
```

```
clusters(x) <- value

unitids(x)

## S4 method for signature 'StudySpecification'
unitids(x)

unitids(x) <- value

## S4 replacement method for signature 'StudySpecification'
unitids(x) <- value

blocks(x, newdata = NULL, by = NULL, ...)

## S4 method for signature 'StudySpecification'
blocks(x, newdata = NULL, by = NULL, ..., implicit = FALSE)

blocks(x) <- value

## S4 replacement method for signature 'StudySpecification'
blocks(x) <- value

has_blocks(x)

forcings(x, newdata = NULL, by = NULL)

## S4 method for signature 'StudySpecification'
forcings(x, newdata = NULL, by = NULL)

forcings(x) <- value

## S4 replacement method for signature 'StudySpecification'
forcings(x) <- value
```

Arguments

x	a StudySpecification object
newdata	optional; an additional data.frame. If passed, and the unit of assignment variable is found in newdata, then the requested variable type for each unit of newdata is returned. See by argument if the name of the unit of assignment differs.
dichotomy	optional; a formula specifying how to dichotomize a non-binary treatment variable. See the Details section of the ett() or att() help pages for information on specifying this formula
by	optional; named vector or list connecting names of unit of assignment/unitid/cluster variables in x to unit of assignment/unitid/cluster variables in data. Names represent variables in x; values represent variables in newdata. Only needed if variable names differ.

...	ignored.
value	replacement. Either a vector/matrix of appropriate dimension, or a named data.frame if renaming variable as well. See Details.
implicit	Should a block-less StudySpecification return a constant 1 when extracting blocks?

Details

For treatment(), when argument binary is FALSE, the treatment variable passed into the StudySpecification is returned as a one-column data.frame regardless of whether it is binary or x has a dichotomy

If a dichotomy is passed, a binary one-column data.frame will be returned. If not and binary is TRUE, unless the StudySpecification has a binary treatment, treatment() will error. If binary is "ifany", it will return the original treatment in this case.

The one-column data.frame returned by treatment() is named as entered in the StudySpecification creation, but if a dichotomy is passed, the column name is "__z" to try and avoid any name conflicts.

For the value when using replacers, the replacement must have the same number of rows as the StudySpecification (the same number of units of assignment). The number of columns can differ (e.g. if the StudySpecification were defined with two variable uniquely identifying blocks, you can replace that with a single variable uniquely identifying blocks, as long as it respects other restrictions.)

If the replacement value is a data.frame, the name of the columns is used as the new variable names. If the replacement is a matrix or vector, the original names are retained. If reducing the number of variables (e.g., moving from two variables uniquely identifying to a single variable), the appropriate number of variable names are retained. If increasing the number of variables, a data.frame with names must be provided.

Value

data.frame containing requested variable, or an updated StudySpecification. treatment() works slightly differently, see Details.

Examples

```
data(simdata)
spec <- obs_spec(z ~ unit_of_assignment(uoa1, uoa2), data = simdata)
blocks(spec) # empty
blocks(spec) <- data.frame(blks = c(1, 1, 2, 2, 3, 3, 4, 4, 5, 5))
blocks(spec)
blocks(spec) <- c(5, 5, 4, 4, 3, 3, 2, 2, 1, 1)
blocks(spec) # notice that variable is not renamed
```

unit_of_assignment *Special terms in* StudySpecification *creation formula*

Description

These are special functions used only in the definition of StudySpecification objects. They identify the units of assignment, blocks and forcing variables. They should never be used outside of the formula argument to obs_spec, rct_spec, or rd_spec.

Usage

```
unit_of_assignment(...)

unitid(...)

cluster(...)

uoa(...)

block(...)

forcing(...)
```

Arguments

...	any number of variables of the same length.

Details

These functions have no use outside of the formula in creating a StudySpecification.

unit_of_assignment, uoa, cluster and unitid are synonyms; you must include one and only one in each StudySpecification. The choice of which to use will have no impact on any analysis, only on some output and the name of the stored element in the StudySpecification. Accessors/ replacers (units_of_assignment, unitids, clusters) respect the choice made at the point of creation of the StudySpecification, and only the appropriate function will work.

See rct_spec, obs_spec, or rd_spec for examples of their usage.

Value

the variables with appropriate labels. No use outside of their inclusion in the formula argument to obs_spec, rct_spec, or rd_spec

var_table *Extract Variable Names from* StudySpecification

Description

Methods to extract the variable names to the elements of the structure of the StudySpecification (e.g. treatment, unit of analysis, etc)

Usage

```
var_table(specification, compress = TRUE, report_all = FALSE)

var_names(specification, type, implicitBlocks = FALSE)
```

Arguments

specification	a StudySpecification object
compress	should multiple variables be compressed into a comma-separated string? Default TRUE. If FALSE, multiple columns can be created instead.
report_all	should we report all possible structures even if they don't exist in the StudySpecification? Default FALSE.
type	one of "t", "u", "b", "f"; for "treatment", "unit_of_assignment", "block", and "forcing" respectively
implicitBlocks	If the StudySpecification is created without blocks, setting this to TRUE will return ".blocks_internal" as the variable name corresponding to the blocks.

Details

When compress is TRUE, the result will always have two columns. When FALSE, the result will have number of columns equal to the largest number of variables in a particular role, plus one. E.g., a call such as rct_spec(z ~ unitid(a, b, c, d) ... will have 4+1=5 columns in the output matrix with compress = FALSE.

When report_all is TRUE, the matrix is guaranteed to have 3 rows (when the specification is an RCT or Obs) or 4 rows (when the specification is a RD), with empty variable entries as appropriate. When FALSE, the matrix will have minimum 2 rows (treatment and unit of assignment/unitid/cluster), with additional rows for blocks and forcing if included in the StudySpecification.

Value

var_table returns the requested table. var_names returns a vector of variable names.

Examples

```
spec <- rct_spec(z ~ uoa(uoa1, uoa2) + block(bid), data = simdata)
var_table(spec)
var_table(spec, compress = FALSE)
var_names(spec, "t")
var_names(spec, "u")
var_names(spec, "b")
```

`vcov.teeMod` *Compute variance-covariance matrix for fitted* `teeMod` *model*

Description

An S3method for `stats::vcov` that computes standard errors for `teeMod` models using `vcov_tee()`.

Usage

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

Arguments

`object`	a fitted `teeMod` model
`...`	additional arguments to `vcov_tee()`.

Details

`vcov.teeMod()` wraps around `vcov_tee()`, so additional arguments passed to `...` will be passed to the `vcov_tee()` call. See documentation for `vcov_tee()` for information about necessary arguments.

Value

A variance-covariance matrix with row and column entries for the estimated coefficients in `x`, the marginal mean outcome in the control condition, the marginal mean `offset` in the control condition (if an `offset` is provided), and if a moderator variable is specified in the formula for `x`, the mean interaction in the control condition of the outcome and `offset` with the moderator variable

`weights,WeightedStudySpecification-method`
Extract Weights from `WeightedStudySpecification`

Description

A `WeightedStudySpecification` object contains a numeric vector with a few additional slots, this extracts only the numeric vector.

Usage

```
## S4 method for signature 'WeightedStudySpecification'
weights(object, ...)
```

Arguments

`object`	a `WeightedStudySpecification` object
`...`	Ignored

Value

A numeric `vector` of the weights

Index

∗ **datasets**
schooldata, 28
∗ **dataset**
GV_data, 19
lsoSynth, 24
michigan_school_pairs, 25
schooldata, 28
simdata, 31
STARplus, 34
*,WeightedStudySpecification,numeric-method *(+,WeightedStudySpecification,numeric-method)*, 3
*,numeric,WeightedStudySpecification-method *(+,WeightedStudySpecification,numeric-method)*, 3
+,WeightedStudySpecification,numeric-method, 3
+,numeric,WeightedStudySpecification-method *(+,WeightedStudySpecification,numeric-method)*, 3
-,WeightedStudySpecification,numeric-method *(+,WeightedStudySpecification,numeric-method)*, 3
-,numeric,WeightedStudySpecification-method *(+,WeightedStudySpecification,numeric-method)*, 3
/,WeightedStudySpecification,numeric-method *(+,WeightedStudySpecification,numeric-method)*, 3
/,numeric,WeightedStudySpecification-method *(+,WeightedStudySpecification,numeric-method)*, 3
[,PreSandwichLayer-method *(subset,PreSandwichLayer-method)*, 36
[,WeightedStudySpecification-method *(subset,WeightedStudySpecification-method)*, 37

a. *(assigned)*, 6
adopters *(assigned)*, 6
as.lmitt, 4
as.SandwichLayer, 5
as.teeMod *(as.lmitt)*, 4
as_obs_spec *(as_rct_spec)*, 7
as_rct_spec, 7
as_rd_spec *(as_rct_spec)*, 7
assigned, 6
assigned(), *5*, *6*, *22*
atc *(ett)*, 16
atc(), *22*
ate *(ett)*, 16
ate(), *9*, *22*, *27*
att *(ett)*, 16
ato(), *22*
att *(ett)*, 16
att(), *22*

block *(unit_of_assignment)*, 42
block(), *27*
blocks *(treatment)*, 39
blocks,StudySpecification-method *(treatment)*, 39
blocks<- *(treatment)*, 39
blocks<-,StudySpecification-method *(treatment)*, 39
bread.glmrob *(estfun.glmrob)*, 13
bread.lmrob *(estfun.lmrob)*, 14
bread.teeMod, 8

c(), *9*
c,WeightedStudySpecification-method, 9
cluster *(unit_of_assignment)*, 42
cluster(), *22*
clusters *(treatment)*, 39
clusters,StudySpecification-method *(treatment)*, 39
clusters<- *(treatment)*, 39
clusters<-,StudySpecification-method *(treatment)*, 39

Comparison, *17*
confint.teeMod, 10
cov_adj, 11
cov_adj(), *4, 5*, *22*

default, 13

estfun.glmrob, 13
estfun.lmrob, 14
estfun.teeMod, 15
etc *(*ett*)*, 16
ett, 16
ett(), *9*, *27*

forcing *(*unit_of_assignment*)*, 42
forcing(), *22*, *27*
forcings *(*treatment*)*, 39
forcings,StudySpecification-method *(*treatment*)*, 39
forcings<- *(*treatment*)*, 39
forcings<-,StudySpecification-method *(*treatment*)*, 39

get_structure, 18
GV_data, 19

has_binary_treatment, 20
has_blocks *(*treatment*)*, 39

identical_StudySpecifications, 20
identify_small_blocks, 21

lm(), *4*, *6, 7*, *17*, *22*
lmitt, 21
lmitt(), *5, 6*, *11*, *17*, *22, 23*, *30*
Logic, *17*
lsoSynth, 24

match(), *17*
michigan_school_pairs, 25

obs_spec *(*rct_spec*)*, 26
obs_spec(), *22*, *26*, *29*, *38*
obs_specification *(*rct_spec*)*, 26
obsstudy_spec *(*rct_spec*)*, 26
obsstudy_specification *(*rct_spec*)*, 26
olw *(*ett*)*, 16
owt *(*ett*)*, 16

print.summary.StudySpecification *(*summary.StudySpecification*)*, 37
print.summary.teeMod *(*summary.teeMod*)*, 38
pwt *(*ett*)*, 16

rct_spec, 26
rct_spec(), *22*, *26*, *29*, *38*
rct_specification *(*rct_spec*)*, 26
rd_spec *(*rct_spec*)*, 26
rd_spec(), *22*, *26*, *29*, *38*
rd_specification *(*rct_spec*)*, 26

schooldata, 28
show,PreSandwichLayer-method, 29
show,StudySpecification-method, 29
show,StudySpecificationStructure-method *(*get_structure*)*, 18
show,teeMod-method, 30
show,WeightedStudySpecification-method, 30
simdata, 31
specification_data_concordance, 31
specification_data_concordance(), *32*
specification_table, 32
stable *(*specification_table*)*, 32
STARplus, 34
stats::vcov(), *11*
studentdata *(*schooldata*)*, 28
subset,PreSandwichLayer-method, 36
subset,WeightedStudySpecification-method, 37
summary(), *37, 38*
summary.StudySpecification, 37
summary.StudySpecification(), *38*
summary.teeMod, 38

table(), *33*
treatment, 39
treatment(), *7*, *41*
treatment,StudySpecification-method *(*treatment*)*, 39
treatment<- *(*treatment*)*, 39
treatment<-,StudySpecification-method *(*treatment*)*, 39

unit_of_assignment, 42
unit_of_assignment(), *22*, *27*

unitid *(*unit_of_assignment*)*, 42
unitids *(*treatment*)*, 39
unitids,StudySpecification-method
*(*treatment*)*, 39
unitids<- *(*treatment*)*, 39
unitids<-,StudySpecification-method
*(*treatment*)*, 39
units_of_assignment *(*treatment*)*, 39
units_of_assignment,StudySpecification-method
*(*treatment*)*, 39
units_of_assignment<- *(*treatment*)*, 39
units_of_assignment<-,StudySpecification-method
*(*treatment*)*, 39
uoa *(*unit_of_assignment*)*, 42
uoa(), *22*

var_names *(*var_table*)*, 43
var_table, 43
vcov.teeMod, 44
vcov_tee(), *11*, *38*

weights,WeightedStudySpecification-method,
45

z. *(*assigned*)*, 6