Package ‘cregg’

June 28, 2020

Type Package

Title Simple Conjoint Tidying, Analysis, and Visualization

Version 0.4.0

Date 2020-06-28

Description Simple tidying, analysis, and visualization of conjoint (factorial) experiments, including estimation and visualization of average marginal component effects ('AMCEs') and marginal means ('MMs') for weighted and un-weighted survey data, along with useful reference category diagnostics and statistical tests. Estimation of 'AMCEs' is based upon methods described by Hainmueller, Hopkins, and Yamamoto (2014) <doi:10.1093/pan/mpt024>.

License MIT + file LICENSE

URL `https://github.com/leeper/cregg`

BugReports `https://github.com/leeper/cregg/issues`

Depends R (>= 3.5.0)

Imports stats, sandwich (>= 2.4-0), survey (>= 3.33), lmtest, ggplot2 (>= 2.0), ggstance, scales, utils

Suggests testthat, knitr, rmarkdown

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.0.2

NeedsCompilation no

Author Thomas J. Leeper [aut, cre] (<https://orcid.org/0000-0003-4097-6326>), Matthew Barnfield [ctb]

Maintainer Thomas J. Leeper <`thosjleeper@gmail.com`>

Repository CRAN

Date/Publication 2020-06-28 21:20:03 UTC

R topics documented:

`amce` *Tidy estimation of AMCEs*

Description

Estimate AMCEs for a conjoint analysis and return a tidy data frame of results

Usage

```
amce(
  data,
  formula,
  id = ~0,
  weights = NULL,
  feature_order = NULL,
  feature_labels = NULL,
  level_order = c("ascending", "descending"),
  alpha = 0.05,
  ...
)

amce_by_reference(data, formula, variable, ...)
```

Arguments

`data`	A data frame containing variables specified in `formula`. All RHS variables should be factors; the base level for each will be used in estimation and its reported AMCE will be NA (for printing). Optionally, this can instead be an object of class "survey.design" returned by `svydesign`.

`formula`	A formula specifying an AMCE model to be estimated. All variables should be factors; all levels across features should be unique. Two-way constraints can be specified with an asterisk (*) between RHS features. The specific constrained level pairs within these features are then detected automatically. Higher-order constraints are not allowed.
`id`	An RHS formula specifying a variable holding respondent identifiers, to be used for clustering standard errors. By default, data are unclustered.
`weights`	An (optional) RHS formula specifying a variable holding survey weights.
`feature_order`	An (optional) character vector specifying the names of feature (RHS) variables in the order they should be encoded in the resulting data frame.
`feature_labels`	A named list of "fancy" feature labels to be used in output. By default, the function looks for a "label" attribute on each variable in `formula` and uses that for pretty printing. This argument overrides those attributes or otherwise provides fancy labels for this purpose. This should be a list with names equal to variables on the righthand side of `formula` and character string values; arguments passed here override variable attributes.
`level_order`	A character string specifying levels (within each feature) should be ordered increasing or decreasing in the final output. This is mostly only consequential for plotting via `plot.cj_mm`, etc.
`alpha`	A numeric value indicating the significance level at which to calculate confidence intervals for the MMs (by default 0.95, meaning 95-percent CIs are returned).
`...`	For `amce`: additional arguments to `glm` or `svyglm`, the latter being used if `weights` is non-NULL. For `amce_by_reference`: additional arguments passed to `amce`.
`variable`	An RHS formula containing a single factor variable from `formula`. This will be used by `amce_by_reference` to estimate AMCEs relative to each possible factor level as a reference category. If more than one RHS variables are specified, the first will be used.

Details

`amce` provides estimates of AMCEs (or rather, average marginal effects for each feature level). Two-way constraints can be specified with an asterisk (*) between features. The specific constrained level pairs within these features are then detected automatically. The function can also be used for calculating average component interaction effects when combined with `interaction`, and for balance testing by specifying a covariate rather outcome on the left-hand side of `formula`. See examples.

`amce_by_reference` provides a tool for quick sensitivity analysis. AMCEs are defined relative to an arbitrary reference category (i.e., feature level). This function will loop over all feature levels (for a specified feature) to show how interpretation will be affected by choice of reference category. The resulting data frame will be a stacked result from `amce`, containing an additional `REFERENCE` column specifying which level of `variable` was used as the reference category. In unconstrained conjoint designs, only AMCEs for `variable` will vary by reference category; in constrained designs, AMCEs for any factor constrained by `variable` may also vary.

Users may desire to specify a `family` argument via `...`, which should be a "family" object such as `gaussian`. Sensible alternatives are `binomial` (for binary outcomes) and quasibinomial (for weighted survey data). See `family` for details. In such cases, effects are always reported on the link (not outcome) scale.

Value

A data frame of class “cj_amce”

See Also

`amce_diffs` `mm` `plot.cj_amce`

Examples

```
data("taxes")
# estimating AMCEs
amce(taxes, chose_plan ~ taxrate1 + taxrate2 + taxrate3 +
     taxrate4 + taxrate5 + taxrate6 + taxrev, id = ~ ID)


data("immigration")
# estimating AMCEs with constraints
amce(immigration, ChosenImmigrant ~ Gender + ReasonForApplication * CountryOfOrigin,
     id = ~CaseID)

# estimating average component interaction effects (AMCEs of feature combinations)
immigration$language_entry <- interaction(immigration$LanguageSkills,
                                          immigration$PriorEntry, sep = "_")
amce(immigration,ChosenImmigrant ~ language_entry, id = ~CaseID)

# balance testing example
plot(amce(immigration[!is.na(immigration$ethnocentrism),],
     ethnocentrism ~ Gender + Education + LanguageSkills, id = ~ CaseID))

# reference category sensitivity
x <- amce_by_reference(immigration, ChosenImmigrant ~ LanguageSkills + Education,
       variable = ~ LanguageSkills, id = ~ CaseID)
# plot
plot(x)
```

`amce_diffs` *Preference Heterogeneity Diagnostics*

Description

Tests for preference heterogeneity in conjoint experiments

Usage

```
amce_diffs(
  data,
  formula,
  by,
  id = ~0,
  weights = NULL,
  feature_order = NULL,
  feature_labels = NULL,
  level_order = c("ascending", "descending"),
  alpha = 0.05,
  ...
)

cj_anova(data, formula, id = NULL, weights = NULL, by = NULL, ...)

mm_diffs(
  data,
  formula,
  by,
  id = ~0,
  weights = NULL,
  feature_order = NULL,
  feature_labels = NULL,
  level_order = c("ascending", "descending"),
  alpha = 0.05,
  h0 = 0,
  ...
)
```

Arguments

`data`	A data frame containing variables specified in `formula`. All RHS variables should be factors; the base level for each will be used in estimation and for AMCEs the base level's AMCE will be NA. Optionally, this can instead be an object of class "survey.design" returned by `svydesign`.
`formula`	A formula specifying a model to be estimated. All variables should be factors; all levels across features should be unique.
`by`	A formula containing only RHS variables, specifying grouping factors over which to perform estimation. For `amce_diffs`, this can be a factor or something coercable to factor. For `mm_diffs`, differences are calculated against the base level of this variable.
`id`	Ignored.
`weights`	An (optional) RHS formula specifying a variable holding survey weights.
`feature_order`	An (optional) character vector specifying the names of feature (RHS) variables in the order they should be encoded in the resulting data frame.

`feature_labels`	A named list of "fancy" feature labels to be used in output. By default, the function looks for a "label" attribute on each variable in `formula` and uses that for pretty printing. This argument overrides those attributes or otherwise provides fancy labels for this purpose. This should be a list with names equal to variables on the righthand side of `formula` and character string values; arguments passed here override variable attributes.
`level_order`	A character string specifying levels (within each feature) should be ordered increasing or decreasing in the final output. This is mostly only consequential for plotting via `plot.cj_mm`, etc.
`alpha`	A numeric value indicating the significance level at which to calculate confidence intervals for the MMs (by default 0.95, meaning 95-percent CIs are returned).
`...`	Additional arguments to `amce`, `cj_freqs`, or `mm`.
`h0`	A numeric value specifying a null hypothesis value to use when generating z-statistics and p-values (only used for `mm_diffs`).

Details

`cj_anova` takes a model formula ("reduced" model) and generates a "full" model with two-way interactions between the variables specified in `by` and all RHS variables in `formula`, then computes an F-test comparing the two models, providing a test for whether preferences vary across levels of `by`. This is, in essence, a test of whether all such interaction coefficients are distinguishable from zero. (Because the test depends on overall model fit, not the coefficient variances, clustering is irrelevant.)

`mm_diffs` provides a data frame of differences in marginal means (literally differencing the results from `mm` across levels of `by`. This provides the clearest direct measure of preference differences from a conjoint design.

`amce_diffs` provides a data frame of differences in AMCEs (the coefficient on an interaction between each RHS factor and the variable in `by`). This provides an estimate of the difference in causal effects of each factor level relative to the baseline level (i.e., the difference in conditional AMCEs). This quantity is easily misinterpreted as the difference in preferences, which it is not. Rather it is a difference in the effect of the factor on preferences relative to the baseline/reference category of that feature. If preferences in the reference category differ across levels of `by`, the the difference in conditional AMCEs will have an unpredictable sign and significance, making differences in marginal means a more sensible quantity of interest. See `amce_by_reference` for a diagnostic.

Note: `amce_diffs` does not work with constrained designs. To obtain such differences, subset the design by constraints and calculate differences within each subset.

Value

`amce_diffs` and `mm_diffs` return a data frame similar to the one returned by `cj`, including a `BY` column (with the value "Difference") for easy merging with results returned by that function.

`cj_anova` returns an `anova` object.

Author(s)

Thomas J. Leeper <thosjleeper@gmail.com>

See Also

amce mm cj_freqs plot.cj_amce

Examples

```
data("immigration")
immigration$contest_no <- factor(immigration$contest_no)
# Test for heterogeneity by profile order
cj_anova(immigration, ChosenImmigrant ~ Gender + Education + LanguageSkills, by = ~ contest_no)

# Test for heterogeneity by CountryOfOrigin feature
cj_anova(immigration, ChosenImmigrant ~ Gender + Education, by = ~ CountryOfOrigin)


# Differences in MMs by Gender feature
mm_diffs(immigration, ChosenImmigrant ~ LanguageSkills + Education, ~ Gender, id = ~ CaseID)

# Differences in AMCEs by Gender feature (i.e., feature interactions)
amce_diffs(immigration, ChosenImmigrant ~ LanguageSkills + Education, ~ Gender, id = ~ CaseID)


# preferences differ for Male and Female immigrants with 'Broken English' ability
(m1 <- mm_diffs(immigration, ChosenImmigrant ~ LanguageSkills, ~ Gender, id = ~ CaseID))

# yet differences in conditional AMCEs  depend on the reference category
amce_diffs(immigration, ChosenImmigrant ~ LanguageSkills, ~ Gender, id = ~ CaseID)
immigration$LanguageSkills2 <- relevel(immigration$LanguageSkills, "Used Interpreter")
amce_diffs(immigration, ChosenImmigrant ~ LanguageSkills2, ~ Gender, id = ~ CaseID)

# while differences in MMs do not depend on the reference cateory
(m2 <- mm_diffs(immigration, ChosenImmigrant ~ LanguageSkills2, ~ Gender, id = ~ CaseID))
```

---

cj *Simple Conjoint Analyses and Visualization*

Description

Simple analyses of conjoint (factorial) experiments and visualization of results.

Usage

```
cj(
  data,
  formula,
  id = ~0,
  weights = NULL,
 estimate = c("amce", "frequencies", "mm", "amce_differences", "mm_differences"),
  feature_order = NULL,
  feature_labels = NULL,
  level_order = c("ascending", "descending"),
  by = NULL,
  ...
)
```

Arguments

`data`	A data frame containing variables specified in `formula`. All RHS variables should be factors; the base level for each will be used in estimation and for AMCEs the base level's AMCE will be zero. Optionally, this can instead be an object of class "survey.design" returned by `svydesign`.
`formula`	A formula specifying a model to be estimated. ; all levels across features should be unique. For `estimate = "amce"` in a constrained conjoint design, two-way interactions can be specified to handle constraints between factors in the design. These are detected automatically. Higher-order constraints are not allowed and interactions are ignored for all other values of `estimate` as constraints are irrelevant to those statistics.
`id`	An RHS formula specifying a variable holding respondent identifiers, to be used for clustering standard errors.
`weights`	An (optional) RHS formula specifying a variable holding survey weights.
`estimate`	A character string specifying an estimate type. Current options are average marginal component effects (or AMCEs, "amce", estimated via `amce`), display frequencies ("frequncies", estimated via `cj_freqs`), marginal means (or AMMs, "mm", estimated via `mm`), differences in MMs ("mm_differences", via `mm_diffs`), or differences in AMCEs ("amce_differences", via `amce_diffs`). Additional options may be made available in the future. Non-ambiguous abbreviations are allowed.
`feature_order`	An (optional) character vector specifying the names of feature (RHS) variables in the order they should be encoded in the resulting data frame.
`feature_labels`	A named list of "fancy" feature labels to be used in output. By default, the function looks for a "label" attribute on each variable in `formula` and uses that for pretty printing. This argument overrides those attributes or otherwise provides fancy labels for this purpose. This should be a list with names equal to variables on the righthand side of `formula` and character string values; arguments passed here override variable attributes.
`level_order`	A character string specifying levels (within each feature) should be ordered increasing or decreasing in the final output. This is mostly only consequential for plotting via `plot.cj_mm`, etc.
`by`	A formula containing only RHS variables, specifying grouping factors over which to perform estimation.
`...`	Additional arguments to `amce`, `cj_freqs`, `mm`, `mm_diffs`, or `amce_diffs`.

Details

The main function `cj` is a convenience function wrapper around the underlying estimation functions that provide for average marginal component effects (AMCEs), by default, via the `amce` function, marginal means (MMs) via the mm function, and display frequencies via cj_freqs and cj_props. Additional estimands may be supported in the future through their own functions and through the cj interface. Plotting is provided via ggplot2 for all types of estimates.

The only additional functionality provided by cj over the underlying functions is the by argument, which will perform operations on subsets of data, returning a single data frame. This can be useful, for example, for evaluating profile spillover effects and subgroup results, or in any situation where one might be inclined to use a for loop or lapply, calling cj repeatedly on subgroups.

Note: Some features of cregg (namely, the amce_diffs) function, or estimate = "amce_diff" here) only work with full factorial conjoint experiments. Designs involving two-way constraints between features are supported simply by expressing interactions between constrained terms in formula (again, except for amce_diffs). Higher-order constraints may be supported in the future.

Value

A data frame with special class to facilitate plotting (e.g., “cj_amce”, “cj_mm”, etc.)

Author(s)

Thomas J. Leeper <thosjleeper@gmail.com>

See Also

Functions: amce, mm, cj_freqs, mm_diffs, plot.cj_amce, cj_tidy Data: immigration, taxes

Examples

```
# load data
requireNamespace("ggplot2")
data("immigration")
immigration$contest_no <- factor(immigration$contest_no)
data("taxes")

# calculate MMs
f1 <- ChosenImmigrant ~ Gender + Education +
         LanguageSkills + CountryOfOrigin + Job + JobExperience +
         JobPlans + ReasonForApplication + PriorEntry
d1 <- cj(immigration, f1, id = ~ CaseID, estimate = "mm", h0 = 0.5)
# plot MMs
plot(d1, vline = 0.5)

# calculate MMs for survey-weighted data
d1 <- cj(taxes, chose_plan ~ taxrate1 + taxrate2 + taxrate3 +
         taxrate4 + taxrate5 + taxrate6 + taxrev, id = ~ ID,
         weights = ~ weight, estimate = "mm", h0 = 0.5)
# plot MMs
plot(d1, vline = 0.5)

# MMs split by profile number
stacked <- cj(immigration, f1, id = ~ CaseID,
              estimate = "mm", by = ~ contest_no)
```

```
## plot with grouping
plot(stacked, group = "contest_no", vline = 0.5, feature_headers = FALSE)

## plot with facetting
plot(stacked) + ggplot2::facet_wrap(~ contest_no, nrow = 1L)

# estimate AMCEs
d2 <- cj(immigration, f1, id = ~ CaseID)

# plot AMCEs
plot(d2)

## subgroup analysis
immigration$ethnosplit <- cut(immigration$ethnocentrism, 2)
x <- cj(na.omit(immigration), ChosenImmigrant ~ Gender + Education + LanguageSkills,
        id = ~ CaseID, estimate = "mm", h0 = 0.5, by = ~ ethnosplit)
plot(x, group = "ethnosplit", vline = 0.5)

# combinations of/interactions between features
immigration$language_entry <-
  interaction(immigration$LanguageSkills, immigration$PriorEntry, sep = "_")

## higher-order MMs for feature combinations
cj(immigration, ChosenImmigrant ~ language_entry,
   id = ~CaseID, estimate = "mm", h0 = 0.5)

## constrained designs
## in a constrained design, some cells are unobserved:
subset(cj_props(immigration, ~ Job + Education), Proportion == 0)
## MMs and AMCEs only use data from observed cells
## In `immigraation`, this means while the MM for `Job == "Janitor"` is an average
## across all levels of Education:
mm(subset(immigration, Job == "Janitor"), ChosenImmigrant ~ Education)
## the MM for `Job == "Doctor"` is an average across only 3 levels of education:
mm(subset(immigration, Job == "Doctor"), ChosenImmigrant ~ Education)
## Use `cj_props()` to see constraints:
subset(cj_props(immigration, ~ Job + Education), Job == "Doctor" & Proportion != 0)

## Substantively, the MM of "Doctor" might be higher than other levels of `Job`
## this could be due to the feature itself or due to the fact that it is constrained
## with a different subset of other feature levels than alternative levels of `Job`
## this may mean analysts want to report MMs (or AMCEs) only for the unconstrained levels:
elev <- c("Two-Year College", "College Degree", "Graduate Degree")
jlev <- c("Financial Analyst", "Computer Programmer", "Research Scientist", "Doctor")
mm(subset(immigration, Education %in% elev), ChosenImmigrant ~ Job)
mm(subset(immigration, Job %in% jlev), ChosenImmigrant ~ Education)
## or, present estimates excluding constrained levels:
mm(subset(immigration, !Education %in% elev), ChosenImmigrant ~ Job)
mm(subset(immigration, !Job %in% jlev), ChosenImmigrant ~ Education)
```

cj_df — *Create a "cj_df" data frame*

Description

A simple data frame extension that preserves attributes during subsetting operations.

Usage

```
cj_df(x)

## S3 method for class 'data.frame'
cj_df(x)

## S3 method for class 'cj_df'
x[i, j, drop = FALSE]
```

Arguments

x	A data frame
i	See [.data.frame
j	See [.data.frame
drop	Ignored.

Value

An data frame with additional "cj_df" class, which has subsetting methods that preserve variables attributes.

Examples

```
x1 <- data.frame(a = 1:3, b = 4:6)
attr(x1$a, "label") <- "Variable A"

# cj_df() returns a data frame
inherits(x1, "data.frame")
class(x1)

# attributes dropped for data frames
attr(x1[1:2,]$a, "label")

# attributes preserved with a cj_df
attr(cj_df(x1)[1:2,]$a, "label")
```

cj_freqs *Conjoint feature frequencies*

Description

Tabulate and visualize conjoint features, and their display frequencies and proportions

Usage

```
cj_freqs(
  data,
  formula,
  id = NULL,
  weights = NULL,
  feature_order = NULL,
  feature_labels = NULL,
  level_order = c("ascending", "descending"),
  ...
)

cj_props(data, formula, id, weights = NULL, margin = NULL, ...)

cj_table(
  data,
  formula,
  feature_order = NULL,
  feature_labels = NULL,
  level_order = c("ascending", "descending"),
  include_reference = FALSE,
  ...
)
```

Arguments

`data`	A data frame containing variables specified in `formula`. All RHS variables should be factors; all levels across features should be unique.
`formula`	An RHS formula specifying conjoint features to tabulate. All RHS variables should be factors; all levels across features should be unique.
`id`	An RHS formula specifying a variable holding respondent identifiers, to be used for clustering standard errors. By default, data are unclustered.
`weights`	An (optional) RHS formula specifying a variable holding survey weights.
`feature_order`	An (optional) character vector specifying the names of feature (RHS) variables in the order they should be encoded in the resulting data frame.
`feature_labels`	A named list of “fancy” feature labels to be used in output. By default, the function looks for a “label” attribute on each variable in `formula` and uses that for

pretty printing. This argument overrides those attributes or otherwise provides fancy labels for this purpose. This should be a list with names equal to variables on the righthand side of `formula` and character string values; arguments passed here override variable attributes.

`level_order` A character string specifying levels (within each feature) should be ordered increasing or decreasing in the final output. This is mostly only consequential for plotting via `plot.cj_mm`, etc.

`...` Ignored.

`margin` A numeric value passed to `prop.table`. If `NULL` overall proportions are calculated.

`include_reference`
A logical indicating whether to include a "reference" column that indicates whether a feature level is the reference category for that feature. Default is `FALSE`.

Details

These functions provide related but slightly different functionality. `cj_table` simply creates a data frame of features and their levels, which is useful for printing. `cj_props` provides tidy proportion tables to examine cross-feature restrictions in conjoint designs that are not equally randomized. This enables, for example, tabulation and visualization of complete restrictions (where combinations of two or more features are not permitted), as well as calculation of AMCEs for constrained designs appropriately weighted by the display proportions for particular combinations of features.

`cj_freqs` provides *marginal* display frequencies, which are a descriptive check on the presentation of individual conjoint features (for example, to ensure equal or intentionally unequal appearance of levels). This is mostly useful for plotting functionality provided in `plot.cj_freqs`, which provides barcharts for the frequency with which each level of each feature was presented.

Value

A data frame of class "cj_freqs", "cj_props", etc.

See Also

`plot.cj_mm`

Examples

```
data(immigration)
# identify all levels
cj_table(immigration, ~ Gender + Education + LanguageSkills)
cj_table(immigration, ~ Gender + Education + LanguageSkills, include_ref = TRUE)

# display frequencies
(f <- cj_freqs(immigration, ~ Gender + Education + LanguageSkills, id = ~ CaseID))

# restrictions
## check display proportions
cj_props(immigration, ~ Job, id = ~ CaseID)
```

```
## check which combinations were not allowed
subset(cj_props(immigration, ~ Job + Education, id = ~ CaseID), Proportion == 0)


# plotting
(p <- plot(f))

## change ggplot2 theme
p + ggplot2::theme_bw()

## monochrome bars
p + ggplot2::scale_fill_manual(values = rep("black", 9)) +
  ggplot2::theme(legend.position = "none")
```

`cj_tidy` *Tidy a conjoint dataset*

Description

Coerce a “wide” conjoint dataset into a “long”/“tidy” one for use with cregg

Usage

```
cj_tidy(data, profile_variables, task_variables, id)
```

Arguments

`data`	A data frame containing a conjoint dataset in “wide” format (see Details).
`profile_variables`	A named list of two-element lists capturing profile-specific variables (either features, or profile-specific outcomes, like rating scales). For each element in the list, the first element contains vectors of feature variable names for the first profile in each decision task (hereafter, profile “A”) and the second element contains vectors of feature variable names for the second profile in each decision task (hereafter, profile “B”). Variables can be specified as character strings or an RHS formula. The names at the highest level are used to name variables in the long/tidy output.
`task_variables`	A named list of vectors of variables constituting task-level variables (i.e., variables that differ by task but not across profiles within a task). Variables can be specified as character strings or an RHS formula. These could be outcome variables, response times, etc.
`id`	An RHS formula specifying a variable holding respondent identifiers.

Details

A conjoint survey typically comes to the analyst in a "wide" form, where the number of rows is equal to the number of survey respondents and columns represent choices and features for each choice task and task profile. For example, a design with 1000 respondents and five forced-choice decision tasks, with 6 features each, will have 1000 rows and 5x2x6 feature columns, plus five forced-choice outcome variable columns recording which alternative was selected for each task. To analyse these data, the data frame needs to be reshaped to "long" or "tidy" format, with 1000x5x2 rows, six feature columns, and one outcome column. Multiple outcomes or other task-specific variables would increase the number of columns in the result, as will respondent-varying characteristics which need to be replicated across each decision task and profile.

This a complex operation because variables vary at three levels: respondent, task, and profile. Thus the reshape is not a simple wide-to-long transformation. It instead requires two reshaping steps, one to create a task-level dataset and a further one to create a profile-level dataset. `cj_tidy` performs this tidying in two steps, through a single function with an easy-to-use API. Users can specify variable names in the `wide` format using either character vectors of righthand-side (RHS) formulae. They are equivalent but depending on the naming of variables, character vectors can be easier to specify (e.g., using regular expressions for pattern matching).

Particular care is needed to decide whether a particular set of "wide" columns belong in `profile_variables` or `task_variables`. This especially applies to outcomes variables. If a variable in the original format records *which* of the two profiles was chosen (e.g., "left" and "right"), it should go in `task_variables`. If it records whether a profile was chosen (e.g., for each task there is a "left_chosen" and "right_chosen" variable), then both variables should go in `profile_variables` as they vary at the profile level. Similarly, one needs to be careful with the output of `cj_tidy` to ensure that a task-level variable is further recoded to encode which alternative was selected (see examples).

Users may find that it is easier to recode features *after* using `cj_tidy` rather than before, as it requires recoding only a number of variables equal to the number of features in the design, rather than recoding all "wide" feature columns before reshaping. Again, however, care should be taken that these variables encode information in the same way so that stacking does not produce a loss of information.

Finally, `data` should not use the variable names "task", "pair", or "profile", which are the names of metadata columns created by reshaping.

Value

A data frame with rows equal to the number of respondents times the number of tasks times the number of profiles (fixed at 2), to be fed into any other function in the package. The columns will include the names of elements in `profile_variables` and `task_variables`, and `id`, along with an indicator `task` (from 1 to the number of tasks), `pair` (an indicator for each task pair from 1 to the number of pairs), `profile` (a fator indicator for profile, either "A" or "B"), and any other respondent-varying covariates not specified. As such, respondent-varying variables do not need to be specified to `cj_tidy` at all.

The returned data frame carries an additional S3 class ("cj_df") with methods that preserve column attributes. See `cj_df`.

See Also

cj, cj_df

Examples

```
## Not run:
data("wide_conjoint")

# character string interface
## profile_variables
list1 <- list(
 feature1 = list(
     names(wide_conjoint)[grep("^feature1.{1}1", names(wide_conjoint))],
     names(wide_conjoint)[grep("^feature1.{1}2", names(wide_conjoint))]
 ),
 feature2 = list(
     names(wide_conjoint)[grep("^feature2.{1}1", names(wide_conjoint))],
     names(wide_conjoint)[grep("^feature2.{1}2", names(wide_conjoint))]
 ),
 feature3 = list(
     names(wide_conjoint)[grep("^feature3.{1}1", names(wide_conjoint))],
     names(wide_conjoint)[grep("^feature3.{1}2", names(wide_conjoint))]
 ),
 rating = list(
     names(wide_conjoint)[grep("^rating.+1", names(wide_conjoint))],
     names(wide_conjoint)[grep("^rating.+2", names(wide_conjoint))]
 )
)
## task variables
list2 <- list(choice = paste0("choice_", letters[1:4]),
              timing = paste0("timing_", letters[1:4]))

# formula interface
## profile_variables
list1 <- list(
   feature1 = list(
       ~ feature1a1 + feature1b1 + feature1c1 + feature1d1,
       ~ feature1a2 + feature1b2 + feature1c2 + feature1d2
   ),
   feature2 = list(
       ~ feature2a1 + feature2b1 + feature2c1 + feature2d1,
       ~ feature2a2 + feature2b2 + feature2c2 + feature2d2
   ),
   feature3 = list(
       ~ feature3a1 + feature3b1 + feature3c1 + feature3d1,
       ~ feature3a2 + feature3b2 + feature3c2 + feature3d2
   ),
   rating = list(
       ~ rating_a1 + rating_b1 + rating_c1 + rating_d1,
       ~ rating_a2 + rating_b2 + rating_c2 + rating_d2
   )
)
```

```
# task variables
list2 <- list(choice = ~ choice_a + choice_b + choice_c + choice_d,
              timing = ~ timing_a + timing_b + timing_c + timing_d)


# perform reshape
str(long <- cj_tidy(wide_conjoint,
                    profile_variables = list1,
                    task_variables = list2,
                    id = ~ respondent))
stopifnot(nrow(long) == nrow(wide_conjoint)*4*2)

# recode outcome so it is coded sensibly
long$chosen <- ifelse((long$profile == "A" & long$choice == 1) |
                       (long$profile == "B" & long$choice == 2), 1, 0)
# use for analysis
cj(long, chosen ~ feature1 + feature2 + feature3, id = ~ respondent)

## End(Not run)
```

`immigration` *Immigration Conjoint Experiment Dataset from Hainmueller et. al. (2014)*

Description

A dataset containing the results of a conjoint survey of a representative sample of American adults who were asked to choose which hypothetical immigrants they think should be admitted into the United States. Each row corresponds to a single profile presented to the respondent. The dataset results from a mostly full factorial design with restrictions on two combinations of features. (1) Profile immigrants from ‘`CountryOfOrigin`’ “India”, “Germany”, “France”, “Mexico”, “Philippines”, and “Poland” could be paired only with ‘`ReasonForApplication`’ “Seek better job” or “Reunite with family”; profiles from the remaining countries could be paired with any ‘`ReasonForApplication`’. (2) Profile immigrants with ‘`Job`’ “Financial Analyst”, “Computer Programmer”, “Research Scientist”, or “Doctor” could not be paired with ‘`Education`’ levels “No Formal”, “4th Grade”, “8th Grade”, or “High School”. All other features were fully randomized against all other features.

Usage

```
data(immigration)
```

Format

A data frame (with additional “cj_df” class) with 13960 observations on the following 16 variables.

‘`CaseID`’ a numeric vector indicating the respondent to which the particular profile corresponds

‘`contest_no`’ a numeric vector indicating the number of the task to which the profile corresponds

‘`Education`’ a factor with levels “No formal”, “4th grade”, “8th grade”, “High school”, “Two-year college”, “college Degree”, “Graduate degree”

'Gender' a factor with levels "Female", "Male"

'CountryOfOrigin' a factor with levels "India", "Germany", "France", "Mexico", "Philippines", "Poland", "China", "Sudan", "Somalia", "Iraq"

'ReasonForApplication' a factor with levels "Reunite with family", "Seek better job", "Escape persecution"

'Job' a factor with levels "Janitor", "Waiter", "Child care provider", "Gardener", "Financial analyst", "Construction worker", "Teacher", "Computer programmer", "Nurse", "Research scientist", "Doctor"

'JobExperience' a factor with levels "None", "1-2 years", "3-5 years", "5+ years"

'JobPlans' a factor with levels "Will look for work", "Contract with employer", "Interviews with employer", "No plans to look for work"

'PriorEntry' a factor with levels "Never", "Once as tourist", "Many times as tourist", "Six months with family", "Once w/o authorization"

'LanguageSkills' a factor with levels "Fluent English", "Broken English", "Tried English but unable", "Used interpreter"

'ChosenImmigrant' a numeric vector denoting whether the immigrant profile was selected

'ethnocentrism' a numeric vector

'profile' a numeric vector giving the profile number

'LangPos' a numeric vector

'PriorPos' a numeric vector

Note

This is a modified version of the 'hainmueller' dataset available from the cjoint package.

Source

Hainmueller, J., Hopkins, D., and Yamamoto T. 2014. "Causal Inference in Conjoint Analysis: Understanding Multi-Dimensional Choices via Stated Preference Experiments." *Political Analysis* 22(1): 1-30. http://doi.org/10.1093/pan/mpt024

See Also

cj taxes cj_df

Examples

```
data("immigration")

# view constraints between features
subset(cj_props(immigration, ~ Job + Education, id = ~ CaseID), Proportion == 0)
subset(cj_props(immigration, ~ ReasonForApplication + CountryOfOrigin,
                id = ~ CaseID), Proportion == 0)

# AMCEs with interactions for constraints
```

```
f1 <- ChosenImmigrant ~ Gender + Education * Job +
        LanguageSkills + CountryOfOrigin * ReasonForApplication +
        JobExperience + JobPlans + PriorEntry
cj(immigration, f1, id = ~ CaseID)
```

---

mm *Marginal Means*

Description

Calculate (descriptive) marginal means (MMs) from a conjoint design

Usage

```
mm(
  data,
  formula,
  id = ~0,
  weights = NULL,
  feature_order = NULL,
  feature_labels = NULL,
  level_order = c("ascending", "descending"),
  alpha = 0.05,
  h0 = 0,
  ...
)
```

Arguments

`data`	A data frame containing variables specified in `formula`. All RHS variables should be factors.
`formula`	A formula specifying an outcome (LHS) and conjoint features (RHS) to describe. All variables should be factors; all levels across features should be unique, with constraints specified with an asterisk (*) between features, as in `amce`.
`id`	An RHS formula specifying a variable holding respondent identifiers, to be used for clustering standard errors. By default, data are unclustered.
`weights`	An (optional) RHS formula specifying a variable holding survey weights.
`feature_order`	An (optional) character vector specifying the names of feature (RHS) variables in the order they should be encoded in the resulting data frame.
`feature_labels`	A named list of "fancy" feature labels to be used in output. By default, the function looks for a "label" attribute on each variable in `formula` and uses that for pretty printing. This argument overrides those attributes or otherwise provides fancy labels for this purpose. This should be a list with names equal to variables on the righthand side of `formula` and character string values; arguments passed here override variable attributes.

`level_order`	A character string specifying levels (within each feature) should be ordered increasing or decreasing in the final output. This is mostly only consequential for plotting via `plot.cj_mm`, etc.
`alpha`	A numeric value indicating the significance level at which to calculate confidence intervals for the MMs (by default 0.95, meaning 95-percent CIs are returned).
`h0`	A numeric value specifying a null hypothesis value to use when generating z-statistics and p-values.
`...`	Ignored.

Details

`mm` provides descriptive representations of conjoint data as marginal means (MMs), which represent the mean outcome across all appearances of a particular conjoint feature level, averaging across all other features. In forced choice conjoint designs with two profiles per choice task, MMs by definition average 0.5 with values above 0.5 indicating features that increase profile favorability and values below 0.5 indicating features that decrease profile favorability. For continuous outcomes, MMs can take any value in the full range of the outcome.

But note that if feature levels can co-occur, such that both alternatives share a feature level, then the MMs on forced choice outcomes are bounded by the probability of co-occurrence (as a lower bound) and 1 minus that probability as an upper bound.

Plotting functionality is provided in `plot.cj_mm`.

Value

A data frame of class "cj_mm"

See Also

`mm_diffs` `plot.cj_mm`

Examples

```
data(immigration)
# marginal means
mm(immigration, ChosenImmigrant ~ Gender + Education + LanguageSkills,
   id = ~ CaseID, h0 = 0.5)

# higher-order marginal means with feature interactions
immigration$language_entry <-
  interaction(immigration$LanguageSkills, immigration$PriorEntry, sep = "_")
mm(immigration, ChosenImmigrant ~ language_entry,
   id = ~CaseID)
```

`plot.cj_amce` *Plot AMCE estimates, MM descriptives, and frequency plots*

Description

ggplot2-based plotting of conjoint AMCEs estimates and MMs, and differences

Usage

```
## S3 method for class 'cj_amce'
plot(
  x,
  group = attr(x, "by"),
  feature_headers = TRUE,
  header_fmt = "(%s)",
  size = 1,
  xlab = "Estimated AMCE",
  ylab = "",
  legend_title = if (is.null(group)) "Feature" else group,
  legend_pos = "bottom",
  xlim = NULL,
  vline = 0,
  vline_color = "gray",
  theme = ggplot2::theme_bw(),
  ...
)

## S3 method for class 'cj_diffs'
plot(
  x,
  group = attr(x, "by"),
  feature_headers = TRUE,
  header_fmt = "(%s)",
  size = 1,
  xlab = "Estimated Difference",
  ylab = "",
  legend_title = if (is.null(group)) "Feature" else group,
  legend_pos = "bottom",
  xlim = NULL,
  vline = 0,
  vline_color = "gray",
  theme = ggplot2::theme_bw(),
  ...
)

## S3 method for class 'cj_freqs'
plot(
  x,
  group = attr(x, "by"),
  feature_headers = TRUE,
  header_fmt = "(%s)",
  xlab = "",
  ylab = "Frequency",
  legend_title = if (is.null(group)) "Feature" else group,
  legend_pos = "bottom",
  theme = ggplot2::theme_bw(),
  ...
)

## S3 method for class 'cj_mm'
plot(
  x,
  group = attr(x, "by"),
  feature_headers = TRUE,
  header_fmt = "(%s)",
  size = 1,
  xlab = "Marginal Mean",
  ylab = "",
  legend_title = if (is.null(group)) "Feature" else group,
  legend_pos = "bottom",
  xlim = NULL,
  vline = 0,
  vline_color = "gray",
  theme = ggplot2::theme_bw(),
  ...
)
```

Arguments

`x`	A data frame returned from `cj` or `mm`.
`group`	Optionally a character string specifying a grouping factor. This is useful when, for example, subgroup analyses or comparing AMCEs for different outcomes. An alternative is to use `facet_wrap` for faceted graphics.
`feature_headers`	A logical indicating whether to include headers for each feature to visually separate levels for each feature (beyond the color palette).
`header_fmt`	A character string specifying a `fmt` argument to `sprintf`, which will be used when generating the feature headers (if `feature_headers = TRUE`).
`size`	A numeric value specifying point size in `geom_point`.
`xlab`	A label for the x-axis
`ylab`	A label for the y-axis
`legend_title`	A character string specifying a label for the legend.
`legend_pos`	An argument forwarded to the `legend.position` argument in `theme`.

`xlim`	A two-element number vector specifying limits for the x-axis. If `NULL`, a default value is calculated from the data.
`vline`	Optionally, a numeric value specifying an x-intercept for a vertical line. This can be useful in distinguishing the midpoint of the estimates (e.g., a zero line for AMCEs).
`vline_color`	A character string specifying a color for the `vline`.
`theme`	A ggplot2 theme object
`...`	Ignored.

Details

These are convenience functions for quickly plotting results from cregg. Because `plot` returns ggplot2 objects, these are easily manipulated using standard ggplot2 operations.

Note that ggplot2, by default, sorts factors (like feature names here) in what might be the opposite order of what you would expect and in the opposite order that cregg functions sort their output.

Value

A ggplot2 object

See Also

`amce`, `mm`

Examples

```
require("ggplot2")
# load data
data("immigration")
immigration$contest_no <- factor(immigration$contest_no)

# calculate MMs
d1 <- mm(immigration, ChosenImmigrant ~ Gender + Education +
         LanguageSkills + CountryOfOrigin + Job + JobExperience +
         JobPlans + ReasonForApplication + PriorEntry, id = ~ CaseID)

# plot MMs
## simple plot
(p <- plot(d1, vline = 0.5))

## gridlines to aid interpretation
p + ggplot2::theme_grey()

## monochrome bars
p + scale_color_manual(values = rep("black", 9))

## plot with estimates shown as text labels
p + ggplot2::geom_text(
  aes(label = sprintf("%0.2f (%0.2f)", estimate, std.error)),
  colour = "black", position = position_nudge(y = .5)
)

## plot with facetting by feature
plot(d1, feature_headers = FALSE) +
  ggplot2::facet_wrap(~feature, ncol = 1L,
                      scales = "free_y", strip.position = "right")

# MMs split by profile number
stacked <- cj(immigration, ChosenImmigrant ~ Gender +
              Education + LanguageSkills + CountryOfOrigin + Job + JobExperience +
              JobPlans + ReasonForApplication + PriorEntry, id = ~ CaseID,
              estimate = "mm", by = ~ contest_no)

## plot with grouping
plot(stacked, group = "contest_no", feature_headers = FALSE)

## plot with facetting
plot(stacked) + ggplot2::facet_wrap(~contest_no, nrow = 1L)

## plot with shapes instead of colors for groups
plot(stacked, group = "contest_no", vline = 0.5) +
 aes(shape = contest_no) + # map group to `shape` aesthetic
 scale_shape_manual(values=c(1, 2, 3, 4, 5)) +
 scale_colour_manual(values=rep("black", 5))

# estimate AMCEs over different subsets of data
reasons12 <- subset(
  immigration, ReasonForApplication %in% levels(ReasonForApplication)[1:2]
)
d2_1 <- cj(immigration, ChosenImmigrant ~ CountryOfOrigin, id = ~ CaseID)
d2_2 <- cj(reasons12, ChosenImmigrant ~ CountryOfOrigin, id = ~ CaseID,
           feature_labels = list(CountryOfOrigin = "Country Of Origin"))
d2_1$reasons <- "1,2,3"
d2_2$reasons <- "1,2"
plot(rbind(d2_1, d2_2), group = "reasons")
```

`taxes` *Tax Preference Conjoint Experiment Dataset from Ballard-Rosa et al. (2016)*

Description

A dataset containing the results of a fully randomized conjoint survey of a representative sample of 2000 American adults who were asked to choose between alternative tax rate policies. Variables '`taxrate1`'-'`taxrate6`' refer to tax rates for different income brackets and '`taxrev`' refers to levels of total tax revenue.

Usage

```
data(taxes)
```

Format

A data frame (with additional "cj_df" class) with 32000 observations on the following 13 variables. Each row corresponds to a single profile presented to a respondent.

'`chose_plan`' A numeric vector denoting whether the immigrant profile was selected (=1) or not (=0).

'`taxrate1`' An experimental factor with levels "<10k: 0%", "<10k: 5%", "<10k: 15%", "<10k: 25%".

'`taxrate2`' An experimental factor with levels "10-35k: 5%", "10-35k: 15%", "10-35k: 25%", "10-35k: 35%".

'`taxrate3`' An experimental factor with levels "35-85k: 5%", "35-85k: 15%", "35-85k: 25%", "35-85k: 35%".

'`taxrate4`' An experimental factor with levels "85-175k: 5%", "85-175k: 15%", "85-175k: 25%", "85-175k: 35%".

'`taxrate5`' An experimental factor with levels "175-375k: 5%", "175-375k: 15%", "175-375k: 25%", "175-375k: 35%", "175-375k: 45%".

'`taxrate6`' An experimental factor with levels ">375k: 5%", ">375k: 15%", ">375k: 25%", ">375k: 35%", ">375k: 45%", ">375k: 55%".

'`taxrev`' An experimental factor with levels "<75%", "75-95%", "95-105%", "105-125%", ">125%".

'`inequality_aversion`' A covariate specifying whether respondent is inequality averse (=1) or not (=0).

'`taxes_harm_economy`' A covariate specifying whether respondent believes taxes harm the economy (=1) or not (=0).

'`partyid`' A factor specifying the respondent's party identification; one of "Independent", "Democrat", "Republican".

'`ID`' A numeric vector indicating the respondent to which the profile corresponds.

'`weight`' A numeric vector containing survey weights.

Source

Ballard-Rosa, Cameron, Lucy Martin, and Kenneth Scheve. 2016. "The Structure of American Income Tax Policy Preferences." *The Journal of Politics* 79(1): 1-16. `http://doi.org/10.1086/687324`

See Also

`cj` `immigration` `cj_df`

Examples

```
data("taxes")
f1 <- chose_plan ~ taxrate1 + taxrate2 + taxrate3 +
       taxrate4 + taxrate5 + taxrate6 + taxrev
cj(taxes, f1, id = ~ ID, weights = ~     weight)
```

`wide_conjoint` *Example of a raw, "wide" conjoint dataset to demonstrate functionality of* `cj_tidy`

Description

A simulated dataset containing 100 respondents' responses to four decision tasks (a,b,c,d) involving a forced choice between two alternative profiles, described by three features (1,2,3), as well as a secondary rating-scale outcome and a response time measure, along with two respondent-varying covariates. This is used in testing and examples within the package.

Usage

```
data(wide_conjoint)
```

Format

A data frame with 100 observations on the following variables:

‘`respondent`’	a numeric vector indicating the respondent identifier
‘`feature1a1`’	Feature 1 for task A left profile, a factor
‘`feature1b1`’	Feature 1 for task B left profile, a factor
‘`feature1c1`’	Feature 1 for task C left profile, a factor
‘`feature1d1`’	Feature 1 for task D left profile, a factor
‘`feature1a2`’	Feature 1 for task A right profile, a factor
‘`feature1b2`’	Feature 1 for task B right profile, a factor
‘`feature1c2`’	Feature 1 for task C right profile, a factor
‘`feature1d2`’	Feature 1 for task D right profile, a factor
‘`feature2a1`’	Feature 2 for task A left profile, a factor
‘`feature2b1`’	Feature 2 for task B left profile, a factor
‘`feature2c1`’	Feature 2 for task C left profile, a factor
‘`feature2d1`’	Feature 2 for task D left profile, a factor
‘`feature2a2`’	Feature 2 for task A right profile, a factor
‘`feature2b2`’	Feature 2 for task B right profile, a factor
‘`feature2c2`’	Feature 2 for task C right profile, a factor

‘`feature2d2`’ Feature 2 for task D right profile, a factor

‘`feature3a1`’ Feature 3 for task A left profile, a factor

‘`feature3b1`’ Feature 3 for task B left profile, a factor

‘`feature3c1`’ Feature 3 for task C left profile, a factor

‘`feature3d1`’ Feature 3 for task D left profile, a factor

‘`feature3a2`’ Feature 3 for task A right profile, a factor

‘`feature3b2`’ Feature 3 for task B right profile, a factor

‘`feature3c2`’ Feature 3 for task C right profile, a factor

‘`feature3d2`’ Feature 3 for task D right profile, a factor

‘`choice_a`’ outcome for task A indicating which profile was chosen, randomly 1 or 2, each equally probable

‘`choice_b`’ outcome for task B indicating which profile was chosen, randomly 1 or 2, each equally probable

‘`choice_c`’ outcome for task C indicating which profile was chosen, randomly 1 or 2, each equally probable

‘`choice_d`’ outcome for task D indicating which profile was chosen, randomly 1 or 2, each equally probable

‘`rating_a1`’ rating for task A left profile, random variable between 1 and 7, uniformly distributed

‘`rating_a2`’ rating for task A right profile, random variable between 1 and 7, uniformly distributed

‘`rating_b1`’ rating for task B left profile, random variable between 1 and 7, uniformly distributed

‘`rating_b2`’ rating for task B right profile, random variable between 1 and 7, uniformly distributed

‘`rating_c1`’ rating for task C left profile, random variable between 1 and 7, uniformly distributed

‘`rating_c2`’ rating for task C right profile, random variable between 1 and 7, uniformly distributed

‘`rating_d1`’ rating for task D left profile, random variable between 1 and 7, uniformly distributed

‘`rating_d2`’ rating for task D right profile, random variable between 1 and 7, uniformly distributed

‘`timing_a`’ timing for task A in seconds, random draws from a beta distribution (2,5) times 10

‘`timing_b`’ timing for task A in seconds, random draws from a beta distribution (2,5) times 10

‘`timing_c`’ timing for task A in seconds, random draws from a beta distribution (2,5) times 10

‘`timing_d`’ timing for task A in seconds, random draws from a beta distribution (2,5) times 10

‘`covariate1`’ random draws from a uniform distribution between -1 and 1

‘`covariate2`’ random draws from the set of 1 and 2

See Also

`cj_tidy` `cj`

Examples

```
## Not run:
data("wide_conjoint")
# feature_variables
list1 <- list(
 feature1 = list(
     names(wide_conjoint)[grep("^feature1.{1}1", names(wide_conjoint))],
     names(wide_conjoint)[grep("^feature1.{1}2", names(wide_conjoint))]
 ),
 feature2 = list(
     names(wide_conjoint)[grep("^feature2.{1}1", names(wide_conjoint))],
     names(wide_conjoint)[grep("^feature2.{1}2", names(wide_conjoint))]
 ),
 feature3 = list(
     names(wide_conjoint)[grep("^feature3.{1}1", names(wide_conjoint))],
     names(wide_conjoint)[grep("^feature3.{1}2", names(wide_conjoint))]
 ),
 rating = list(
     names(wide_conjoint)[grep("^rating.+1", names(wide_conjoint))],
     names(wide_conjoint)[grep("^rating.+2", names(wide_conjoint))]
 )
)
# task variables
list2 <- list(choice = paste0("choice_", letters[1:4]),
              timing = paste0("timing_", letters[1:4]))
str(cj_tidy(wide_conjoint, profile_variables = list1, task_variables = list2, id = ~ respondent))

## End(Not run)
```

Index