Package: choicedata (via r-universe)

October 21, 2026

Type Package

Title Working with Choice Data

Version 0.2.1

Description Offers a set of objects tailored to simplify working with choice data. It enables the computation of choice probabilities and the likelihood of various types of choice models based on given data.

License GPL (>= 3)

Encoding UTF-8

Language en-US

URL https://github.com/loelschlaeger/choicedata, https://loelschlaeger.de/choicedata/

BugReports https://github.com/loelschlaeger/choicedata/issues

Suggests AER, mlogit, mvtnorm, testthat (>= 3.0.0)

Config/testthat/edition 3

Imports checkmate, cli, dplyr, Formula, optimizeR (>= 1.2.1), oeli (>= 0.7.9), Rdpack, Rcpp, rlang, tibble, tidyr, utils

LinkingTo oeli (>= 0.7.9), Rcpp, RcppArmadillo

RdMacros Rdpack

LazyData true

LazyDataCompression xz

Depends R (>= 4.1.0)

Config/roxygen2/version 8.1.0

NeedsCompilation yes

Author Lennart Oelschläger [aut, cre] (ORCID: <https://orcid.org/0000-0001-5421-9313>)

Maintainer Lennart Oelschläger <oelschlaeger.lennart@gmail.com>

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

Date/Publication 2026-10-20 14:40:02 UTC

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

RemoteRef HEAD

RemoteSha 8c240c27ff6adf70f52812c038e13cacd3deaa57

Contents

choice_alternatives . . . 2
choice_covariates . . . 4
choice_data . . . 7
choice_effects . . . 10
choice_formula . . . 12
choice_identifiers . . . 15
choice_likelihood . . . 17
choice_parameters . . . 19
choice_preferences . . . 23
choice_probabilities . . . 24
choice_responses . . . 32
compute_P . . . 34
lichess_berserk_choice . . . 35
train_test . . . 36
water_conservation_choice . . . 37
wind_power_choice . . . 38

Index **40**

`choice_alternatives` *Define choice alternatives*

Description

The `choice_alternatives` object defines the set of choice alternatives.

Usage

```
choice_alternatives(
  J = length(alternatives),
  alternatives = LETTERS[1:J],
  base = NULL,
  ordered = FALSE
)

## S3 method for class 'choice_alternatives'
print(x, ...)
```

Arguments

J	[integer(1)] The number ≥ 2 of choice alternatives.
alternatives	[character(J)] Unique labels for the choice alternatives.
base	[character(1)] The name of the base alternative for alternative-constant covariates, see details. If NULL (default), it is set to the first retained alternative. This is the first supplied alternative when ordered = TRUE and the first alphabetically sorted alternative otherwise.
ordered	[logical(1)] Should the supplied order of alternatives be preserved and treated as an intrinsic ranking (for ordered response models)? When TRUE, the alternatives are kept in the given order. Otherwise, they are sorted alphabetically.
x	[choice_alternatives] A choice_alternatives object.
...	Currently not used.

Value

An object of class choice_alternatives, i.e. a character vector of the choice alternatives with attributes:

J The number of choice alternatives.

base The name of the base alternative.

ordered Do the alternatives encode an inherent ordering?

Base alternative

The full set of coefficients for covariates that are constant across alternatives (including alternative-specific constants) is not identified. To achieve identifiability, the coefficient of alternative base is fixed to zero. The other coefficients then have to be interpreted with respect to base. The base alternative is marked with a * when printing a choice_alternatives object.

Examples

```
### unordered choice alternatives: heating
choice_alternatives(
  J = 3,
  alternatives = c("gas", "electricity", "oil"),
  base = "gas"
)

### ordered choice alternatives: opinion
choice_alternatives(
  J = 4,
```

```
  alternatives = c("very good", "good", "neither good nor bad", "bad"),
  ordered = TRUE
)
```

---

choice_covariates *Define choice covariates*

Description

The choice_covariates object defines the choice model covariates.

- generate_choice_covariates() samples covariates.
- covariate_names() gives the covariate names for given choice_effects.
- design_matrices() builds design matrices.

Usage

```
choice_covariates(
  data_frame,
  format = "wide",
  column_decider = "deciderID",
  column_occasion = NULL,
  column_alternative = NULL,
  column_ac_covariates = NULL,
  column_as_covariates = NULL,
  delimiter = "_",
  cross_section = is.null(column_occasion)
)

generate_choice_covariates(
  choice_effects = NULL,
  choice_identifiers = generate_choice_identifiers(N = 100),
  labels = if (is.null(choice_effects)) {
     character()
 } else {

    covariate_names(choice_effects)
 },
  n = nrow(choice_identifiers),
  marginals = list(),
  correlation = diag(length(labels)),
  verbose = FALSE,
  delimiter = "_"
)

covariate_names(choice_effects)
```

```
design_matrices(
  x,
  choice_effects,
  choice_identifiers = extract_choice_identifiers(x)
)
```

Arguments

`data_frame`	[`data.frame`] Contains the choice covariates.
`format`	[`character(1)`] Format of `data_frame`. Use `"wide"` when covariates for all alternatives are stored in a single row per occasion and `"long"` when each alternative forms a separate row.
`column_decider`	[`character(1)`] Column name with decider identifiers.
`column_occasion`	[`character(1)` | `NULL`] Column name with occasion identifiers. Set to `NULL` for cross-sectional data.
`column_alternative`	[`character(1)` | `NULL`] Column name with alternative identifiers when `format = "long"`.
`column_ac_covariates`	[`character()` | `NULL`] Column names with alternative-constant covariates.
`column_as_covariates`	[`character()` | `NULL`] Column names with alternative-specific covariates.
`delimiter`	[`character(1)`] Delimiter separating alternative identifiers from covariate names when `format = "wide"`.
`cross_section`	[`logical(1)`] Treat choice data as cross-sectional?
`choice_effects`	[`choice_effects` | `NULL`] A `choice_effects` object.
`choice_identifiers`	[`choice_identifiers`] A `choice_identifiers` object.
`labels`	[`character()`] Unique labels for the regressors.
`n`	[`integer(1)`] The number of values per regressor.
`marginals`	[`list()`] Optionally marginal distributions for regressors. If not specified, standard normal marginal distributions are used. Each list entry must be named according to a regressor label, and the following distributions are currently supported:

discrete distributions
- Poisson: `list(type = "poisson", lambda = ...)`
- categorical: `list(type = "categorical", p = c(...))`

continuous distributions
- normal: `list(type = "normal", mean = ..., sd = ...)`
- uniform: `list(type = "uniform", min = ..., max = ...)`

`correlation` [`matrix()`]
A correlation matrix of dimension `length(labels)`, where the `(p, q)`-th entry defines the correlation between regressor `labels[p]` and `labels[q]`.

`verbose` [`logical(1)`]
Print information about the simulated regressors?

`x` A `choice_data` or `choice_covariates` object.

Value

`choice_covariates()` and `generate_choice_covariates()` return a `choice_covariates` tibble with the identifier and covariate columns. The column roles are stored in the attributes `format`, `column_decider`, `column_occasion`, `column_alternative`, `column_ac_covariates`, `column_as_covariates`, `delimiter`, and `cross_section`, analogous to `choice_data`.

`covariate_names()` returns a `character` vector.

`design_matrices()` returns a `list` of class `choice_design_matrices` with one numeric design matrix per choice occasion, see the section below. The attributes `Tp` (the number of choice occasions per decider), `alternatives`, `availability` (the indices of the available alternatives per occasion), and `choice_type` describe the structure.

Design matrices

A covariate design matrix contains the choice covariates of a decider at a choice occasion. It is of dimension `J` x `P`, where `J` is the number of choice alternatives and `P` the number of effects.

Examples

```
### sample covariates from choice effects
choice_effects <- choice_effects(
  choice_formula = choice_formula(
    formula = choice ~ price | income | comfort,
    error_term = "probit",
    random_effects = c(
      "price" = "cn",
      "income" = "cn"
    )
  ),
  choice_alternatives = choice_alternatives(J = 3)
)
(choice_covariates <- generate_choice_covariates(
  choice_effects = choice_effects,
  choice_identifiers = generate_choice_identifiers(N = 3, Tp = 2)
))
```

choice_data *Define choice data*

Description

The choice_data object defines the choice data, it is a combination of choice_responses and choice_covariates.

Usage

```
choice_data(
  data_frame,
  format = "wide",
  column_choice = "choice",
  column_decider = "deciderID",
  column_occasion = NULL,
  column_alternative = NULL,
  column_ac_covariates = NULL,
  column_as_covariates = NULL,
  delimiter = "_",
  cross_section = is.null(column_occasion),
  choice_type = c("unordered", "ordered", "ranked")
)

generate_choice_data(
  choice_effects,
  choice_identifiers = generate_choice_identifiers(N = 100),
 choice_covariates = generate_choice_covariates(choice_effects, choice_identifiers),
  choice_parameters = generate_choice_parameters(choice_effects),
 choice_preferences = generate_choice_preferences(choice_effects, choice_parameters,
    choice_identifiers),
  column_choice = "choice",
  choice_type = c("unordered", "ordered", "ranked")
)

long_to_wide(
  data_frame,
  column_ac_covariates = NULL,
  column_as_covariates = NULL,
  column_choice = "choice",
  column_alternative = "alternative",
  column_decider = "deciderID",
  column_occasion = NULL,
  alternatives = as.character(unique(data_frame[[column_alternative]])),
  delimiter = "_",
  choice_type = c("unordered", "ordered", "ranked")
)
```

```
wide_to_long(
  data_frame,
  column_ac_covariates = NULL,
  column_choice = "choice",
  column_alternative = "alternative",
  alternatives = NULL,
  delimiter = "_",
  choice_type = c("unordered", "ordered", "ranked")
)
```

Arguments

data_frame	[data.frame] Contains the choice data.
format	[character(1)] Format of data_frame. Use "wide" when each row contains all alternatives of an occasion and "long" when each row contains a single alternative.
column_choice	[character(1) | NULL] Column name with the observed choices. In wide layout this column should contain a single value per observation: for unordered data the value is the label of the chosen alternative, for ordered data it is the ordered factor or integer score, and for ranked data it is omitted in favor of one column per alternative (see choice_type). In long layout the same column is evaluated once per alternative: unordered data use a binary indicator, 1 or TRUE for the chosen alternative and 0 or FALSE otherwise; ordered data repeats the ordinal value for every alternative, and ranked data stores consecutive ranks 1:k for the observed top k alternatives and NA for unranked alternatives. An entirely missing response marks an occasion that is omitted from the likelihood. Set to NULL for purely covariate tables.
column_decider	[character(1)] Column name with decider identifiers.
column_occasion	[character(1) | NULL] Column name with occasion identifiers. Set to NULL in cross-sectional data.
column_alternative	[character(1) | NULL] Column name with alternative identifiers when format = "long".
column_ac_covariates	[character() | NULL] Column names with alternative-constant covariates.
column_as_covariates	[character() | NULL] Column names of data_frame with alternative-specific covariates.
delimiter	[character(1)] Delimiter separating alternative identifiers from covariate names when format = "wide".

`cross_section` [logical(1)]
Treat choice data as cross-sectional?

`choice_type` [character(1)]
Requested response type. Use "unordered" (default), "ordered", or "ranked".

`choice_effects` [choice_effects]
A choice_effects object.

`choice_identifiers`
[choice_identifiers]
A choice_identifiers object.

`choice_covariates`
[choice_covariates]
A choice_covariates object.

`choice_parameters`
[choice_parameters]
A choice_parameters object.

`choice_preferences`
[choice_preferences]
A choice_preferences object.

`alternatives` [character(J)]
Unique labels for the choice alternatives.

Value

A choice_data tibble in the supplied format. It contains the identifier columns, the response column(s), and the covariate columns of data_frame. The column roles are stored as attributes, which the functions consuming the object rely on:

`format` Either "wide" or "long".

`column_choice` The name of the response column.

`column_decider`, `column_occasion` The identifier columns.

`column_alternative` The alternative column (format = "long").

`column_ac_covariates`, `column_as_covariates` The names of the alternative-constant and alternative-specific covariates.

`column_as_covariates_wide` The alternative-specific covariate columns in wide layout.

`delimiter`, `cross_section`, `choice_type` The corresponding input arguments.

Examples

```
### simulate data from a multinomial probit model
choice_effects <- choice_effects(
  choice_formula = choice_formula(
    formula = choice ~ A | B,
    error_term = "probit",
    random_effects = c("A" = "cn")
  ),
  choice_alternatives = choice_alternatives(J = 3)
```

```
)
generate_choice_data(choice_effects = choice_effects)


### transform from long to wide format
data("TravelMode", package = "AER")
TravelMode$choice <- TravelMode$choice == "yes"
long_to_wide(
  data_frame = TravelMode,
  column_alternative = "mode",
  column_decider = "individual"
)


### transform from wide to long format
data("Train", package = "mlogit")
wide_to_long(
  data_frame = Train
)

### individual choice sets and a missing response
partial_data <- data.frame(
  deciderID = c(1, 1, 2),
  alternative = c("A", "B", "B"),
  choice = c(1L, 0L, NA),
  cost = c(1.2, 1.5, 0.8)
)
choice_data(
  data_frame = partial_data,
  format = "long",
  column_decider = "deciderID",
  column_alternative = "alternative",
  column_as_covariates = "cost"
)
```

`choice_effects` *Define choice model effects*

Description

This function constructs an object of class `choice_effects`, which defines the effects of a choice model.

Usage

```
choice_effects(
  choice_formula,
  choice_alternatives,
  choice_data = NULL,
  delimiter = "_"
```

```
)

## S3 method for class 'choice_effects'
print(x, ...)
```

Arguments

choice_formula	[choice_formula] A choice_formula object.
choice_alternatives	[choice_alternatives] A choice_alternatives object.
choice_data	[NULL | choice_data] A choice_data object. Required to resolve data-dependent elements in choice_formula (if any).
delimiter	[character(1)] The delimiter between covariate and alternative name.
x	[choice_effects] A choice_effects object.
...	Currently not used.

Value

A choice_effects object, which is a data.frame, where each row is a model effect, and columns are

1. "effect_name", the name for the effect which is composed of covariate and alternative name,
2. "generic_name", the generic effect name "beta_<effect number>",
3. "covariate", the (transformed) covariate name connected to the effect,
4. "alternative", the alternative name connected to the effect (only if the effect is alternative-specific),
5. "as_covariate", indicator whether the covariate is alternative-specific,
6. "as_effect", indicator whether the effect is alternative-specific,
7. "mixing", a factor with levels in the order
   (a) "cn" (correlated normal distribution),
   (b) "cln" (positively signed correlated log-normal distribution),
   (c) "cln-" (negatively signed correlated log-normal distribution),
   (d) "n" (uncorrelated normal distribution),
   (e) "ln" (positively signed uncorrelated log-normal distribution),
   (f) "ln-" (negatively signed uncorrelated log-normal distribution),

   indicating the type of random effect,
8. "latent_class", indicator whether the effect differs between latent classes.

For identification, the choice effects are ordered according to the following rules:

1. Non-random effects come before random effects.
2. Otherwise, the order is determined by occurrence in `formula`.

It contains the arguments `choice_formula`, `choice_alternatives`, and `delimiter` as attributes.

Examples

```
### construct choice effects
choice_effects(
  choice_formula = choice_formula(
    formula = choice ~ price | income | I(comfort == 1),
    error_term = "probit",
    random_effects = c(
      "price" = "cn",
      "income" = "cn"
     )
  ),
  choice_alternatives = choice_alternatives(J = 3)
)
```

`choice_formula` *Define choice model formula*

Description

The `choice_formula` object defines the choice model equation.

Usage

```
choice_formula(
  formula,
  error_term = "probit",
  random_effects = character(),
  latent_class_effects = character()
)

## S3 method for class 'choice_formula'
print(x, ...)
```

Arguments

`formula`	[`formula`] A symbolic description of the choice model, see the details on specifying the model formula.
`error_term`	[`character(1)`] Defines the model's error term. Current options are: • `"probit"` (default): errors are multivariate normally distributed • `"logit"`: errors follow a type-I extreme value distribution

`random_effects` [character()]
Named vector defining random effects, see the details on specifying random effects.

`latent_class_effects`
[character()]
Names of covariates whose effects differ between latent classes, see the details on specifying latent class effects.

`x` [choice_formula]
A choice_formula object.

`...` Currently not used.

Value

An object of class `choice_formula`, which is a `list` of the elements:

`formula` The model formula.

`error_term` The name of the model's error term specification.

`choice` The name of the response variable.

`covariate_types` The (up to) three different types of covariates.

`ASC` Does the model have ASCs?

`random_effects` The names of covariates with random effects.

`latent_class_effects` The names of covariates with latent class effects.

`xlevels` The factor levels of the covariates, added once the formula has been resolved with data in choice_effects.

Specifying the model formula

The structure of `formula` is `choice ~ A | B | C`, i.e., a standard formula object but with three parts on the right-hand side, separated by `|`, where

- `choice` is the name of the discrete response variable,
- `A` are names of **alternative-specific covariates** with **a coefficient that is constant across alternatives**,
- `B` are names of **covariates that are constant across alternatives**,
- and `C` are names of **alternative-specific covariates** with **alternative-specific coefficients**.

The following rules apply:

1. By default, intercepts (referred to as alternative-specific constants, ASCs) are added to the model. They can be removed by adding `+ 0` in the second part, e.g., `choice ~ A | B + 0 | C`. To not include any covariates of the second type but to estimate ASCs, add `1` in the second part, e.g., `choice ~ A | 1 | C`. The expression `choice ~ A | 0 | C` is interpreted as no covariates of the second type and no ASCs.
2. To not include covariates of any type, add `0` in the respective part, e.g., `choice ~ 0 | B | C`.
3. Some parts of the formula can be omitted when there is no ambiguity. For example, `choice ~ A` is equivalent to `choice ~ A | 1 | 0`.

4. Multiple covariates in one part are separated by a + sign, e.g., `choice ~ A1 + A2`.
5. Arithmetic transformations of covariates in all three parts of the right-hand side are possible via the function `I()`, e.g., `choice ~ I(A1^2 + A2 * 2)`. In this case, a random effect can be defined for the transformed covariate, e.g., `random_effects = c("I(A1^2 + A2 * 2)" = "cn")`.
6. Ordered choice models have a single utility per choice occasion. Their covariates must be placed in the first part, e.g., `choice ~ age + income`. The intercept (named `"ASC"`) is included by default and removed by `0` in the second part.

Specifying random effects

Specify random effects as `"<covariate>" = "<distribution>"`. Each covariate must appear explicitly on the right-hand side of `formula`; use `"ASC"` for alternative-specific constants.

Available distributions are:

- `"cn"`: correlated normal
- `"n"`: uncorrelated normal
- `"cln"`: positively signed correlated log-normal
- `"ln"`: positively signed uncorrelated log-normal
- `"cln-"`: negatively signed correlated log-normal
- `"ln-"`: negatively signed uncorrelated log-normal

Specifying latent class effects

The covariates in `latent_class_effects` have effects that differ between the latent classes of a mixture model; use `"ASC"` for alternative-specific constants. A random effect named here has a class-specific mean and covariance, any other effect a class-specific coefficient. Effects that are not named are the same in every class, and random effects with and without latent class effects are uncorrelated. A model with more than one latent class needs at least one latent class effect.

Examples

```
### specify a choice formula
choice_formula(
  formula = choice ~ I(A^2 + 1) | B | I(log(C)),
  error_term = "probit",
  random_effects = c("I(A^2+1)" = "cn", "B" = "cn"),
  latent_class_effects = "B"
)
```

choice_identifiers *Define choice identifiers*

Description

The choice_identifiers object defines identifiers for the deciders and choice occasions.

- generate_choice_identifiers() generates identifiers.
- extract_choice_identifiers() extracts choice identifiers.

Usage

```
choice_identifiers(
  data_frame,
  format = "wide",
  column_decider = "deciderID",
  column_occasion = "occasionID",
  cross_section = is.null(column_occasion)
)

generate_choice_identifiers(
  N = length(Tp),
  Tp = 1,
  column_decider = "deciderID",
  column_occasion = "occasionID"
)

extract_choice_identifiers(
  x,
  format = attr(x, "format"),
  column_decider = attr(x, "column_decider"),
  column_occasion = attr(x, "column_occasion"),
  cross_section = attr(x, "cross_section")
)
```

Arguments

data_frame	[data.frame] Contains the choice identifiers.
format	[character(1)] Either "wide" or "long". In the long case, unique combinations of column_decider and column_occasion (if present) are used.
column_decider	[character(1)] The name of the identifier column for deciders.

column_occasion
: [character(1) | NULL]
The name of the identifier column for choice occasions (panel data). Can be NULL for the cross-sectional case.

cross_section
: [logical(1)]
Treat choice data as cross-sectional?

N
: [integer(1)]
The number of deciders.

Tp
: [integer(1) | integer(N)]
The number of choice occasions per decider.

Can also be of length N for a variable number of choice occasions per decider.

x
: A choice_data or choice_covariates object.

Value

An object of class choice_identifiers, which is a tibble with columns:

1. column_decider contains the decider identifiers,
2. column_occasion contains the choice occasion identifiers (only if column_occasion is not NULL and cross_section = FALSE).

The column names are stored in the attributes column_decider and column_occasion, the attribute cross_section records the data type.

Examples

```
### panel case
generate_choice_identifiers(N = 2, Tp = 2)

### cross-sectional case
generate_choice_identifiers(N = 5, column_occasion = NULL)


### read choice identifiers
data("TravelMode", package = "AER")
choice_identifiers(
  data_frame = TravelMode,
  format = "long",
  column_decider = "individual",
  column_occasion = NULL,
  cross_section = TRUE
)
```

choice_likelihood *Define and compute choice likelihood*

Description

These functions prepare and evaluate the likelihood contribution of observed choices for a given choice model.

- `choice_likelihood()` pre-computes the design matrices and choice indices implied by `choice_data` and `choice_effects`. The returned object stores these quantities so that repeated likelihood evaluations during maximum likelihood estimation avoid redundant work.
- `compute_choice_likelihood()` evaluates the (log-)likelihood for given `choice_parameters` and a pre-computed `choice_likelihood` object.

Usage

```
choice_likelihood(
  choice_data,
  choice_effects,
  choice_identifiers = extract_choice_identifiers(choice_data),
  input_checks = TRUE,
  ...
)

compute_choice_likelihood(
  choice_parameters,
  choice_likelihood,
  logarithm = TRUE,
  negative = FALSE,
  aggregate = c("total", "decider", "occasion"),
  ...
)
```

Arguments

`choice_data`	[choice_data] A `choice_data` object.
`choice_effects`	[choice_effects] A `choice_effects` object.
`choice_identifiers`	[choice_identifiers] A `choice_identifiers` object. The default is extracted from `choice_data`.
`input_checks`	[logical(1)] Check the pre-computed design matrices and choice indices against the parameters on the first likelihood evaluation? Later evaluations skip these checks because the pre-computed quantities do not change.

`...` Additional arguments for the probability computation:

- `n_draws` [`integer(1)`]: The number of draws for simulated probabilities, which are required for mixed logit models and for mixed probit models with non-normal random effects. The default is `200`.
- `draws` [`matrix`]: A matrix of standard normal draws with one column per random effect that replaces the generated draws.
- `cml` [`character(1)`]: Composite marginal likelihood for panel probit models. Either `"no"` (default, the full likelihood), `"fp"` (all pairs of choice occasions), or `"ap"` (adjacent pairs of choice occasions).
- `ghk_draws` [`integer(1)`]: The number of draws of the GHK simulator for multivariate normal probabilities in probit models, see `pmvnorm`. Probabilities of up to three dimensions are computed exactly; higher dimensions use the GHK simulator on a fixed sequence of quasi-random Halton points. The default is `500`.

`choice_parameters`
[`choice_parameters | numeric() | list()`]
A `choice_parameters` object.

A numeric vector in optimization space is also accepted and converted with `switch_parameter_space()`.

A list of either representation is evaluated as a batch of parameter draws.

`choice_likelihood`
[`choice_likelihood`]
A `choice_likelihood` object.

`logarithm` [`logical(1)`]
Return the log-likelihood? If `FALSE`, the likelihood is returned.

`negative` [`logical(1)`]
Return the negative (log-)likelihood? Useful for minimization routines.

`aggregate` [`character(1)`]
Unit of the returned likelihood:

- `"occasion"` returns one contribution per observed choice occasion,
- `"decider"` returns one joint contribution per decider,
- `"total"` sums the decider contributions.

Value

`choice_likelihood()` returns an object of class `choice_likelihood`, which is a `list` containing the design matrices, choice indices, and identifiers.

`compute_choice_likelihood()` returns a numeric scalar for `aggregate = "total"` and a named numeric vector otherwise.

Examples

```
### compute choice likelihood
data("Train", package = "mlogit")

choice_effects <- choice_effects(
```

```
  choice_formula = choice_formula(
    formula = choice ~ price | time,
    error_term = "probit"
  ),
  choice_alternatives = choice_alternatives(
    J = 2,
    alternatives = c("A", "B")
  )
)

choice_data <- choice_data(
  data_frame = Train,
  format = "wide",
  column_choice = "choice",
  column_decider = "id",
  column_occasion = "choiceid"
)

likelihood <- choice_likelihood(
  choice_data = choice_data,
  choice_effects = choice_effects
)

choice_parameters <- generate_choice_parameters(
  choice_effects = choice_effects
)

compute_choice_likelihood(
  choice_parameters = choice_parameters,
  choice_likelihood = likelihood,
  logarithm = TRUE
)
```

`choice_parameters` *Define choice model parameters*

Description

These functions construct, validate, and transform an object of class `choice_parameters`, which defines the parameters of a choice model.

- `choice_parameters()` constructs a `choice_parameters` object.
- `generate_choice_parameters()` samples parameters at random, see the details on sampling missing choice model parameters.
- `validate_choice_parameters()` checks model-specific dimensions.
- `switch_parameter_space()` transforms a `choice_parameters` object between the interpretation and optimization space, see the details on the parameter spaces.

Usage

```
choice_parameters(
  beta = NULL,
  Omega = NULL,
  Sigma = NULL,
  gamma = NULL,
  weights = NULL
)

generate_choice_parameters(
  choice_effects,
  fixed_parameters = choice_parameters(),
  C = 1L
)

validate_choice_parameters(
  choice_parameters,
  choice_effects,
  allow_missing = FALSE
)

switch_parameter_space(choice_parameters, choice_effects)
```

Arguments

beta	[numeric(P) | list(C) | NULL] The coefficient vector for computing the linear-in-parameters systematic utility $V = X\beta$. For a latent class model, a list of one coefficient vector per class. Only the effects named in latent_class_effects of the choice_formula may differ between the classes.
Omega	[matrix(nrow = P_r, ncol = P_r) | list(C) | NULL] The covariance matrix of random effects. Not used when P_r = 0. In a latent class model, a list of one covariance matrix per class. Only the block of the random effects with latent class effects may differ between the classes, and it is uncorrelated with the other random effects. Covariances involving uncorrelated random effects are fixed to zero.
Sigma	[matrix(nrow = J, ncol = J) | numeric(1) | NULL] Only relevant in the probit model. For unordered alternatives it is the covariance matrix for the Gaussian error term $\epsilon = U - V$. In ordered models it reduces to a single variance term.
gamma	[numeric(J - 1) | NULL] Vector of strictly increasing threshold parameters required for ordered models. The first element must equal zero for identification.

`weights` [numeric(C) | NULL]
Positive latent class weights.

`choice_effects` [choice_effects]
A choice_effects object.

`fixed_parameters`
[choice_parameters]
A choice_parameters object. Its supplied components are kept fixed. Missing components are completed as described below. A named beta is matched to the effects by name and may contain only some of the effects; the remaining coefficients are drawn.

`C` [integer(1)]
Number of latent classes.

`choice_parameters`
[choice_parameters | numeric()]
A choice_parameters object. For switch_parameter_space() and validate_choice_parameters() a numeric vector in optimization space is also accepted and converted back to a choice_parameters object.

`allow_missing` [logical(1)]
Allow required parameter components to be omitted?

Value

choice_parameters(), generate_choice_parameters(), and validate_choice_parameters() return a choice_parameters list with the elements:

`beta` The coefficient vector (if any).

`Omega` The random-effect covariance matrix on the underlying normal scale (if any).

`Sigma` The error term covariance matrix (or variance in ordered models).

`gamma` Threshold parameters for ordered models (if any).

`weights` The latent class weights (if any).

switch_parameter_space() returns a named numeric vector when given a choice_parameters object and a choice_parameters object when given a numeric optimization vector.

Sampling missing choice model parameters

generate_choice_parameters() completes required components that are absent from fixed_parameters.

Missing components are generated as follows:

`beta` Drawn from a multivariate normal distribution with zero mean and identity covariance matrix, independently for each class for the effects with latent classes and once for the others. A named beta with fewer than P entries fixes the named effects and draws the others.

`Omega` Drawn from an Inverse-Wishart distribution with identity scale matrix and degrees of freedom equal to the dimension plus two, independently for each class for the block of the random effects with latent classes and once for the block of the others. Covariances involving uncorrelated random effects are then set to zero.

`Sigma` For unordered probit models, the lower right block is drawn from an Inverse-Wishart distribution with `J + 1` degrees of freedom and identity scale matrix. The first row and column are fixed to zero and the matrix is scaled so that element $(2, 2)$ equals one. For ordered probit models, `Sigma` is set to one; logit models do not use `Sigma`.

`gamma` For ordered models with two categories, set to zero. Otherwise, positive increments are drawn as `exp(z)`, where the elements of `z` are independent standard normal draws, and cumulatively added to the first threshold zero. Unordered models do not use `gamma`.

`weights` Set to equal class probabilities `1 / C`.

Parameter spaces

The `switch_parameter_space()` function transforms a `choice_parameters` object between the interpretation and optimization space.

- The interpretation space is a `list` of (not necessarily identified) parameters that can be interpreted.
- The optimization space is a `numeric` vector of identified parameters that can be optimized:
  - `beta` is not transformed
  - `Omega` is represented by its vectorized unique Cholesky factor; elements involving uncorrelated random effects are omitted
  - for unordered probit models, `Sigma` is represented through utility differences relative to the first alternative, with the first variance fixed to one, and transformed to a vectorized unique Cholesky factor
  - for ordered probit models, the positive scalar `Sigma` is log-transformed
  - the first ordered threshold is fixed to zero and omitted; logarithms of the remaining positive threshold increments are used
  - in a latent class model, the coefficients and Cholesky elements of the effects with latent classes are concatenated in class order and followed by those of the other effects, which appear once; `C - 1` log weight ratios use the first class as reference

Examples

```
### generate choice parameters at random
J <- 3
choice_effects <- choice_effects(
  choice_formula = choice_formula(
    formula = choice ~ x | y, error_term = "probit",
    random_effects = c("x" = "cn")
  ),
  choice_alternatives = choice_alternatives(J = J)
)
(parameters <- generate_choice_parameters(
  choice_effects = choice_effects,
  fixed_parameters = choice_parameters(
    Sigma = diag(c(0, rep(1, J - 1))) # scale and level normalization
  )
))

### switch between interpretation and optimization spaces
```

```
(optimization_parameters <- switch_parameter_space(
  choice_parameters = parameters,
  choice_effects = choice_effects
))
switch_parameter_space(
  choice_parameters = optimization_parameters,
  choice_effects = choice_effects
)
```

`choice_preferences` *Define choice preferences*

Description

The `choice_preferences` object defines the deciders' preferences in the choice model.

- `choice_preferences()` constructs a `choice_preferences` object.
- `generate_choice_preferences()` samples choice preferences at random.

Usage

```
choice_preferences(data_frame, column_decider = "deciderID")

generate_choice_preferences(
  choice_effects,
  choice_parameters = generate_choice_parameters(choice_effects),
  choice_identifiers = generate_choice_identifiers(N = 100)
)
```

Arguments

`data_frame`	[`data.frame`] Contains the deciders' preferences.
`column_decider`	[`character(1)` | `NULL`] The column name of `data_frame` with the decider identifiers. If `NULL`, decider identifiers are generated.
`choice_effects`	[`choice_effects`] A `choice_effects` object.
`choice_parameters`	[`choice_parameters`] A `choice_parameters` object.
`choice_identifiers`	[`choice_identifiers`] A `choice_identifiers` object.

Value

An object of class choice_preferences, which is a tibble with the deciders' preferences. The column names are the names of the effects in the choice model. The first column contains the decider identifiers, its name is stored in the attribute column_decider.

Examples

```
### generate choice preferences from choice parameters and effects
set.seed(1)
choice_effects <- choice_effects(
  choice_formula = choice_formula(
    formula = choice ~ price | income | comfort,
    error_term = "probit",
    random_effects = c(
      "price" = "cn",
      "income" = "cn"
    ),
    latent_class_effects = "price"
  ),
  choice_alternatives = choice_alternatives(J = 3)
)
choice_parameters <- generate_choice_parameters(
  choice_effects = choice_effects, C = 2
)
(choice_preferences <- generate_choice_preferences(
  choice_parameters = choice_parameters,
  choice_effects = choice_effects,
  choice_identifiers = generate_choice_identifiers(N = 4)
))
```

---

choice_probabilities *Define choice probabilities*

Description

The choice_probabilities object defines the choice probabilities.

- compute_choice_probabilities() calculates the choice probabilities based on the choice parameters and the choice data.

Usage

```
choice_probabilities(
  data_frame,
  choice_only = TRUE,
  column_decider = "deciderID",
  column_occasion = NULL,
  cross_section = is.null(column_occasion),
  column_probabilities = NULL,
```

```
  logarithm = FALSE,
  aggregate = c("occasion", "decider")
)

compute_choice_probabilities(
  choice_parameters,
  choice_data,
  choice_effects,
  choice_only = TRUE,
  input_checks = TRUE,
  aggregate = c("occasion", "decider"),
  logarithm = FALSE,
  ...
)
```

Arguments

data_frame	[data.frame] Contains the choice probabilities.
choice_only	[logical(1)] Only the probabilities for the chosen alternatives?
column_decider	[character(1)] The name of the identifier column for deciders.
column_occasion	[character(1) | NULL] The name of the identifier column for choice occasions (panel data). Can be NULL for the cross-sectional case.
cross_section	[logical(1)] Treat choice data as cross-sectional?
column_probabilities	[character()] The column name of the data_frame with the choice probabilities for all choice alternatives. If choice_only = TRUE, it is the name of a single column that contains the probabilities for the chosen alternatives.
logarithm	[logical(1)] Are the supplied or requested values log-probabilities?
aggregate	[character(1)] Probability unit. "occasion" returns one result per choice occasion. "decider" returns the joint result for each decider's observed sequence of choices.
choice_parameters	[choice_parameters | numeric() | list()] A choice_parameters object. A numeric vector in optimization space, as created by switch_parameter_space, is also accepted.
choice_data	[choice_data] A choice_data object.

choice_effects [choice_effects]
A choice_effects object.

input_checks [logical(1)]
Should additional internal input checks be performed before computing the probabilities?

... Additional arguments for the probability computation:

- n_draws [integer(1)]: The number of draws for simulated probabilities, which are required for mixed logit models and for mixed probit models with non-normal random effects. The default is 200.
- draws [matrix]: A matrix of standard normal draws with one column per random effect that replaces the generated draws.
- cml [character(1)]: Composite marginal likelihood for panel probit models. Either "no" (default, the full likelihood), "fp" (all pairs of choice occasions), or "ap" (adjacent pairs of choice occasions).
- ghk_draws [integer(1)]: The number of draws of the GHK simulator for multivariate normal probabilities in probit models, see pmvnorm. Probabilities of up to three dimensions are computed exactly; higher dimensions use the GHK simulator on a fixed sequence of quasi-random Halton points. The default is 500.

Value

A choice_probabilities tibble with the identifier columns followed by the probability column(s). If choice_only = TRUE, there is a single column choice_probability (or log_choice_probability if logarithm = TRUE). Otherwise, there is one column per choice alternative. The attributes column_decider, column_occasion, cross_section, column_probabilities, choice_only, logarithm, and aggregate store the column roles and the probability type.

If choice_only = FALSE and aggregate = "decider", the tibble instead has one row per possible outcome sequence and decider, with the sequence in the list column outcome.

Examples

```
### multinomial logit
set.seed(1)
effects <- choice_effects(
  choice_formula = choice_formula(
    formula = choice ~ x + z | 0,
    error_term = "logit"
  ),
  choice_alternatives = choice_alternatives(J = 2)
)
parameters <- choice_parameters(beta = c(0.2, -0.1))
simulated_data <- generate_choice_data(
  choice_effects = effects,
  choice_identifiers = generate_choice_identifiers(N = 1L),
  choice_parameters = parameters
)
compute_choice_probabilities(
  choice_parameters = parameters,
```

```
  choice_data = simulated_data,
  choice_effects = effects
)

### multinomial probit
set.seed(1)
effects <- choice_effects(
  choice_formula = choice_formula(
    formula = choice ~ x + z | 0,
    error_term = "probit"
  ),
  choice_alternatives = choice_alternatives(J = 2)
)
parameters <- choice_parameters(
  beta = c(0.2, -0.1),
  Sigma = diag(c(0, 1))
)
simulated_data <- generate_choice_data(
  choice_effects = effects,
  choice_identifiers = generate_choice_identifiers(N = 1L),
  choice_parameters = parameters
)
compute_choice_probabilities(
  choice_parameters = parameters,
  choice_data = simulated_data,
  choice_effects = effects
)

### ordered logit
set.seed(1)
effects <- choice_effects(
  choice_formula = choice_formula(
    formula = choice ~ x + z | 0,
    error_term = "logit"
  ),
  choice_alternatives = choice_alternatives(
    J = 3,
    ordered = TRUE
  )
)
parameters <- choice_parameters(
  beta = c(0.2, -0.1),
  gamma = c(0, 1)
)
simulated_data <- generate_choice_data(
  choice_effects = effects,
  choice_identifiers = generate_choice_identifiers(N = 1L),
  choice_parameters = parameters,
  choice_type = "ordered"
)
compute_choice_probabilities(
  choice_parameters = parameters,
  choice_data = simulated_data,
```

```
  choice_effects = effects
)

### ordered probit
set.seed(1)
effects <- choice_effects(
  choice_formula = choice_formula(
    formula = choice ~ x + z | 0,
    error_term = "probit"
  ),
  choice_alternatives = choice_alternatives(
    J = 3,
    ordered = TRUE
  )
)
parameters <- choice_parameters(
  beta = c(0.2, -0.1),
  Sigma = 1,
  gamma = c(0, 1)
)
simulated_data <- generate_choice_data(
  choice_effects = effects,
  choice_identifiers = generate_choice_identifiers(N = 1L),
  choice_parameters = parameters,
  choice_type = "ordered"
)
compute_choice_probabilities(
  choice_parameters = parameters,
  choice_data = simulated_data,
  choice_effects = effects
)

### ranked logit
set.seed(1)
effects <- choice_effects(
  choice_formula = choice_formula(
    formula = choice ~ x + z | 0,
    error_term = "logit"
  ),
  choice_alternatives = choice_alternatives(J = 2)
)
parameters <- choice_parameters(beta = c(0.2, -0.1))
simulated_data <- generate_choice_data(
  choice_effects = effects,
  choice_identifiers = generate_choice_identifiers(N = 1L),
  choice_parameters = parameters,
  choice_type = "ranked"
)
compute_choice_probabilities(
  choice_parameters = parameters,
  choice_data = simulated_data,
  choice_effects = effects
)
```

```
### ranked probit
set.seed(1)
effects <- choice_effects(
  choice_formula = choice_formula(
    formula = choice ~ x + z | 0,
    error_term = "probit"
  ),
  choice_alternatives = choice_alternatives(J = 2)
)
parameters <- choice_parameters(
  beta = c(0.2, -0.1),
  Sigma = diag(c(0, 1))
)
simulated_data <- generate_choice_data(
  choice_effects = effects,
  choice_identifiers = generate_choice_identifiers(N = 1L),
  choice_parameters = parameters,
  choice_type = "ranked"
)
compute_choice_probabilities(
  choice_parameters = parameters,
  choice_data = simulated_data,
  choice_effects = effects
)

### mixed logit
set.seed(1)
effects <- choice_effects(
  choice_formula = choice_formula(
    formula = choice ~ x + z | 0,
    error_term = "logit",
    random_effects = c(
      x = "cn",
      z = "cn"
    )
  ),
  choice_alternatives = choice_alternatives(J = 2)
)
parameters <- choice_parameters(
  beta = c(0.2, -0.1),
  Omega = matrix(c(0.1, 0.02, 0.02, 0.1), nrow = 2)
)
simulated_data <- generate_choice_data(
  choice_effects = effects,
  choice_identifiers = generate_choice_identifiers(N = 1L),
  choice_parameters = parameters
)
compute_choice_probabilities(
  choice_parameters = parameters,
  choice_data = simulated_data,
  choice_effects = effects
)
```

```
### mixed probit
set.seed(1)
effects <- choice_effects(
  choice_formula = choice_formula(
    formula = choice ~ x + z | 0,
    error_term = "probit",
    random_effects = c(
      x = "cn",
      z = "cn"
    )
  ),
  choice_alternatives = choice_alternatives(J = 2)
)
parameters <- choice_parameters(
  beta = c(0.2, -0.1),
  Omega = matrix(c(0.1, 0.02, 0.02, 0.1), nrow = 2),
  Sigma = diag(c(0, 1))
)
simulated_data <- generate_choice_data(
  choice_effects = effects,
  choice_identifiers = generate_choice_identifiers(N = 1L),
  choice_parameters = parameters
)
compute_choice_probabilities(
  choice_parameters = parameters,
  choice_data = simulated_data,
  choice_effects = effects
)

### panel logit
set.seed(1)
effects <- choice_effects(
  choice_formula = choice_formula(
    formula = choice ~ x + z | 0,
    error_term = "logit",
    random_effects = c(
      x = "cn",
      z = "cn"
    )
  ),
  choice_alternatives = choice_alternatives(J = 2)
)
parameters <- choice_parameters(
  beta = c(0.2, -0.1),
  Omega = matrix(c(0.1, 0.02, 0.02, 0.1), nrow = 2)
)
simulated_data <- generate_choice_data(
  choice_effects = effects,
  choice_identifiers = generate_choice_identifiers(
    N = 1L,
    Tp = 2L
  ),
```

```
  choice_parameters = parameters
)
compute_choice_probabilities(
  choice_parameters = parameters,
  choice_data = simulated_data,
  choice_effects = effects
)

### panel probit
set.seed(1)
effects <- choice_effects(
  choice_formula = choice_formula(
    formula = choice ~ x + z | 0,
    error_term = "probit",
    random_effects = c(
      x = "cn",
      z = "cn"
    )
  ),
  choice_alternatives = choice_alternatives(J = 2)
)
parameters <- choice_parameters(
  beta = c(0.2, -0.1),
  Omega = matrix(c(0.1, 0.02, 0.02, 0.1), nrow = 2),
  Sigma = diag(c(0, 1))
)
simulated_data <- generate_choice_data(
  choice_effects = effects,
  choice_identifiers = generate_choice_identifiers(
    N = 1L,
    Tp = 2L
  ),
  choice_parameters = parameters
)
compute_choice_probabilities(
  choice_parameters = parameters,
  choice_data = simulated_data,
  choice_effects = effects,
  aggregate = "decider",
  cml = "ap"
)

### latent class logit
set.seed(1)
effects <- choice_effects(
  choice_formula = choice_formula(
    formula = choice ~ x + z | 0,
    error_term = "logit",
    latent_class_effects = c("x", "z")
  ),
  choice_alternatives = choice_alternatives(J = 2)
)
parameters <- choice_parameters(
```

```
  beta = list(c(0.2, -0.1), c(-0.2, 0.1)),
  weights = c(0.5, 0.5)
)
simulated_data <- generate_choice_data(
  choice_effects = effects,
  choice_identifiers = generate_choice_identifiers(N = 1L),
  choice_parameters = parameters
)
compute_choice_probabilities(
  choice_parameters = parameters,
  choice_data = simulated_data,
  choice_effects = effects
)

### latent class probit
set.seed(1)
effects <- choice_effects(
  choice_formula = choice_formula(
    formula = choice ~ x + z | 0,
    error_term = "probit",
    latent_class_effects = c("x", "z")
  ),
  choice_alternatives = choice_alternatives(J = 2)
)
parameters <- choice_parameters(
  beta = list(c(0.2, -0.1), c(-0.2, 0.1)),
  Sigma = diag(c(0, 1)),
  weights = c(0.5, 0.5)
)
simulated_data <- generate_choice_data(
  choice_effects = effects,
  choice_identifiers = generate_choice_identifiers(N = 1L),
  choice_parameters = parameters
)
compute_choice_probabilities(
  choice_parameters = parameters,
  choice_data = simulated_data,
  choice_effects = effects
)
```

`choice_responses` *Define choice response*

Description

The `choice_responses` object defines the observed choice responses.

- `generate_choice_responses()` simulates choices

Usage

```
choice_responses(
  data_frame,
  column_choice = "choice",
  column_decider = "deciderID",
  column_occasion = NULL,
  cross_section = is.null(column_occasion)
)

generate_choice_responses(
  choice_effects,
 choice_covariates = generate_choice_covariates(choice_effects = choice_effects),
 choice_parameters = generate_choice_parameters(choice_effects = choice_effects),
  choice_identifiers = extract_choice_identifiers(choice_covariates),
 choice_preferences = generate_choice_preferences(choice_parameters = choice_parameters,
    choice_effects = choice_effects, choice_identifiers = choice_identifiers),
  column_choice = "choice",
  choice_type = c("unordered", "ordered", "ranked")
)
```

Arguments

`data_frame`	[`data.frame`] Contains the choice responses.
`column_choice`	[`character(1)`] The column name of `data_frame` with the choice responses.
`column_decider`	[`character(1)`] The name of the identifier column for deciders.
`column_occasion`	[`character(1)` | `NULL`] The name of the identifier column for choice occasions (panel data). Can be `NULL` for the cross-sectional case.
`cross_section`	[`logical(1)`] Treat choice data as cross-sectional?
`choice_effects`	[`choice_effects`] A `choice_effects` object.
`choice_covariates`	[`choice_covariates`] A `choice_covariates` object.
`choice_parameters`	[`choice_parameters`] A `choice_parameters` object.
`choice_identifiers`	[`choice_identifiers`] A `choice_identifiers` object.

choice_preferences
: [choice_preferences]
A choice_preferences object.

choice_type
: [character(1)]
The response type to simulate. Use "unordered" (default), "ordered", or "ranked".

Value

A choice_responses tibble with the identifier columns followed by the response column(s). The attributes column_choice, column_decider, column_occasion, cross_section, and column_response_columns (all non-identifier columns, e.g., the ranking columns for ranked responses) store the column roles.

Examples

```
### generate choice responses from choice effects
choice_effects <- choice_effects(
  choice_formula = choice_formula(
    formula = choice ~ price | time,
    error_term = "probit"
  ),
  choice_alternatives = choice_alternatives(J = 5)
)
(generate_choice_responses(
  choice_effects = choice_effects,
  choice_type = "ranked"
))
```

---

compute_P *Count model effects*

Description

compute_P() returns the total number of effects in a choice model.

Usage

```
compute_P(choice_effects)
```

Arguments

choice_effects
: [choice_effects]
A choice_effects object.

Value

An integer with the number of model effects.

`lichess_berserk_choice` *Berserk Choices in a Lichess Arena Tournament*

Description

Revealed binary Berserk choices made by 5,852 players in the Lichess Yearly Rapid Arena held on April 16, 2026. The chess tournament used a 10-minute clock and lasted eight hours. Going Berserk at the beginning of a game reduced a player's clock to five minutes. In return, a win earned one extra tournament point. A player is considered to have been on a streak when they entered the game after at least two consecutive wins. While on a streak, a win was worth four instead of two points and a draw two instead of one. A Berserk win on a streak was therefore worth five points. A draw or loss ends the streak.

This data can be used to model the risky choice to go Berserk based on playing color, player rating, rating difference to the opponent, remaining tournament time, and risk to lose their winning streak.

Usage

```
lichess_berserk_choice
```

Format

A `tibble` with 37,416 rows and 8 columns:

deciderID [`character`] The player's Lichess username.

occasionID [`integer`] The player's games in chronological order, numbered from 1.

berserk [`logical`] Whether the player chose to go Berserk.

white [`logical`] Whether the player had the white pieces.

rating [`integer`] The player's Rapid rating at the beginning of the game.

ratingDifference [`integer`] The player's rating minus the opponent's rating.

minutesRemaining [`numeric`] The scheduled tournament time remaining, in minutes, when the game was created.

streak [`logical`] Whether the player was on a double-point win streak at the beginning of the game.

Source

Derived from the public game export, results, and tournament metadata for the Lichess Yearly Rapid Arena using the Lichess Arena API. The streak and Berserk rules are described in the Arena tournament FAQ. Lichess database exports are released under the CC0 license.

`train_test` *Split choice data into a train and a test subset*

Description

This function splits choice data by deciders or by choice occasions, for example to fit a model on the train subset and to evaluate its predictions on the test subset.

Usage

```
train_test(
  data_frame,
  test_proportion = NULL,
  test_number = NULL,
  by = "decider",
  random = FALSE,
  column_decider = "deciderID",
  column_occasion = NULL
)
```

Arguments

`data_frame`	[`data.frame`] Contains the choice data.
`test_proportion`	[`numeric(1)`	`NULL`] The proportion of deciders or occasions in the test subset.
`test_number`	[`integer(1)`	`NULL`] The number of deciders, or of occasions per decider, in the test subset.
`by`	[`character(1)`] Split by `"decider"` or by `"occasion"`.
`random`	[`logical(1)`] Draw the test subset at random? Else, it is the last deciders or occasions.
`column_decider`	[`character(1)`] Column name with decider identifiers.
`column_occasion`	[`character(1)`	`NULL`] Column name with occasion identifiers. Set to `NULL` in cross-sectional data.

Details

Exactly one of `test_proportion` and `test_number` sets the size of the test subset. Splitting by occasions keeps every decider in both subsets and applies the size per decider, which requires panel data.

Value

A `list` of two subsets of `data_frame`, named `train` and `test`.

Examples

```
data("Train", package = "mlogit")

### 20% of the deciders in the test subset
parts <- train_test(Train, test_proportion = 0.2, column_decider = "id")
lengths(lapply(parts, function(part) unique(part$id)))

### the last choice occasion of every decider in the test subset
parts <- train_test(
  Train, test_number = 1, by = "occasion",
  column_decider = "id", column_occasion = "choiceid"
)
nrow(parts$test)
```

---

water_conservation_choice

Dairy-Farm Water Conservation Choices

Description

Stated panel choices by 98 dairy farmers in Mejia, Ecuador. Each farmer completed four choice tasks with two conservation plans and a status quo. The data have 1176 rows in long format.

This data can be used to model the choice between two water-conservation plans and the status quo based on irrigation, manure and waste management, training, cost-share payment, farm size, milk production, and cattle density.

Usage

```
water_conservation_choice
```

Format

A `tibble` with 1176 rows and 12 columns:

farmer [`integer`] The farmer identifier.

occasion [`integer`] The choice occasion from 1 to 4.

alternative [`character`] Plan 1, plan 2, or the status quo.

choice [`integer`] Whether the alternative was chosen.

irrigation [`factor`] No new system, micro-sprinklers, or solid rain.

manure [`factor`] No new practice, composting, or dispersion.

waste [`factor`] No new practice, a collection center, or a municipal container.

training [`logical`] Whether training for water conflict resolution and cooperation is offered.

payment [`numeric`] The cost-share payment in US dollars per hectare.

farm_size [`numeric`] The farm size in hectares.

milk_production [`numeric`] The daily milk production in liters.

cattle_density [`numeric`] The number of cattle per hectare.

Source

Adapted from the Mendeley Data source doi:10.17632/ncj6ws6hbj.1. The source data are licensed under CC BY 4.0.

References

Ortiz CA, Avila-Santamaría JJ, Martinez-Cruz AL (2023). "Dairy Farmers' Willingness to Adopt Cleaner Production Practices for Water Conservation: A Discrete Choice Experiment in Mejia, Ecuador." *Agricultural Water Management*, **278**, 108168. doi:10.1016/j.agwat.2023.108168.

`wind_power_choice` *Choices for a Proposed Wind-Power Project*

Description

Stated panel choices by 308 residents for a proposed wind-power project near Setskog, Norway. Each respondent completed six choice tasks. The alternatives are the status quo and two project plans. Compensation values were converted to Euro at an exchange rate of 10.8770 Norwegian kroner = 1 Euro.

This data can be used to model the choice between two wind-power project plans and the status quo based on the number and height of turbines, power-line placement, compensation, and collective psychological ownership.

Usage

```
wind_power_choice
```

Format

A `tibble` with 1848 rows and 16 columns:

respondent [`integer`] The respondent identifier.

occasion [`integer`] The choice occasion from 1 to 6.

choice [`character`] The chosen alternative from 1 to 3.

turbines_1–turbines_3 [`integer`] The number of turbines for each alternative.

height_1–height_3 [`integer`] The turbine height in meters for each alternative.

powerline_1–powerline_3 [`factor`] The power-line route and placement: none, overhead or underground throughout, or mixed between forests and residential areas.

compensation_1–compensation_3 [`numeric`] The annual reduction in municipal taxes offered as compensation, in euros.

psychological_ownership [`numeric`] A respondent-specific, model-estimated latent score for *collective psychological ownership* of the natural area affected by the proposed wind farm. It was constructed from three seven-point Likert items asking whether the area is "ours" and belongs collectively to residents. The construct was normalized to mean zero and standard deviation one in the study: positive values indicate stronger and negative values weaker feelings of shared ownership relative to the sample average.

Source

Adapted from the Mendeley Data source doi:10.17632/3pdx4p3s9g.1. The source data are licensed under CC BY 4.0.

References

Dugstad A, Grimsrud K, Kipperberg G, Lindhjem H, Navrud S (2023). "Place Attachment and Preferences for Wind Energy – A Value-Based Approach." *Energy Research & Social Science*, **100**, 103094. doi:10.1016/j.erss.2023.103094.

Dugstad A, Brouwer R, Grimsrud K, Kipperberg G, Lindhjem H, Navrud S (2024). "Nature Is Ours! – Psychological Ownership and Preferences for Wind Energy." *Energy Economics*, **129**, 107239. doi:10.1016/j.eneco.2023.107239.

Index

∗ **dataset**
  lichess_berserk_choice, 35
  water_conservation_choice, 37
  wind_power_choice, 38
∗ **data**
  choice_covariates, 4
  choice_data, 7
  choice_identifiers, 15
  choice_responses, 32
  train_test, 36
∗ **model**
  choice_alternatives, 2
  choice_effects, 10
  choice_formula, 12
  choice_parameters, 19
  choice_preferences, 23
  compute_P, 34
∗ **probability**
  choice_likelihood, 17
  choice_probabilities, 24

choice_alternatives, 2, *3*, *11*
choice_covariates, 4, *6*, *9*, *16*, *33*
choice_data, *6*, 7, *11*, *16, 17*, *25*
choice_effects, *5*, *9*, 10, *11*, *13*, *17*, *21*, *23*, *26*, *33, 34*
choice_formula, *11*, 12, *13*, *20*
choice_identifiers, *5*, *9*, 15, *17*, *23*, *33*
choice_likelihood, 17, *18*
choice_parameters, *9*, *18*, 19, *21*, *23*, *25*, *33*
choice_preferences, *9*, 23, *34*
choice_probabilities, 24
choice_responses, 32
compute_choice_likelihood *(*choice_likelihood*)*, 17
compute_choice_probabilities *(*choice_probabilities*)*, 24
compute_P, 34
covariate_names *(*choice_covariates*)*, 4

design_matrices *(*choice_covariates*)*, 4

extract_choice_identifiers *(*choice_identifiers*)*, 15

formula, *13*

generate_choice_covariates *(*choice_covariates*)*, 4
generate_choice_data *(*choice_data*)*, 7
generate_choice_identifiers *(*choice_identifiers*)*, 15
generate_choice_parameters *(*choice_parameters*)*, 19
generate_choice_preferences *(*choice_preferences*)*, 23
generate_choice_responses *(*choice_responses*)*, 32

lichess_berserk_choice, 35
long_to_wide *(*choice_data*)*, 7

pmvnorm, *18*, *26*
print.choice_alternatives *(*choice_alternatives*)*, 2
print.choice_effects *(*choice_effects*)*, 10
print.choice_formula *(*choice_formula*)*, 12

switch_parameter_space, *25*
switch_parameter_space *(*choice_parameters*)*, 19

train_test, 36

validate_choice_parameters *(*choice_parameters*)*, 19

water_conservation_choice, 37
wide_to_long *(*choice_data*)*, 7
wind_power_choice, 38