Package 'FactorialSurvey'

November 20, 2026

Type Package

Title Design and Analysis of Factorial Survey (Vignette) Experiments

Version 0.3.0

Description An integrated workflow for factorial survey (vignette) experiments. Constructs D-efficient sets of vignettes from a factorial structure and reports their D- and A-efficiency following Kuhfeld, Tobias and Garratt (1994) <doi:10.1177/002224379403100408>, allocates them to blocks and to respondents in carryover-balanced orders, renders them as prose, and writes them out for survey platforms. Simulates responses to evaluate statistical power before fielding. Estimates beta regression models for bounded ratings and logit models for yes/no responses, with standard errors clustered by respondent, and derives average marginal effects, marginal willingness to pay and the expected willingness to pay for complete policy scenarios. The latter is the mean of willingness to pay truncated at zero, Hanemann (1989) <doi:10.2307/1242685>, applied to factorial surveys as in Tanaka et al. (2027) <doi:10.1016/j.tra.2026.105293>.
Efficient designs and optimal blocking use the exchange algorithms of Fedorov (1972, ISBN:0122507509) as implemented in 'AlgDesign'. Presentation orders follow Williams (1949) <doi:10.1071/CH9490149>. Beta regression follows Ferrari and Cribari-Neto (2004) <doi:10.1080/0266476042000214501>. Bounded ratings are mapped to the open unit interval following Smithson and Verkuilen (2006) <doi:10.1037/1082-989X.11.1.54>.

License MIT + file LICENSE

Encoding UTF-8

Language en-GB

LazyData true

Depends R (>= 4.1)

Imports stats, tools, utils, AlgDesign, betareg, ggplot2 (>= 3.4.0), sandwich

Suggests openxlsx, margins, testthat (>= 3.0.0), withr, knitr, rmarkdown

VignetteBuilder knitr

Config/testthat/edition 3

RoxygenNote 7.3.1

URL https://github.com/tanakak/FactorialSurvey

BugReports https://github.com/tanakak/FactorialSurvey/issues

NeedsCompilation no

Author Katsuya Tanaka [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0003-0455-0133>)

Maintainer Katsuya Tanaka <katsuyat@gmail.com>

Repository CRAN

Date/Publication 2026-11-20 12:10:02 UTC

Contents

as.data.frame.fs_ewtp . . . 3
as.data.frame.fs_mwtp . . . 4
bento . . . 4
fs_assign . . . 5
fs_betareg . . . 7
fs_betareg-extractors . . . 10
fs_defficient . . . 11
fs_design_eval . . . 14
fs_efficiency . . . 15
fs_ewtp . . . 17
fs_export . . . 19
fs_logit . . . 21
fs_logit-extractors . . . 23
fs_mwtp . . . 24
fs_plot . . . 26
fs_power . . . 27
fs_simulate_ratings . . . 30
fs_vignette_text . . . 32
fs_williams . . . 33
plot.fs_power . . . 34
print.fs_assignment . . . 35
print.fs_betareg . . . 36
print.fs_design . . . 36
print.fs_design_eval . . . 37
print.fs_efficiency . . . 38
print.fs_ewtp . . . 38
print.fs_logit . . . 39
print.fs_mwtp . . . 40
print.fs_power . . . 40
print.summary.fs_betareg . . . 41

print.summary.fs_logit 42
set_labels 42
summary.fs_betareg 43
summary.fs_logit 44
theme_fs 45

Index **46**

`as.data.frame.fs_ewtp` *Coerce expected willingness to pay estimates to a data frame*

Description

Coerce expected willingness to pay estimates to a data frame

Usage

```
## S3 method for class 'fs_ewtp'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	An object of class `fs_ewtp`, from `fs_ewtp()`.
`row.names`	Passed to `as.data.frame()`; normally left as `NULL`.
`optional`	Passed to `as.data.frame()`; normally left as `FALSE`.
`...`	Ignored.

Value

A data frame with columns `scenario`, `index`, `ewtp`, `std_error`, `conf_low` and `conf_high`.

Examples

```
set.seed(1)
d <- data.frame(id = rep(1:40, each = 4), q = rbinom(160, 1, 0.5),
                cost = sample(c(100, 300, 500), 160, replace = TRUE))
d$y <- rbinom(160, 1, plogis(0.5 + d$q - 0.004 * d$cost))
fit <- fs_logit(y ~ q + cost, data = d, cluster = ~ id, compute_me = FALSE)
as.data.frame(fs_ewtp(fit, data.frame(q = 0:1), price = "cost"))
```

as.data.frame.fs_mwtp *Coerce marginal willingness to pay estimates to a data frame*

Description

Coerce marginal willingness to pay estimates to a data frame

Usage

```
## S3 method for class 'fs_mwtp'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	An object of class fs_mwtp, from fs_mwtp().
row.names	Passed to as.data.frame(); normally left as NULL.
optional	Passed to as.data.frame(); normally left as FALSE.
...	Ignored.

Value

A data frame with columns variable, mwtp, std_error, conf_low and conf_high.

Examples

```
set.seed(1)
d <- data.frame(id = rep(1:40, each = 3), x = rnorm(120),
                price = runif(120, 100, 900))
d$rating <- 10 * plogis(0.7 * d$x - 0.002 * d$price + rnorm(120, sd = 0.4))
fit <- fs_betareg(rating ~ x + price, data = d, cluster = ~ id,
                  compute_me = FALSE)
as.data.frame(fs_mwtp(fit, price = "price"))
```

bento *Ratings of take-away lunch boxes from a classroom factorial survey*

Description

Ratings of eight take-away lunch box (bento) vignettes by 48 university students. Each vignette describes a lunch box by its main dish, whether it comes with a side dish, whether nutrition information is shown, and its price; respondents rated how likely they would be to buy it on a 0 to 10 scale. Every respondent rated all eight vignettes.

Usage

```
bento
```

Format

A data frame with 384 rows (48 respondents by 8 vignettes) and 7 variables:

respondent Respondent identifier, 1 to 48.

vignette Vignette identifier, 1 to 8.

main Main dish, a factor with levels `"grilled fish"` (the reference), `"teriyaki chicken"` and `"tofu hamburger"`.

side Side dish included: 1 yes, 0 no.

info Nutrition information shown: 1 yes, 0 no.

price Price in Japanese yen: 700, 900, 1100 or 1300.

rating Purchase intention, from 0 ("would definitely not buy") to 10 ("would definitely buy").

Details

The data are small and were collected for teaching, which makes them convenient for illustrating the workflow: estimating a beta regression with standard errors clustered by respondent, then marginal and expected willingness to pay.

Source

Collected from students in the course "Environmental Data Analysis" at Shiga University in 2025, with their permission to publish the anonymous responses.

Examples

```
head(bento)
fit <- fs_betareg(rating ~ main + side + info + price, data = bento,
                  cluster = ~ respondent, compute_me = FALSE)
fs_mwtp(fit, price = "price")
```

`fs_assign` *Allocate blocks and presentation orders to respondents*

Description

Turns a design into the schedule a survey platform actually needs: one row per respondent per vignette, stating which vignette that respondent sees in which position. Blocks are dealt out evenly across respondents, and the order within a block can be balanced with a Williams design so that position and carryover effects do not confound the attributes.

Usage

```
fs_assign(
  design,
  n_respondents,
  order = c("williams", "random", "as_is"),
  seed = NULL
)
```

Arguments

design	An object of class fs_design from fs_defficient(), or a data frame with a block column.
n_respondents	The number of respondents to schedule.
order	How the vignettes within a block are ordered for each respondent. One of "williams" (the default, a carryover-balanced set of orders from fs_williams()), "random" (an independent random order per respondent) or "as_is" (the order the rows appear in the design).
seed	A single number making the allocation reproducible, or NULL. The caller's random number state is restored on exit.

Details

Respondents are assigned to blocks by dealing round: respondent 1 gets block 1, respondent 2 block 2, and so on, wrapping around. When n_respondents is not a multiple of the number of blocks, the earlier blocks receive one extra respondent and a message says so.

With order = "williams" the respondents within a block cycle through the rows of fs_williams(). Carryover balance is only fully achieved once the number of respondents per block is a multiple of the number of orders, which is the block size for an even block size and twice it for an odd one; a message reports the number of orders so this can be checked when choosing the sample size.

Value

An object of class fs_assignment, which inherits from data.frame, with one row per respondent per vignette and the columns:

respondent Respondent identifier, 1 to n_respondents.

block The block that respondent was given.

position Position in the questionnaire, 1 upwards.

vignette Row number of the vignette in design, so that responses can be matched back to the design.

... Every column of design, including any text column added by fs_vignette_text().

The number of distinct orders used is attached as the n_orders attribute, and the ordering method as order.

See Also

fs_williams(), fs_defficient(), fs_export()

Examples

```
d <- fs_defficient(list(damage = c("minor", "severe"),
                        cost   = c(100, 500, 1000)),
                   n_vignettes = 6, n_blocks = 2,
                   seed = 1, verbose = FALSE)
```

```
schedule <- fs_assign(d, n_respondents = 8, seed = 1)
head(schedule)

# Every respondent sees every vignette in their block exactly once
table(schedule$respondent)

# Across respondents, each vignette appears in each position equally often
with(subset(schedule, block == 1), table(vignette, position))
```

`fs_betareg` *Beta regression for bounded rating outcomes*

Description

Fits a beta regression to the ratings collected in a factorial survey. Beta regression is well suited to responses that are bounded and often skewed, such as a 0 to 10 acceptability score, because it models the mean on a bounded support instead of assuming an unbounded normal error.

The observed scale is mapped onto the open unit interval, the model is fitted with `betareg::betareg()`, and – optionally – average marginal effects are reported on both the unit interval and the original scale.

Usage

```
fs_betareg(
  formula,
  data,
  scale = c(0, 10),
  transform = TRUE,
  cluster = NULL,
  compute_me = TRUE,
  verbose = FALSE,
  ...
)
```

Arguments

`formula`	A symbolic description of the model, passed to `betareg::betareg()`. Extended formulas of the form `y ~ x | z`, which model the precision parameter as well as the mean, are supported.
`data`	A data frame containing the variables in `formula`.
`scale`	A numeric vector of length two giving the lower and upper bound of the observed rating scale. The default `c(0, 10)` matches a 0 to 10 scale. Ignored when `transform` is `FALSE`.
`transform`	A logical value. If `TRUE` (the default), the response is rescaled from `scale` onto the open interval (0, 1). If `FALSE`, the response is assumed to already lie strictly inside (0, 1).

`cluster`	How to account for the fact that each respondent rates several vignettes. Give the respondent identifier as a one-sided formula such as `~ id`, as the name of a column of `data`, or as a vector with one element per row of `data`. Standard errors, marginal effects and everything computed from them – including `fs_mwtp()` – are then cluster-robust. Use `FALSE` to treat every rating as independent. The default `NULL` does the same but issues a warning, because in a factorial survey independence is almost never true and ordinary standard errors are then not valid.
`compute_me`	A logical value. If `TRUE` (the default), average marginal effects are computed with the **margins** package when it is installed.
`verbose`	A logical value. If `TRUE`, progress is reported through `message()`. The default is `FALSE`; use `summary()` on the result to see the fitted model.
`...`	Further arguments passed to `betareg::betareg()`, such as `link` or `subset`.

Details

The response is mapped onto (0, 1) in two steps. It is first shifted and scaled to $[0, 1]$ using `scale`, then pulled away from the endpoints with the transformation of Smithson and Verkuilen (2006), $(y(N-1)+0.5)/N$, where N is the number of observations. Average marginal effects on the unit interval are converted back to the original scale by multiplying by $(u-l)N/(N-1)$, where l and u are the bounds in `scale`.

Two pseudo R-squared values are reported because they measure different things and are easy to confuse. `pseudo_r2_cox_snell` is the likelihood-ratio measure of Cox and Snell, $1-\exp(2(\ell_0-\ell)/n)$, computed against an intercept-only model that this function fits for the purpose. `pseudo_r2_betareg` is the value **betareg** itself reports, namely the squared correlation between the linear predictor and the link-transformed response.

With `cluster`, the covariance matrix is the cluster-robust sandwich estimator of `sandwich::vcovCL()` with the usual $G/(G-1)$ small-sample adjustment, where G is the number of respondents. It allows the ratings of one respondent to be correlated in any way, which is what a respondent-specific taste or response style produces. With few respondents (say, fewer than 30) these standard errors are themselves imprecise.

McFadden's $1-\ell/\ell_0$ is deliberately **not** reported. It is designed for discrete outcomes, whose log-likelihoods are always negative. A beta regression is a continuous density, so its log-likelihood is frequently positive, and McFadden's ratio then takes values outside $[0, 1]$ and cannot be interpreted. The Cox and Snell measure is well behaved regardless of the sign of the log-likelihood.

Value

An object of class `fs_betareg`, a list with components:

`model` The fitted `betareg::betareg()` object.

`vcov` The covariance matrix of the estimates used for all inference: cluster-robust when `cluster` is given, otherwise the model-based matrix.

`cluster` A list describing the clustering (`name`, `n_clusters`), or `NULL`.

`marginal_effects` The `margins` object, or `NULL`.

`marginal_effects_rescaled` A data frame of average marginal effects expressed on the original rating scale, or `NULL`.

`gof` A named list of goodness-of-fit measures: `logLik`, `logLik_null`, `AIC`, `BIC`, `lr_stat`, `lr_df`, `lr_p_value`, `pseudo_r2_cox_snell`, `pseudo_r2_betareg` and `nobs`.

`scale` The rating scale supplied.

`transform` Whether the response was rescaled.

`rescale_factor` The factor mapping effects on (0, 1) back to the original scale, or `NA` when `transform` is `FALSE`.

`call` The matched call.

Clustered data

When each respondent rates several vignettes the ratings of one person are correlated, and standard errors that assume independence are not valid. The error is largest for characteristics of the respondent, such as age or gender, which are constant within a respondent: there, ordinary standard errors are much too small (in the package's validation simulations, nominal 95 percent intervals covered the truth only about 80 percent of the time). For attributes varied within respondents they were close to right or somewhat too large. Cluster-robust standard errors are valid in both cases, so use `cluster`.

References

Ferrari, S. L. P. and Cribari-Neto, F. (2004). Beta regression for modelling rates and proportions. *Journal of Applied Statistics*, 31(7), 799–815. doi:10.1080/0266476042000214501

Smithson, M. and Verkuilen, J. (2006). A better lemon squeezer? Maximum-likelihood regression with beta-distributed dependent variables. *Psychological Methods*, 11(1), 54–71. doi:10.1037/1082989X.11.1.54

See Also

`fs_mwtp()` for turning the fitted coefficients into a willingness to pay, `fs_defficient()` for building the design.

Examples

```
set.seed(123)
# 50 respondents, each rating 3 vignettes
n <- 150
d <- data.frame(
  id      = rep(1:50, each = 3),
  quality = rnorm(n),
  cost    = runif(n, 100, 1000)
)
taste <- rnorm(50, sd = 0.4)[d$id]   # respondent-specific shift
d$rating <- 10 * plogis(0.6 * d$quality - 0.002 * (d$cost - 500) +
                          taste + rnorm(n, sd = 0.5))

fit <- fs_betareg(rating ~ quality + cost, data = d, cluster = ~ id,
                  compute_me = FALSE)
fit
summary(fit)
```

`fs_betareg-extractors` *Extract components from a factorial survey beta regression*

Description

Methods that forward to the underlying `betareg::betareg()` fit, so that `fs_betareg` objects can be used with the usual extractor functions. `vcov()` returns the covariance matrix used for inference, which is cluster-robust when the model was fitted with `cluster`.

Usage

```
## S3 method for class 'fs_betareg'
coef(object, ...)

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

## S3 method for class 'fs_betareg'
logLik(object, ...)

## S3 method for class 'fs_betareg'
nobs(object, ...)

## S3 method for class 'fs_betareg'
fitted(object, ...)
```

Arguments

`object`	An object of class `fs_betareg`.
`...`	Passed on to the corresponding method for `betareg` objects.

Value

The value returned by the corresponding `betareg` method: a named numeric vector for `coef()`, a matrix for `vcov()`, an object of class `logLik` for `logLik()`, a single integer for `nobs()`, and a numeric vector for `fitted()`.

Examples

```
set.seed(1)
d <- data.frame(id = rep(1:20, each = 3), x = rnorm(60))
d$rating <- 10 * plogis(0.5 * d$x + rnorm(60, sd = 0.4))
fit <- fs_betareg(rating ~ x, data = d, cluster = ~ id,
                  compute_me = FALSE)
coef(fit)
nobs(fit)
```

`fs_defficient` *Generate a D-efficient set of vignettes for a factorial survey*

Description

Selects a subset of the full factorial that is efficient for estimating the effects of the attributes, using the exchange algorithm of `AlgDesign::optFederov()`, and splits the selected vignettes into blocks that can be administered to different respondents.

Usage

```
fs_defficient(
  factors,
  n_vignettes,
  n_blocks = 1,
  model = ~.,
  linear = NULL,
  restrictions = NULL,
  blocking = c("optimal", "random"),
  n_repeats = 50,
  shuffle_within_blocks = TRUE,
  seed = NULL,
  max_candidates = 1e+05,
  verbose = TRUE
)
```

Arguments

`factors`	A named list defining the attributes and their levels, in the form accepted by `expand.grid()`. Each element may be numeric, character or a factor. Every attribute is treated as categorical during the search, so that each of its levels appears in the design, even when the levels are numbers such as `1:3` or `c(400, 600, 800)`; see `linear` for the exception.
`n_vignettes`	The number of vignettes to select. Must not exceed the number of distinct combinations, and must be at least the number of parameters implied by `model`.
`n_blocks`	The number of blocks the vignettes are divided into. `n_vignettes` must be divisible by `n_blocks`.
`model`	A one-sided formula giving the model the design should be efficient for. The default `~ .` covers main effects only; use `~ .^2` to make the design efficient for two-way interactions as well.
`linear`	An optional character vector naming numeric attributes for which only a linear effect is to be estimated. The design is then made efficient for a slope rather than for separate level effects, which pushes the selection towards the extreme levels and may leave intermediate levels out altogether. Leave it `NULL` (the default) unless that is what you want: a price attribute that will enter the analysis as a continuous variable is still best given all its levels in the design.

`restrictions` An optional one-sided formula, or a list of such formulas, describing combinations that must **not** appear. Each formula is evaluated within the full factorial and rows for which it is `TRUE` are removed before the search, for example `~ tenure == "new" & condition == "derelict"`.

`blocking` How the selected vignettes are divided into blocks. Either `"optimal"` (the default), which uses `AlgDesign::optBlock()` to make each block as informative as possible on its own, or `"random"`, which shuffles the vignettes and deals them out in equal numbers. Ignored when `n_blocks` is 1.

`n_repeats` The number of random starts used by the exchange algorithm. Larger values search harder at the cost of run time.

`shuffle_within_blocks`
A logical value. If `TRUE` (the default), the row order within each block is randomised, which helps to break up order effects in the questionnaire. For a systematic treatment of order, see `fs_assign()`.

`seed` A single number used to make the search and the shuffling reproducible, or `NULL` for no seed. The state of the caller's random number generator is restored when the function returns.

`max_candidates` The largest candidate set the function will build before giving up, as a guard against an accidentally enormous factorial.

`verbose` A logical value. If `TRUE` (the default), progress is reported through `message()`.

Details

The search is carried out with every attribute coded as a factor, so the design is efficient for estimating a separate effect for each level. This matters when levels are supplied as numbers: treated as a continuous variable, a three-level attribute coded `1:3` would be optimised for a slope, and the middle level would rarely or never be shown. Attributes named in `linear` are the deliberate exception. In the returned design the columns keep the type of the levels supplied: numbers stay numbers.

The reported D- and A-efficiencies follow Kuhfeld, Tobias and Garratt (1994) and equal 100 for an orthogonal array. See `fs_efficiency()`.

With `blocking = "optimal"` the selected vignettes are allocated to blocks by `AlgDesign::optBlock()`, which searches for an allocation that keeps each block as informative as possible in its own right. This matters when a respondent sees only one block and you want each block to support within-respondent comparisons. With `blocking = "random"` the vignettes are shuffled and dealt out; block sizes are still exactly equal, but nothing guarantees that any individual block is well balanced.

Optimal blocking is the better default, but it is a search: with an awkward number of blocks it can fail to converge, in which case the function falls back to random allocation and says so.

The function does not write files. Use `fs_export()` to write a design to disk in a form a survey platform will accept.

Value

An object of class `fs_design`, which inherits from `data.frame`. It has one row per selected vignette, the attribute columns, and a `block` column. The following attributes are attached:

`factors` The list of attributes and levels supplied.

`d_efficiency` The D-efficiency in percent, relative to a hypothetical orthogonal design of the same size; see `fs_efficiency()`.

`a_efficiency` The A-efficiency in percent.

`linear` The attributes treated as linear.

`n_candidates` The number of candidate combinations searched.

`model` The model formula the design was optimised for.

`blocking` The blocking method used.

`block_criterion` The raw blocked criterion from `AlgDesign::optBlock()`, or `NA` for random or absent blocking. It is an internal search criterion, not an efficiency.

References

Fedorov, V. V. (1972). *Theory of Optimal Experiments*. Academic Press.

Kuhfeld, W. F., Tobias, R. D. and Garratt, M. (1994). Efficient experimental design with marketing research applications. *Journal of Marketing Research*, 31(4), 545–557. doi:10.1177/002224379403100408

See Also

`fs_efficiency()` for the efficiency of any design, `fs_design_eval()` for level balance and correlation diagnostics, `set_labels()` for attaching the wording shown to respondents, `fs_vignette_text()` for turning rows into prose, `fs_assign()` for allocating blocks and presentation orders to respondents.

Examples

```
attrs <- list(
  damage   = c("minor", "moderate", "severe"),
  distance = c("near", "far"),
  cost     = c(100, 500, 1000)
)

d <- fs_defficient(attrs, n_vignettes = 12, n_blocks = 2,
                   seed = 42, verbose = FALSE)
head(as.data.frame(d))
table(d$block)

# Exclude a combination that would not make sense to a respondent
d2 <- fs_defficient(attrs, n_vignettes = 9,
                    restrictions = ~ damage == "minor" & cost == 1000,
                    seed = 42, verbose = FALSE)
any(d2$damage == "minor" & d2$cost == 1000)
```

`fs_design_eval` *Diagnostics for a factorial survey design*

Description

Reports how evenly the levels of each attribute appear in a design and how strongly the attributes are correlated with one another. A well-behaved design has near-equal level counts and near-zero correlations; large correlations mean the effects of the attributes involved will be hard to separate.

Usage

```
fs_design_eval(
  design,
  by_block = TRUE,
  factors = NULL,
  model = NULL,
  linear = NULL
)
```

Arguments

`design`	An object of class `fs_design`, from `fs_defficient()`, or a plain data frame of attribute columns.
`by_block`	A logical value. If `TRUE`, level counts are reported separately for each block as well as overall.
`factors`, `model`, `linear`	Passed to `fs_efficiency()`. For an `fs_design` object they are taken from the object when left `NULL`; for a plain data frame, supplying `factors` lets levels that never appear be detected.

Value

An object of class `fs_design_eval`, a list with components:

`balance` A list of tables of level counts, one per attribute.

`balance_by_block` A list of tables of level counts by block, or `NULL` when `by_block` is `FALSE` or there is a single block.

`correlation` The correlation matrix of the attributes, computed on their numeric codes.

`max_abs_correlation` The largest absolute off-diagonal correlation.

`efficiency` The result of `fs_efficiency()`: D- and A-efficiency in percent, estimability and missing levels.

`n_vignettes` The number of vignettes.

See Also

`fs_defficient()`

Examples

```
d <- fs_defficient(list(damage   = c("low", "high"),
                        distance = c("near", "far"),
                        cost     = c(100, 500, 1000)),
                   n_vignettes = 12, n_blocks = 2,
                   seed = 7, verbose = FALSE)
fs_design_eval(d)
```

`fs_efficiency` *D- and A-efficiency of a factorial survey design*

Description

Computes the D-efficiency and A-efficiency of a set of vignettes, expressed as a percentage of a hypothetical orthogonal and balanced design with the same number of rows. A value of 100 means the design is as informative as an orthogonal array; an orthogonal array, where one exists, attains 100.

Use it to check a design produced by `fs_defficient()` or any design built elsewhere, for example one used in an earlier survey.

Usage

```
fs_efficiency(design, factors = NULL, model = ~., linear = NULL)
```

Arguments

`design`	A data frame with one row per vignette and one column per attribute. A `block` column, if present, is ignored. An `fs_design` object from `fs_defficient()` is accepted as well.
`factors`	An optional named list giving the full set of levels of each attribute, in the form accepted by `fs_defficient()`. Supplying it is strongly recommended: if a level never appears in `design`, it can only be detected when the intended levels are known. When `NULL`, the levels are taken from `attr(design, "factors")` if present, and otherwise inferred from the values that occur in `design`.
`model`	A one-sided formula giving the model the design is meant to estimate. The default `~ .` covers main effects; use `~ .^2` to include two-way interactions.
`linear`	An optional character vector naming numeric attributes that enter the model as a single linear term rather than as a set of level effects. Defaults to `attr(design, "linear")` when present.

Details

Each attribute is coded so that, over a full factorial, its columns have mean zero, unit mean square and are mutually orthogonal: orthogonal polynomial contrasts scaled by $\sqrt{k}$ for an attribute with k

levels, and the standardised value for an attribute listed in `linear`. With this coding an orthogonal and balanced design has information matrix $X'X/N = I$, and the efficiencies are

$$D = 100\,|X'X/N|^{1/p}, \qquad A = 100\,p/\mathrm{tr}\{(X'X/N)^{-1}\},$$

following Kuhfeld, Tobias and Garratt (1994). Both are zero when the design cannot estimate the model.

The D-efficiency is invariant to the choice of coding only up to this normalisation, which is why the value reported by some optimisation routines – for instance the criterion `D` returned by `AlgDesign::optFederov()`, computed on treatment-coded dummies – is not comparable across designs and is not an efficiency in this sense.

When restrictions exclude some combinations, 100 is usually out of reach; the value then measures the cost of the restrictions as well as the quality of the search.

Value

A list of class `fs_efficiency` with components:

`d_efficiency` D-efficiency in percent.

`a_efficiency` A-efficiency in percent.

`n_vignettes` The number of rows, N.

`n_parameters` The number of model parameters, p, including the intercept.

`estimable` `TRUE` when every parameter can be estimated, that is, when the information matrix is non-singular.

`missing_levels` A named list of levels that never appear in `design`, with one element per attribute that has any.

References

Kuhfeld, W. F., Tobias, R. D. and Garratt, M. (1994). Efficient experimental design with marketing research applications. *Journal of Marketing Research*, 31(4), 545–557. doi:10.1177/002224379403100408

See Also

`fs_defficient()`, `fs_design_eval()`

Examples

```
# A 3^4 orthogonal array (L9) is 100 percent efficient
l9 <- data.frame(a = c(1, 1, 1, 2, 2, 2, 3, 3, 3),
                 b = c(1, 2, 3, 1, 2, 3, 1, 2, 3),
                 c = c(1, 2, 3, 2, 3, 1, 3, 1, 2),
                 d = c(1, 2, 3, 3, 1, 2, 2, 3, 1))
fs_efficiency(l9)

# A design that never shows the middle level cannot estimate its effect
bad <- l9
bad$a[bad$a == 2] <- 3
fs_efficiency(bad, factors = list(a = 1:3, b = 1:3, c = 1:3, d = 1:3))
```

`fs_ewtp` *Expected willingness to pay for a policy scenario*

Description

Computes the expected willingness to pay (EWTP) for one or more policy scenarios – complete bundles of attribute levels – from a model with a logit link and a cost attribute. Where `fs_mwtp()` values a change in a single attribute, `fs_ewtp()` values a whole scenario.

Usage

```
fs_ewtp(
  object,
  newdata,
  price,
  method = c("delta", "krinsky-robb"),
  average = FALSE,
  level = 0.95,
  n_sim = 5000,
  seed = NULL
)
```

Arguments

`object`	A fitted model with a logit link: an `fs_logit` object from `fs_logit()`, an `fs_betareg` object from `fs_betareg()`, or a plain binomial-logit `stats::glm()` or logit-link `betareg::betareg()` fit. The covariance matrix returned by `vcov(object)` is used, so a model fitted with `cluster` gives cluster-robust intervals.
`newdata`	A data frame with one row per scenario, giving the level of every attribute (and any respondent covariate) in the model. The price column, if present, is ignored: EWTP is the payment the scenario is worth, not the response at a given price. Row names, when set, are used to label the scenarios.
`price`	The name of the price (cost) variable. It must enter the model as a single numeric term with a negative coefficient.
`method`	How the confidence interval is obtained: `"delta"` (the default) or `"krinsky-robb"` (simulation from the estimated sampling distribution of the coefficients).
`average`	A logical value. If `TRUE`, the EWTP is averaged over the rows of `newdata` and a single estimate is returned. Use it with one row per respondent to obtain a sample mean of individual EWTP.
`level`	The confidence level of the interval.
`n_sim`	The number of draws used by `method = "krinsky-robb"`.
`seed`	A single number making the Krinsky-Robb draws reproducible, or `NULL`. The caller's random number state is restored on exit.

Details

Write the model's linear predictor for a scenario x offered at cost R as $x'\beta - \gamma R$, where $\gamma > 0$ is minus the price coefficient. With a logit link the probability of support (for `fs_logit()`) or the expected rating on the unit interval (for `fs_betareg()`) is $\Lambda(x'\beta - \gamma R)$. Treating this, as in the utility-difference model of Hanemann (1984), as the probability that willingness to pay exceeds R, and willingness to pay as non-negative, the expected willingness to pay is

$$E(\mathrm{WTP}) = \int_0^\infty \Lambda(x'\beta - \gamma R)\, dR = \frac{1}{\gamma} \ln\{1 + \exp(x'\beta)\},$$

the mean of the distribution truncated at zero (Hanemann 1989). Tanaka (2025) and Tanaka et al. (2027, Eq. 6) apply it to complete policy scenarios in factorial surveys, including ratings analysed by beta regression. It is expressed in the units of the price variable.

The intercept and every attribute enter $x'\beta$, so EWTP depends on the full scenario, not only on differences between scenarios, and it is not additive: the EWTP of a bundle is not the sum of marginal WTPs. For a beta regression the interpretation rests on reading the mean rating, after mapping the scale to the unit interval, as a propensity to support the scenario.

The delta-method standard error uses the gradient $\partial E/\partial\beta = \Lambda(x'\beta)\, x/\gamma$ and $\partial E/\partial\beta_{price} = E/\gamma$. Because EWTP is a non-linear function with γ in the denominator, the Krinsky-Robb interval is preferable when the price coefficient is imprecise.

Value

An object of class `fs_ewtp`, a list with components:

`estimates` A data frame with one row per scenario (or a single row when `average = TRUE`) and columns `scenario`, `index` (the utility index $x'\beta$ at zero cost), `ewtp`, `std_error`, `conf_low` and `conf_high`.

`price` The name of the price variable.

`method` The interval method used.

`level` The confidence level.

`draws` The simulated EWTP values, for `method = "krinsky-robb"`; otherwise `NULL`.

`n_invalid` The number of Krinsky-Robb draws discarded because the simulated price coefficient was not negative.

References

Hanemann, W. M. (1984). Welfare evaluations in contingent valuation experiments with discrete responses. *American Journal of Agricultural Economics*, 66(3), 332–341. doi:10.2307/1240800

Hanemann, W. M. (1989). Welfare evaluations in contingent valuation experiments with discrete response data: Reply. *American Journal of Agricultural Economics*, 71(4), 1057–1061. doi:10.2307/1242685

Tanaka, K. (2025). A unified framework for estimating willingness to pay using factorial survey experiments. SSRN Working Paper. doi:10.2139/ssrn.5238343

Tanaka, K., Dill, J., McNeil, N., Geller, R. and Kondo, N. (2027). What are safer streets worth? Bridging resident preferences and policy design in traffic calming. *Transportation Research Part A*, 215, 105293. doi:10.1016/j.tra.2026.105293

See Also

`fs_mwtp()` for the willingness to pay for a single attribute.

Examples

```
set.seed(42)
n_resp <- 80
d <- data.frame(
  id      = rep(seq_len(n_resp), each = 4),
  quality = sample(0:1, 4 * n_resp, replace = TRUE),
  speed   = sample(0:1, 4 * n_resp, replace = TRUE),
  cost    = sample(c(500, 1000, 2000), 4 * n_resp, replace = TRUE)
)
d$support <- rbinom(nrow(d), 1, plogis(0.5 + 0.8 * d$quality +
                                       0.4 * d$speed - 0.001 * d$cost))
fit <- fs_logit(support ~ quality + speed + cost, data = d,
                cluster = ~ id, compute_me = FALSE)

scenarios <- data.frame(quality = c(0, 1, 1), speed = c(0, 0, 1),
                        row.names = c("status quo", "quality", "both"))
fs_ewtp(fit, scenarios, price = "cost")
```

---

fs_export *Write a design or schedule out for a survey platform*

Description

Saves a design from `fs_defficient()` or a respondent schedule from `fs_assign()` in a form a survey tool or a fieldwork agency will accept. Text is always written as UTF-8, optionally with a byte order mark so that Excel on Windows opens non-ASCII files correctly instead of showing mojibake.

Usage

```
fs_export(
  x,
  file,
  format = c("csv", "tsv", "qualtrics", "xlsx"),
  by_block = FALSE,
  columns = NULL,
  bom = TRUE,
  overwrite = FALSE
)
```

Arguments

x	An `fs_design`, an `fs_assignment`, or any data frame.

file	The path to write to. When the output is split by block, the block number is inserted before the file extension, so design.csv becomes design_block1.csv, design_block2.csv and so on.
format	The output format. One of "csv", "tsv", "qualtrics" or "xlsx". See Details.
by_block	A logical value. If TRUE, one file is written per block. Ignored by "xlsx", which always writes one sheet per block, and forced to TRUE by "qualtrics" when there is more than one block. Defaults to FALSE.
columns	An optional character vector naming the columns to write, in the order wanted. The default writes every column except block, which is redundant once the output is split.
bom	A logical value. If TRUE (the default), a UTF-8 byte order mark is written at the start of "csv" and "tsv" files. This is what makes Japanese, accented and other non-ASCII text display correctly when the file is double-clicked into Excel on Windows. Set it to FALSE for tools that mistake the mark for data.
overwrite	A logical value. If FALSE (the default), the function refuses to replace an existing file.

Details

"csv" and "tsv" write the object as it stands, one row per vignette or per respondent-position.

"qualtrics" writes a loop table: one row per loop iteration, a leading Loop column numbering the iterations, and one column per field. This is the shape the Loop & Merge editor expects, so the file can be opened in a spreadsheet and pasted in, with field 1 of the loop referring to the first attribute column. Qualtrics does not publish a documented file-import specification for Loop & Merge, so this is a table shaped for pasting rather than a format guaranteed to import unchanged; check the field order once before fielding. When the design has several blocks, one file per block is written, since a loop belongs to a single block.

"xlsx" writes a single workbook with one sheet per block, named block1, block2 and so on. This is the format fieldwork agencies most often ask for. It requires the **openxlsx** package.

Value

A character vector of the paths written, invisibly.

See Also

fs_defficient(), fs_assign(), fs_vignette_text()

Examples

```
d <- fs_defficient(list(damage = c("minor", "severe"),
                        cost   = c(100, 500, 1000)),
                   n_vignettes = 6, n_blocks = 2,
                   seed = 1, verbose = FALSE)
d <- fs_vignette_text(d, "Damage is {{damage}}; the cost is {{cost}}.")

# Write into the session's temporary directory
path <- file.path(tempdir(), "design.csv")
```

```
fs_export(d, path)
read.csv(path, fileEncoding = "UTF-8-BOM")[1:2, ]

# One Qualtrics loop table per block
files <- fs_export(d, file.path(tempdir(), "loop.csv"), format = "qualtrics")
basename(files)

unlink(c(path, files))
```

`fs_logit` *Logit model for binary responses to vignettes*

Description

Fits a binary logit model to yes/no responses collected in a factorial survey – for example, whether a respondent would support the policy described in a vignette at the stated cost. Together with `fs_betareg()` for ratings, it covers the two response formats of the framework of Tanaka (2025); `fs_mwtp()` and `fs_ewtp()` accept either fit.

Usage

```
fs_logit(
  formula,
  data,
  cluster = NULL,
  compute_me = TRUE,
  verbose = FALSE,
  ...
)
```

Arguments

`formula`	A model formula with a binary response on the left: a numeric 0/1 variable, a logical, or a factor with two levels (the second level is the "yes" outcome, as in `stats::glm()`).
`data`	A data frame containing the variables in `formula`.
`cluster`	How to account for repeated responses from the same respondent, exactly as in `fs_betareg()`: a one-sided formula such as `~ id`, a column name, a vector with one element per row, or `FALSE`. The default `NULL` treats responses as independent and warns.
`compute_me`	A logical value. If `TRUE` (the default), average marginal effects on the probability scale are computed with the **margins** package when it is installed.
`verbose`	A logical value. If `TRUE`, progress is reported through `message()`.
`...`	Further arguments passed to `stats::glm()`, such as `weights` or `subset`.

Details

For a binary outcome McFadden's pseudo R-squared, $1 - \ell/\ell_0$, is well defined and is reported, together with the share of responses predicted correctly at a 0.5 threshold. Cluster-robust standard errors are computed as in `fs_betareg()`.

Value

An object of class `fs_logit`, a list with components:

`model` The fitted `stats::glm()` object.

`vcov` The covariance matrix used for inference: cluster-robust when `cluster` is given, otherwise model-based.

`cluster` A list describing the clustering, or `NULL`.

`marginal_effects` The `margins` object, or `NULL`.

`gof` A named list of fit measures: `logLik`, `logLik_null`, `AIC`, `BIC`, `lr_stat`, `lr_df`, `lr_p_value`, `pseudo_r2_mcfadden`, `percent_correct` and `nobs`.

`call` The matched call.

References

Tanaka, K. (2025). A unified framework for estimating willingness to pay using factorial survey experiments. SSRN Working Paper. doi:10.2139/ssrn.5238343

See Also

`fs_ewtp()` and `fs_mwtp()` for willingness to pay, `fs_betareg()` for rating responses.

Examples

```
set.seed(42)
n_resp <- 80
d <- data.frame(
  id      = rep(seq_len(n_resp), each = 4),
  quality = sample(0:1, 4 * n_resp, replace = TRUE),
  cost    = sample(c(500, 1000, 2000), 4 * n_resp, replace = TRUE)
)
taste <- rnorm(n_resp, sd = 0.5)[d$id]
d$support <- rbinom(nrow(d), 1,
                    plogis(1 + 0.8 * d$quality - 0.001 * d$cost + taste))

fit <- fs_logit(support ~ quality + cost, data = d, cluster = ~ id,
                compute_me = FALSE)
summary(fit)
fs_mwtp(fit, price = "cost")
```

`fs_logit-extractors` *Extract components from a factorial survey logit model*

Description

Methods that forward to the underlying `stats::glm()` fit. `vcov()` returns the covariance matrix used for inference, which is cluster-robust when the model was fitted with `cluster`.

Usage

```
## S3 method for class 'fs_logit'
coef(object, ...)

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

## S3 method for class 'fs_logit'
logLik(object, ...)

## S3 method for class 'fs_logit'
nobs(object, ...)

## S3 method for class 'fs_logit'
fitted(object, ...)
```

Arguments

`object`	An object of class `fs_logit`.
`...`	Passed on to the corresponding method for `glm` objects.

Value

The value returned by the corresponding `glm` method.

Examples

```
set.seed(1)
d <- data.frame(id = rep(1:30, each = 4), x = rnorm(120))
d$y <- rbinom(120, 1, plogis(0.8 * d$x))
fit <- fs_logit(y ~ x, data = d, cluster = ~ id, compute_me = FALSE)
coef(fit)
vcov(fit)
```

`fs_mwtp` *Marginal willingness to pay from a fitted model*

Description

Converts a pair of estimated coefficients into a marginal willingness to pay (MWTP), the amount of the price attribute a respondent would trade for a one-unit improvement in another attribute:

$$MWTP = -\beta_{attribute}/\beta_{price}.$$

Because MWTP is a ratio of two estimates its sampling distribution is skewed, and more so the closer the price coefficient sits to zero. Three interval methods are therefore offered, and it is worth comparing them before reporting a result.

Usage

```
fs_mwtp(
  object,
  variable = NULL,
  price,
  method = c("delta", "krinsky-robb", "fieller"),
  level = 0.95,
  n_sim = 5000,
  seed = NULL
)
```

Arguments

`object`	A fitted model: an `fs_betareg` object from `fs_betareg()` (whose cluster-robust covariance matrix, if any, is used), a `betareg::betareg()` fit, or any model for which `coef()` and `vcov()` return a named vector and a conformable matrix.
`variable`	The name of the attribute coefficient, as it appears in `coef(object)`. If `NULL` (the default), every coefficient other than the intercept, the price coefficient and the precision parameter is used, and a table of MWTP values is returned.
`price`	The name of the price coefficient.
`method`	The interval method. One of `"delta"` (the delta method, the default), `"krinsky-robb"` (simulation from the estimated sampling distribution of the coefficients) or `"fieller"` (Fieller's theorem, which respects the skewness of a ratio).
`level`	The confidence level. Defaults to `0.95`.
`n_sim`	The number of draws used by `method = "krinsky-robb"`.
`seed`	A single number making the Krinsky-Robb draws reproducible, or `NULL`. The caller's random number state is restored on exit.

Details

The delta method takes a first-order Taylor expansion of the ratio and yields a symmetric interval. It is quick and familiar but understates skewness, and can produce implausible intervals when the price coefficient is imprecisely estimated.

Krinsky and Robb (1986) draw repeatedly from a multivariate normal with the estimated mean and covariance of the coefficients, form the ratio in each draw, and read the interval off the empirical quantiles. This captures the skewness and is the usual recommendation in the stated preference literature.

Fieller's theorem inverts the test that the ratio equals a given value. It is exact under normality of the coefficients but the resulting region is unbounded when the price coefficient is not clearly distinguishable from zero; in that case the interval is returned as `NA` with a warning.

The coefficients enter on the scale of the linear predictor. For a beta regression with a logit link the ratio is therefore a ratio of effects on the latent scale, which is the conventional definition of MWTP but is not the same as the ratio of average marginal effects on the response scale.

Value

An object of class `fs_mwtp`, a list with components:

`estimates` A data frame with one row per attribute and columns `variable`, `mwtp`, `std_error`, `conf_low` and `conf_high`.

`price` The name of the price coefficient.

`method` The interval method used.

`level` The confidence level.

`draws` The simulated MWTP values, for `method = "krinsky-robb"`; otherwise `NULL`.

References

Krinsky, I. and Robb, A. L. (1986). On approximating the statistical properties of elasticities. *Review of Economics and Statistics*, 68(4), 715–719. doi:10.2307/1924536

Fieller, E. C. (1954). Some problems in interval estimation. *Journal of the Royal Statistical Society, Series B*, 16(2), 175–185. doi:10.1111/j.25176161.1954.tb00159.x

See Also

`fs_betareg()`

Examples

```
set.seed(123)
n <- 200   # 50 respondents x 4 vignettes
d <- data.frame(
  id      = rep(1:50, each = 4),
  quality = sample(0:1, n, replace = TRUE),
  speed   = rnorm(n),
  price   = runif(n, 100, 1000)
)
```

```
d$rating <- 10 * plogis(0.8 * d$quality + 0.3 * d$speed -
                          0.0015 * d$price + rnorm(n, sd = 0.5))

fit <- fs_betareg(rating ~ quality + speed + price, data = d,
                  cluster = ~ id, compute_me = FALSE)

# All attributes at once
fs_mwtp(fit, price = "price")

# A single attribute, with a simulation-based interval
fs_mwtp(fit, variable = "quality", price = "price",
        method = "krinsky-robb", n_sim = 500, seed = 1)
```

`fs_plot` *Plot estimates with confidence intervals*

Description

Draws a forest plot – one point and interval per row – of willingness to pay or of the effects of the attributes. Each method returns a `ggplot2::ggplot()` object, so titles, labels and themes can be changed with the usual + syntax.

Usage

```
## S3 method for class 'fs_mwtp'
plot(x, ...)

## S3 method for class 'fs_ewtp'
plot(x, ...)

## S3 method for class 'fs_betareg'
plot(x, type = c("auto", "ame", "coef"), level = 0.95, ...)

## S3 method for class 'fs_logit'
plot(x, type = c("auto", "ame", "coef"), level = 0.95, ...)
```

Arguments

`x`	An object of class `fs_mwtp` (from `fs_mwtp()`), `fs_ewtp` (from `fs_ewtp()`), `fs_betareg` (from `fs_betareg()`) or `fs_logit` (from `fs_logit()`).
`...`	Ignored.
`type`	For model fits only: `"ame"` plots average marginal effects – on the rating scale for `fs_betareg`, on the probability scale for `fs_logit` – and requires that they were computed when the model was fitted; `"coef"` plots the coefficients of the mean equation on the logit scale with Wald intervals from `vcov(x)`. The default uses marginal effects when they are available and coefficients otherwise.
`level`	For `type = "coef"`, the confidence level of the intervals.

Details

Rows appear from top to bottom in the order of the estimates. A dashed vertical line marks zero. The plots use a single dark colour and no fill, so they reproduce in black and white.

Value

A ggplot object.

Examples

```
set.seed(1)
n <- 240
d <- data.frame(id      = rep(1:60, each = 4),
                quality = rbinom(n, 1, 0.5),
                speed   = rbinom(n, 1, 0.5),
                price   = sample(c(100, 300, 500), n, replace = TRUE))
d$rating <- round(10 * plogis(0.8 * d$quality + 0.4 * d$speed -
                                0.004 * d$price + rnorm(n, sd = 0.5)))
fit <- fs_betareg(rating ~ quality + speed + price, data = d,
                  cluster = ~ id, compute_me = FALSE)

plot(fs_mwtp(fit, price = "price"))
plot(fit, type = "coef")
plot(fs_ewtp(fit, data.frame(quality = c(0, 1, 1), speed = c(0, 0, 1),
                             row.names = c("none", "quality", "both")),
             price = "price")) +
  ggplot2::labs(x = "Expected WTP (yen)")
```

`fs_power` *Power and precision of a factorial survey by simulation*

Description

Answers the two questions asked when a factorial survey is planned: with this design and this many respondents, how likely is each attribute effect to be detected, and how precisely will willingness to pay be estimated? Data are simulated repeatedly with `fs_simulate_ratings()` and analysed exactly as real data would be – `fs_betareg()` or `fs_logit()` with standard errors clustered by respondent – and the results are summarised.

Usage

```
fs_power(
  design,
  n_respondents,
  coef,
  formula,
  price = NULL,
  response = c("rating", "binary"),
```

```
  phi = 10,
  sd_respondent = 0.5,
  scale = c(0, 10),
  round = TRUE,
  alpha = 0.05,
  n_sim = 200,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

design	Either an fs_design object from fs_defficient() (or any data frame of vignettes with a block column), in which case blocks and presentation orders are allocated with fs_assign(), or an fs_assignment schedule from fs_assign(), which is used as it is.
n_respondents	A vector of sample sizes to evaluate.
coef	A named numeric vector of coefficients on the logit scale. The names must match the columns of the model matrix implied by formula, including "(Intercept)"; the error message lists them if they do not.
formula	A one-sided formula giving the systematic part of the model, such as ~ damage + cost. Character attributes become factors with the levels in the design, so their coefficients are named as in stats::model.matrix(), for example "damagesevere".
price	Optional name of the price variable. When given, the precision of the marginal willingness to pay for each other attribute is also reported.
response	The kind of response to generate: "rating" (the default) for a bounded rating, or "binary" for a yes/no answer.
phi	The precision of the beta distribution for ratings. Larger values give less noise around the mean. Ignored for binary responses.
sd_respondent	The standard deviation of a normally distributed respondent-specific shift on the logit scale. It makes the ratings of one respondent correlated, as they are in real data. Use 0 for independent responses.
scale	A numeric vector of length two giving the bounds of the rating scale.
round	A logical value. If TRUE (the default), ratings are rounded to whole numbers, as on a 0 to 10 scale.
alpha	The significance level of the two-sided tests.
n_sim	The number of simulated surveys per sample size.
seed	A single number making the simulation reproducible, or NULL. The caller's random number state is restored on exit.
verbose	A logical value. If TRUE, progress is reported through message().

Details

Each simulated survey uses the same design and allocation rules as fs_assign(). Effects are tested with cluster-robust Wald tests, so the respondent-level correlation introduced by sd_respondent is taken into account, as it would be in the real analysis.

Because coefficients estimated without a respondent effect are attenuated when `sd_respondent > 0`, power refers to detecting the effect that the analysis actually estimates. The true MWTP is the ratio of the supplied coefficients, which the attenuation leaves almost unchanged.

Choosing `coef` is the hard part. Values from a pilot study or a similar published survey are best. Run time grows with `n_sim` times the number of sample sizes; 200 to 500 simulations are usually enough to compare designs.

Value

An object of class `fs_power`, a data frame with one row per sample size and model term, and the columns:

`n_respondents` The sample size.

`term` The model term (a column of the model matrix).

`coef` The true coefficient on the logit scale.

`power` The share of simulated surveys in which the term was significant at level `alpha`.

`mwtp` The true marginal willingness to pay, when `price` is given.

`mwtp_halfwidth` The average half-width of the delta-method confidence interval for the MWTP, in units of the price variable.

`mwtp_relative` That half-width as a share of the true MWTP: 0.2 means the interval is roughly the estimate plus or minus 20 percent.

The settings, the number of failed fits and `n_sim` are attached as attributes.

See Also

`fs_simulate_ratings()`, `fs_defficient()`

Examples

```
design <- fs_defficient(
  list(damage = c("minor", "severe"), cost = c(100, 300, 500)),
  n_vignettes = 6, n_blocks = 2, seed = 1, verbose = FALSE
)
pw <- fs_power(design, n_respondents = c(50, 150),
               coef = c("(Intercept)" = 1, damagesevere = 0.4,
                        cost = -0.003),
               formula = ~ damage + cost, price = "cost",
               n_sim = 20, seed = 1, verbose = FALSE)
pw

plot(pw)
```

`fs_simulate_ratings` *Simulate responses to a factorial survey*

Description

Generates artificial responses to a design from a model whose coefficients you choose. Because the true values are known, the result is useful for checking that an analysis recovers them, for teaching, and – through `fs_power()` – for deciding how many respondents a survey needs.

Usage

```
fs_simulate_ratings(
  design,
  n_respondents = NULL,
  coef,
  formula,
  response = c("rating", "binary"),
  phi = 10,
  sd_respondent = 0.5,
  scale = c(0, 10),
  round = TRUE,
  name = if (response == "rating") {
     "rating"
 } else {
     "response"
 },
  seed = NULL
)
```

Arguments

`design`	Either an `fs_design` object from `fs_defficient()` (or any data frame of vignettes with a `block` column), in which case blocks and presentation orders are allocated with `fs_assign()`, or an `fs_assignment` schedule from `fs_assign()`, which is used as it is.
`n_respondents`	The number of respondents. Required when `design` is a design; ignored for a schedule.
`coef`	A named numeric vector of coefficients on the logit scale. The names must match the columns of the model matrix implied by `formula`, including `"(Intercept)"`; the error message lists them if they do not.
`formula`	A one-sided formula giving the systematic part of the model, such as `~ damage + cost`. Character attributes become factors with the levels in the design, so their coefficients are named as in `stats::model.matrix()`, for example `"damagesevere"`.
`response`	The kind of response to generate: `"rating"` (the default) for a bounded rating, or `"binary"` for a yes/no answer.

phi	The precision of the beta distribution for ratings. Larger values give less noise around the mean. Ignored for binary responses.
sd_respondent	The standard deviation of a normally distributed respondent-specific shift on the logit scale. It makes the ratings of one respondent correlated, as they are in real data. Use 0 for independent responses.
scale	A numeric vector of length two giving the bounds of the rating scale.
round	A logical value. If TRUE (the default), ratings are rounded to whole numbers, as on a 0 to 10 scale.
name	The name of the response column.
seed	A single number making the simulation reproducible, or NULL. The caller's random number state is restored on exit.

Details

For respondent i and vignette j the linear predictor is $\eta_{ij} = x'_{ij}\beta + u_i$ with $u_i \sim N(0, \sigma_u^2)$, and $\mu_{ij} = \Lambda(\eta_{ij})$. A rating is drawn from a beta distribution with mean μ_{ij} and precision ϕ and mapped onto scale; a binary response is drawn from a Bernoulli distribution with probability μ_{ij}.

Because of the respondent shift, coefficients estimated by fs_betareg() or fs_logit(), which do not model it, are attenuated towards zero relative to coef; ratios such as willingness to pay are much less affected.

Value

The schedule as a data frame (class fs_assignment retained) with the response column added. The coefficients, phi, sd_respondent and scale used are attached as the truth attribute.

See Also

fs_power(), which repeats the simulation to estimate power.

Examples

```
design <- fs_defficient(
  list(damage = c("minor", "severe"), cost = c(100, 300, 500)),
  n_vignettes = 6, n_blocks = 2, seed = 1, verbose = FALSE
)
sim <- fs_simulate_ratings(
  design, n_respondents = 60,
  coef = c("(Intercept)" = 1, damagesevere = 0.8, cost = -0.004),
  formula = ~ damage + cost, seed = 1
)
head(sim)

fit <- fs_betareg(rating ~ damage + cost, data = sim,
                  cluster = ~ respondent, compute_me = FALSE)
fs_mwtp(fit, price = "cost")   # true value: 0.8 / 0.004 = 200
```

`fs_vignette_text` *Turn design rows into vignette prose*

Description

Fills a sentence template with the attribute levels of each row, producing the text that will be shown to respondents. Placeholders are written `{{attribute}}`, and optional formatting functions let numbers be rendered as currency, dates, or anything else, without changing the design itself.

Usage

```
fs_vignette_text(
  design,
  template,
  column = "vignette_text",
  format = NULL,
  na_text = ""
)
```

Arguments

`design`	An object of class `fs_design` from `fs_defficient()`, an `fs_assignment` from `fs_assign()`, or any data frame.
`template`	A single character string containing the sentence, with `{{attribute}}` where a level should appear. Whitespace inside the braces is ignored, so `{{ cost }}` and `{{cost}}` are the same placeholder. Non-ASCII templates are supported and are treated as UTF-8.
`column`	The name of the column to add. Defaults to `"vignette_text"`.
`format`	An optional named list of functions, one per attribute, each taking the column's values and returning a character vector of the same length. Use this to render numbers the way respondents should read them.
`na_text`	The text substituted for missing values.

Details

Substitution is literal: whatever the formatting function returns is placed in the sentence unchanged. Word order, particles and agreement are the template author's responsibility, which keeps the behaviour predictable in any language. Where a level needs different wording in different sentences, the usual approach is to apply `set_labels()` first, or to pass a formatting function here.

Value

The input data frame with the text column added, keeping its class and attributes. The template is attached as the `template` attribute.

See Also

set_labels() for attaching level labels, fs_export() for writing the result out.

Examples

```
d <- fs_defficient(list(damage = c("minor", "severe"),
                        cost   = c(100, 1000)),
                   n_vignettes = 4, seed = 1, verbose = FALSE)

d <- fs_vignette_text(
  d,
  "This building suffered {{damage}} damage. Repair costs {{cost}}."
)
d$vignette_text

# Render the cost as currency without altering the design column
d2 <- fs_vignette_text(
  d,
  "Repairs cost {{cost}} per year.",
  column = "cost_sentence",
  format = list(cost = function(x) paste0("$", formatC(x, big.mark = ",")))
)
d2$cost_sentence
```

---

fs_williams *Construct a Williams design*

Description

Builds a set of presentation orders in which every item appears equally often in every position **and** every ordered pair of items appears equally often as immediate neighbours. The second property is what distinguishes a Williams design from an ordinary Latin square: it balances first-order carry-over, so that the effect of having just seen one vignette on the rating of the next does not confound any attribute.

Usage

```
fs_williams(k)
```

Arguments

k	The number of items to be ordered – in a factorial survey, the number of vignettes one respondent sees.

Details

The first order is built as $1, 2, k, 3, k-1, 4, \ldots$, and the remaining orders are cyclic shifts of it. For even k this single square is already balanced. For odd k no single square can be, so the square is paired with its mirror image, and each ordered pair of neighbours then occurs exactly twice.

Value

An integer matrix with k columns. Each row is one presentation order, giving the item shown in each position. There are k rows when k is even and 2 * k rows when k is odd, which is the smallest number of orders that achieves carryover balance.

References

Williams, E. J. (1949). Experimental designs balanced for the estimation of residual effects of treatments. *Australian Journal of Scientific Research, Series A*, 2(2), 149–168. doi:10.1071/CH9490149

See Also

`fs_assign()`, which uses this to allocate orders to respondents.

Examples

```
# Even k: one square is enough
fs_williams(4)

# Odd k: the square is paired with its mirror image
fs_williams(3)

# Every item appears once in every position
table(fs_williams(4)[, 1])
```

---

plot.fs_power *Plot a power analysis*

Description

Draws power against the number of respondents, one panel per model term, with a dashed line at 0.8.

Usage

```
## S3 method for class 'fs_power'
plot(x, ...)
```

Arguments

x	An object of class fs_power, from `fs_power()`.
...	Ignored.

Value

A ggplot object.

Examples

```
design <- fs_defficient(list(a = c("x", "y"), cost = c(1, 2, 3)),
                        n_vignettes = 6, seed = 1, verbose = FALSE)
pw <- fs_power(design, c(20, 60), coef = c("(Intercept)" = 0, ay = 0.5,
                                           cost = -0.3),
               formula = ~ a + cost, n_sim = 10, seed = 1,
               verbose = FALSE)
plot(pw)
```

---

print.fs_assignment *Print a respondent schedule*

Description

Print a respondent schedule

Usage

```
## S3 method for class 'fs_assignment'
print(x, n = 10L, ...)
```

Arguments

x	An object of class fs_assignment, from fs_assign().
n	The number of rows to show.
...	Ignored.

Value

x, invisibly. Called for the side effect of printing.

Examples

```
d <- fs_defficient(list(a = 1:2, b = 1:3), n_vignettes = 6,
                   seed = 1, verbose = FALSE)
print(fs_assign(d, n_respondents = 4, seed = 1))
```

print.fs_betareg *Print a fitted factorial survey beta regression*

Description

Print a fitted factorial survey beta regression

Usage

```
## S3 method for class 'fs_betareg'
print(x, digits = 4, ...)
```

Arguments

x	An object of class fs_betareg.
digits	Number of significant digits.
...	Ignored.

Value

x, invisibly. Called for the side effect of printing.

Examples

```
set.seed(1)
d <- data.frame(id = rep(1:20, each = 3), x = rnorm(60))
d$rating <- 10 * plogis(0.5 * d$x + rnorm(60, sd = 0.4))
print(fs_betareg(rating ~ x, data = d, cluster = ~ id,
                 compute_me = FALSE))
```

print.fs_design *Print a factorial survey design*

Description

Print a factorial survey design

Usage

```
## S3 method for class 'fs_design'
print(x, n = 10L, ...)
```

Arguments

x	An object of class fs_design, from fs_defficient().
n	The number of rows to show.
...	Ignored.

Value

x, invisibly. Called for the side effect of printing.

Examples

```
d <- fs_defficient(list(a = 1:2, b = 1:3), n_vignettes = 6,
                   seed = 1, verbose = FALSE)
print(d)
```

---

print.fs_design_eval *Print design diagnostics*

Description

Print design diagnostics

Usage

```
## S3 method for class 'fs_design_eval'
print(x, digits = 3, ...)
```

Arguments

x	An object of class fs_design_eval, from fs_design_eval().
digits	Number of digits used for the correlation matrix.
...	Ignored.

Value

x, invisibly. Called for the side effect of printing.

Examples

```
d <- fs_defficient(list(a = 1:2, b = 1:3), n_vignettes = 6,
                   seed = 1, verbose = FALSE)
print(fs_design_eval(d))
```

print.fs_efficiency *Print design efficiencies*

Description

Print design efficiencies

Usage

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

Arguments

x	An object of class fs_efficiency, from fs_efficiency().
...	Ignored.

Value

x, invisibly. Called for the side effect of printing.

Examples

```
print(fs_efficiency(expand.grid(a = 1:2, b = 1:3)))
```

print.fs_ewtp *Print expected willingness to pay estimates*

Description

Print expected willingness to pay estimates

Usage

```
## S3 method for class 'fs_ewtp'
print(x, digits = 4, ...)
```

Arguments

x	An object of class fs_ewtp, from fs_ewtp().
digits	Number of significant digits.
...	Ignored.

Value

x, invisibly. Called for the side effect of printing.

Examples

```
set.seed(1)
d <- data.frame(id = rep(1:40, each = 4), q = rbinom(160, 1, 0.5),
                cost = sample(c(100, 300, 500), 160, replace = TRUE))
d$y <- rbinom(160, 1, plogis(0.5 + d$q - 0.004 * d$cost))
fit <- fs_logit(y ~ q + cost, data = d, cluster = ~ id, compute_me = FALSE)
print(fs_ewtp(fit, data.frame(q = 0:1), price = "cost"))
```

---

print.fs_logit *Print a fitted factorial survey logit model*

Description

Print a fitted factorial survey logit model

Usage

```
## S3 method for class 'fs_logit'
print(x, digits = 4, ...)
```

Arguments

x	An object of class fs_logit.
digits	Number of significant digits.
...	Ignored.

Value

x, invisibly. Called for the side effect of printing.

Examples

```
set.seed(1)
d <- data.frame(id = rep(1:30, each = 4), x = rnorm(120))
d$y <- rbinom(120, 1, plogis(0.8 * d$x))
print(fs_logit(y ~ x, data = d, cluster = ~ id, compute_me = FALSE))
```

print.fs_mwtp *Print marginal willingness to pay estimates*

Description

Print marginal willingness to pay estimates

Usage

```
## S3 method for class 'fs_mwtp'
print(x, digits = 4, ...)
```

Arguments

x	An object of class fs_mwtp, from fs_mwtp().
digits	Number of significant digits.
...	Ignored.

Value

x, invisibly. Called for the side effect of printing.

Examples

```
set.seed(1)
d <- data.frame(id = rep(1:40, each = 3), x = rnorm(120),
                price = runif(120, 100, 900))
d$rating <- 10 * plogis(0.7 * d$x - 0.002 * d$price + rnorm(120, sd = 0.4))
fit <- fs_betareg(rating ~ x + price, data = d, cluster = ~ id,
                  compute_me = FALSE)
print(fs_mwtp(fit, price = "price"))
```

print.fs_power *Print a power analysis*

Description

Print a power analysis

Usage

```
## S3 method for class 'fs_power'
print(x, digits = 2, ...)
```

Arguments

x	An object of class fs_power, from fs_power().
digits	Number of digits.
...	Ignored.

Value

x, invisibly. Called for the side effect of printing.

Examples

```
design <- fs_defficient(list(a = c("x", "y"), cost = c(1, 2, 3)),
                        n_vignettes = 6, seed = 1, verbose = FALSE)
print(fs_power(design, 40, coef = c("(Intercept)" = 0, ay = 0.5,
                                    cost = -0.3),
               formula = ~ a + cost, n_sim = 10, seed = 1,
               verbose = FALSE))
```

---

print.summary.fs_betareg

Print the summary of a factorial survey beta regression

Description

Print the summary of a factorial survey beta regression

Usage

```
## S3 method for class 'summary.fs_betareg'
print(x, digits = 4, ...)
```

Arguments

x	An object of class summary.fs_betareg.
digits	Number of significant digits.
...	Ignored.

Value

x, invisibly. Called for the side effect of printing.

Examples

```
set.seed(1)
d <- data.frame(id = rep(1:20, each = 3), x = rnorm(60))
d$rating <- 10 * plogis(0.5 * d$x + rnorm(60, sd = 0.4))
print(summary(fs_betareg(rating ~ x, data = d, cluster = ~ id,
               compute_me = FALSE)))
```

print.summary.fs_logit *Print the summary of a factorial survey logit model*

Description

Print the summary of a factorial survey logit model

Usage

```
## S3 method for class 'summary.fs_logit'
print(x, digits = 4, ...)
```

Arguments

x	An object of class summary.fs_logit.
digits	Number of significant digits.
...	Ignored.

Value

x, invisibly. Called for the side effect of printing.

Examples

```
set.seed(1)
d <- data.frame(id = rep(1:30, each = 4), x = rnorm(120))
d$y <- rbinom(120, 1, plogis(0.8 * d$x))
print(summary(fs_logit(y ~ x, data = d, cluster = ~ id,
                       compute_me = FALSE)))
```

set_labels *Attach readable level labels to design or survey variables*

Description

Converts one or more columns of a data frame into factors with explicit labels. This is normally applied to a design produced by fs_defficient(), whose columns carry the level *codes* used in the questionnaire rather than the wording shown to respondents, or to a returned data set in which the attributes are stored as integer codes.

Usage

```
set_labels(df, label_list, levels_list = NULL, strict = TRUE)
```

Arguments

df	A data frame.
label_list	A named list of character vectors. Each name must match a column of df, and each element gives the labels for that column's levels, in the order given by levels_list.
levels_list	An optional named list giving, for each variable, the values in df that the labels correspond to. Names must be a subset of the names of label_list. When a variable is not listed, the values are taken to be 1, 2, ..., length(labels) if the column is numeric, and sort(unique(x)) otherwise.
strict	A logical value. If TRUE (the default), an error is thrown when a column contains values that are not covered by the supplied levels. If FALSE, uncovered values become NA with a warning.

Value

The input data frame, with the named columns replaced by labelled factors. Other columns and all attributes of df are left unchanged.

See Also

fs_defficient() for generating the design these labels describe.

Examples

```
d <- data.frame(alert = c(1, 2, 1), timing = c(2, 1, 2), rating = c(7, 3, 8))

# Numeric codes 1, 2 are matched to labels in order
set_labels(d, list(alert = c("Low", "High"),
                   timing = c("Morning", "Evening")))

# Codes that do not start at 1 are given explicitly
d0 <- data.frame(alert = c(0, 1, 0))
set_labels(d0, list(alert = c("Absent", "Present")),
           levels_list = list(alert = c(0, 1)))
```

---

summary.fs_betareg *Summarise a fitted factorial survey beta regression*

Description

Summarise a fitted factorial survey beta regression

Usage

```
## S3 method for class 'fs_betareg'
summary(object, ...)
```

Arguments

object	An object of class fs_betareg.
...	Ignored.

Value

An object of class summary.fs_betareg, a list holding the **betareg** summary, the fit measures and the rescaled marginal effects.

Examples

```
set.seed(1)
d <- data.frame(id = rep(1:20, each = 3), x = rnorm(60))
d$rating <- 10 * plogis(0.5 * d$x + rnorm(60, sd = 0.4))
summary(fs_betareg(rating ~ x, data = d, cluster = ~ id,
               compute_me = FALSE))
```

---

summary.fs_logit *Summarise a fitted factorial survey logit model*

Description

Summarise a fitted factorial survey logit model

Usage

```
## S3 method for class 'fs_logit'
summary(object, ...)
```

Arguments

object	An object of class fs_logit.
...	Ignored.

Value

An object of class summary.fs_logit, a list holding the coefficient table (with cluster-robust standard errors when the model was fitted with cluster), the fit measures and the marginal effects.

Examples

```
set.seed(1)
d <- data.frame(id = rep(1:30, each = 4), x = rnorm(120))
d$y <- rbinom(120, 1, plogis(0.8 * d$x))
summary(fs_logit(y ~ x, data = d, cluster = ~ id, compute_me = FALSE))
```

`theme_fs` *A plain theme for factorial survey plots*

Description

The theme used by the package's plot methods: white background, light horizontal guides, no minor grid and no vertical grid. Add it to your own plots for a consistent look.

Usage

```
theme_fs(base_size = 11, base_family = "")
```

Arguments

`base_size`	Base font size, in points.
`base_family`	Base font family.

Value

A `ggplot2::theme()` object.

Examples

```
library(ggplot2)
ggplot(mtcars, aes(wt, mpg)) + geom_point() + theme_fs()
```

Index

∗ **datasets**
bento, 4

AlgDesign::optBlock(), *12, 13*
AlgDesign::optFederov(), *11*, *16*
as.data.frame(), *3, 4*
as.data.frame.fs_ewtp, 3
as.data.frame.fs_mwtp, 4

bento, 4
betareg::betareg(), *7, 8*, *10*, *17*, *24*

coef(), *24*
coef.fs_betareg *(*fs_betareg-extractors*)*, 10
coef.fs_logit *(*fs_logit-extractors*)*, 23

expand.grid(), *11*

fitted.fs_betareg *(*fs_betareg-extractors*)*, 10
fitted.fs_logit *(*fs_logit-extractors*)*, 23
fs_assign, 5
fs_assign(), *12, 13*, *19, 20*, *28*, *30*, *32*, *34, 35*
fs_betareg, 7
fs_betareg(), *17, 18*, *21, 22*, *24–27*, *31*
fs_betareg-extractors, 10
fs_defficient, 11
fs_defficient(), *6*, *9*, *14–16*, *19, 20*, *28–30*, *32*, *36*, *42, 43*
fs_design_eval, 14
fs_design_eval(), *13*, *16*, *37*
fs_efficiency, 15
fs_efficiency(), *12–14*, *38*
fs_ewtp, 17
fs_ewtp(), *3*, *21, 22*, *26*, *38*
fs_export, 19
fs_export(), *6*, *12*, *33*
fs_logit, 21
fs_logit(), *17, 18*, *26, 27*, *31*
fs_logit-extractors, 23
fs_mwtp, 24
fs_mwtp(), *4*, *8, 9*, *17*, *19*, *21, 22*, *26*, *40*
fs_plot, 26
fs_power, 27
fs_power(), *30, 31*, *34*, *41*
fs_simulate_ratings, 30
fs_simulate_ratings(), *27*, *29*
fs_vignette_text, 32
fs_vignette_text(), *6*, *13*, *20*
fs_williams, 33
fs_williams(), *6*

ggplot2::ggplot(), *26*
ggplot2::theme(), *45*

logLik.fs_betareg *(*fs_betareg-extractors*)*, 10
logLik.fs_logit *(*fs_logit-extractors*)*, 23

message(), *8*, *12*, *21*, *28*

nobs.fs_betareg *(*fs_betareg-extractors*)*, 10
nobs.fs_logit *(*fs_logit-extractors*)*, 23

plot.fs_betareg *(*fs_plot*)*, 26
plot.fs_ewtp *(*fs_plot*)*, 26
plot.fs_logit *(*fs_plot*)*, 26
plot.fs_mwtp *(*fs_plot*)*, 26
plot.fs_power, 34
print.fs_assignment, 35
print.fs_betareg, 36
print.fs_design, 36
print.fs_design_eval, 37
print.fs_efficiency, 38
print.fs_ewtp, 38
print.fs_logit, 39
print.fs_mwtp, 40
print.fs_power, 40

print.summary.fs_betareg, 41
print.summary.fs_logit, 42

sandwich::vcovCL(), *8*
set_labels, 42
set_labels(), *13*, *32*, *33*
stats::glm(), *17*, *21–23*
stats::model.matrix(), *28*, *30*
summary.fs_betareg, 43
summary.fs_logit, 44

theme_fs, 45

vcov(), *24*
vcov.fs_betareg
*(*fs_betareg-extractors*)*, 10
vcov.fs_logit *(*fs_logit-extractors*)*, 23