Package 'optimall'

February 15, 2024

Type Package

Title Allocate Samples Among Strata

Version 0.1.5

Maintainer Jasper Yang <jbyang@uw.edu>

Description Functions for the design process of survey sampling, with specific tools for multi-wave and multi-phase designs. Perform optimum allocation using Neyman (1934) <doi:10.2307/2342192> or Wright (2012) <doi:10.1080/00031305.2012.733679> allocation, split strata based on quantiles or values of known variables, randomly select samples from strata, allocate sampling waves iteratively, and organize a complex survey design. Also includes a Shiny application for observing the effects of different strata splits.

License GPL-3

URL https://github.com/yangjasp/optimall

BugReports https://github.com/yangjasp/optimall/issues

Depends R (>= 3.5.0)

Imports dplyr (>= 1.0.5), glue (>= 1.4.0), magrittr (>= 2.0.0), methods (>= 4.0.0), rlang (>= 0.2.2), stats (>= 4.0.2), tibble (>= 1.4.2), utils (>= 3.5.0),

Suggests bslib (>= 0.2.4), DiagrammeR (>= 1.0.0), DT (>= 0.15), datasets, globals (>= 0.12), knitr (>= 1.28), rmarkdown (>= 2.7), shiny (>= 1.6.0), shinytest (>= 1.4.0), survey (>= 4.0), testthat (>= 3.0.2), webshot (>= 0.5)

VignetteBuilder knitr

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

Collate 'allocate_wave.R' 'sample_strata.R' 'wave.R' 'phase.R' 'multiwave.R' 'merge_samples.R' 'optimum_allocation.R' 'get_data.R' 'apply_multiwave.R' 'matwgt_sim.R' 'merge_strata.R' 'multiwave_diagram.R' 'new_multiwave.R' 'optimall_shiny.R' 'split_strata.R' 'summary.multiwave.R'

NeedsCompilation no

Author Jasper Yang [aut, cre],
Pamela Shaw [aut],
Bryan Shepherd [ctb],
Thomas Lumley [ctb],
Gustavo Amorim [rev]

Repository CRAN

Date/Publication 2024-02-15 18:50:02 UTC

R topics documented:

`allocate_wave` *Adaptive Multi-Wave Sampling*

Description

Determines the adaptive optimum sampling allocation for a new sampling wave based on results from previous waves. Using Neyman or Wright (2014) allocation, `allocate_wave` calculates the optimum allocation for the *total* number of samples across waves, determines how many were allocated to each strata in previous waves, and allocates the remaining samples to make up the difference.

Usage

```
allocate_wave(
  data,
  strata,
  y,
  already_sampled,
  nsample,
  allocation_method = c("WrightII", "WrightI", "Neyman"),
  method = c("iterative", "simple"),
  detailed = FALSE
)
```

Arguments

`data`	A data frame or matrix with one row for each sampling unit, one column specifying each unit's stratum, one column holding the value of the continuous variable for which the variance should be minimized, and one column containing a binary indicator, `already_sampled`, specifying whether each unit has already been sampled.
`strata`	A character string or vector of character strings specifying the name of columns that indicate the stratum that each unit belongs to.
`y`	A character string specifying the name of the continuous variable for which the variance should be minimized.
`already_sampled`	A character string specifying the name of a column that contains a binary (`Y`/`N` or `1`/`0`) indicator specifying whether each unit has already been sampled in a previous wave.
`nsample`	The desired sample size of the next wave.
`allocation_method`	A character string specifying the method of optimum sample allocation to use. For details see `optimum_allocation()`. Defaults to `WrightII` which is more exact than `Neyman` but may run slower.
`method`	A character string specifying the method to be used if at least one group was oversampled. Must be one of: • `"iterative"`, the default, will require a longer runtime but may be a more precise method of handling oversampled strata. If there are multiple oversampled strata, this method closes strata and re-calculates optimum allocation one by one. • `"simple"` closes all oversampled together and re-calculates optimum allocation on the rest of the strata only once. In certain cases where many strata have been oversampled in prior waves, it is possible that this method will output a negative value in n_to_sample. When this occurs, the function will print a warning, and it is recommended that the user re-runs the allocation with the 'iterative' method.
`detailed`	A logical value indicating whether the output dataframe should include details about each stratum including the true optimum allocation without the constraint

of previous waves of sampling and stratum standard deviations. Defaults to FALSE. These details are all available from `optimum_allocation()`.

Details

If the optimum sample size in a stratum is smaller than the amount it was allocated in previous waves, that strata has been *oversampled*. When oversampling occurs, `allocate_wave` "closes" the oversampled strata and re-allocates the remaining samples optimally among the open strata. Under these circumstances, the total sampling allocation is no longer optimal, but `optimall` will output the *most* optimal allocation possible for the next wave.

Value

Returns a dataframe with one row for each stratum and columns specifying the stratum name ("strata"), population stratum size (`"npop"`), cumulative sample in that strata (`"nsample_actual"`), prior number sampled in that strata (`"nsample_prior"`), and the optimally allocated number of units in each strata for the next wave (`"n_to_sample"`).

References

McIsaac MA, Cook RJ. Adaptive sampling in two-phase designs: a biomarker study for progression in arthritis. Statistics in medicine. 2015 Sep 20;34(21):2899-912.

Reilly, M., & Pepe, M. S. (1995). A mean score method for missing and auxiliary covariate data in regression models. Biometrika, 82(2), 299-314.

Wright, T. (2014). A Simple Method of Exact Optimal Sample Allocation under Stratification with any Mixed Constraint Patterns, Research Report Series (Statistics #2014-07), Center for Statistical Research and Methodology, U.S. Bureau of the Census, Washington, D.C.

Examples

```
# Create dataframe with a column specifying strata, a variable of interest
# and an indicator for whether each unit was already sampled
set.seed(234)
mydata <- data.frame(Strata = c(rep(1, times = 20),
                                rep(2, times = 20),
                                rep(3, times = 20)),
                     Var = c(rnorm(20, 1, 0.5),
                             rnorm(20, 1, 0.9),
                             rnorm(20, 1.5, 0.9)),
                     AlreadySampled = rep(c(rep(1, times = 5),
                                            rep(0, times = 15)),
                                          times = 3))

x <- allocate_wave(
  data = mydata, strata = "Strata",
  y = "Var", already_sampled = "AlreadySampled",
  nsample = 20, method = "simple"
)
```

apply_multiwave *Apply a basic optimall function to a Multiwave Object*

Description

Given a specified phase and wave of an object of class multiwave, apply_multiwave applies one of four optimall functions and returns an updated multiwave object with the output of the applied function in its specified slot.

Usage

```
apply_multiwave(x, phase, wave, fun, ...)

## S4 method for signature 'Multiwave'
apply_multiwave(x, phase, wave, fun, ...)
```

Arguments

x	An Object of class "multiwave"
phase	A numeric or character value specifying the phase of multiwave where the desired output should be placed.
wave	A numeric or character value specifying the wave of phase in multiwave where the output should be placed.
fun	A character value specifying the name of the optimall function to apply. The four available functions are: optimum_allocation, allocate_wave, sample_strata, and merge_samples.

- optimum_allocation: Uses the data from the previous wave (or previous phase if wave = 1) to determine the optimum sampling allocation for the specified wave. If used, the output multiwave object contains an updated "design" slot in the specified wave.
- allocate_wave: Uses the data from the previous wave (or previous phase if wave = 1) to determine the optimum sampling allocation for the specified wave. If used, the outputted multiwave object contains an updated "design" slot in the specified wave. The default argument when allocate_wave is applied in a apply_multiwave() is detailed = TRUE.
- sample_strata: Uses the data from the previous wave (or previous phase if wave = 1) and design from current wave to generate a vector of ids to sample for the current wave. If used, the output multiwave object contains an updated "samples" slot in the specified wave.
- merge_samples: Uses the data from the previous wave (or previous phase if wave = 1) and sampled_data from the specified wave to generate the final, merged data for the current wave. If used, the output multiwave object contains an updated "data" slot in the specified wave. Note that merge_samples is already a method for multiwave objects, so calling it through apply_multiwave is the exact same as calling it on its own.

See documentation of these functions for more details on the specific uses and arguments.

`...` Optional arguments to be given to `fun`. Not necessary if the arguments are already provided as named values in the wave, phase, or overall metadata in the multiwave object. Arguments provided here will override specifications in the metadata if provided in both places.

Value

The inputted multiwave object with one slot updated to include the output of the specified function.

Note that the phase and wave arguments specify where the function *output* should be placed. `apply_multiwave` will determine where to get the input dataframes from (returning an error if those slots are empty or invalid) given the specified wave for the output. For example, if `phase = 2, wave = 2, function = "allocate_wave"`, the data to determine the optimum allocation will be taken from the previous wave (phase 2, wave 1) and the output multiwave object will have an updated `"design"` slot of phase 2, wave 2.

Examples

```
library(datasets)

MySurvey <- new_multiwave(phases = 2, waves = c(1, 3))
get_data(MySurvey, phase = 1, slot = "data") <-
  dplyr::select(datasets::iris, -Sepal.Width)

# Get Design by applying optimum_allocation
MySurvey <- apply_multiwave(MySurvey,
  phase = 2, wave = 1,
  fun = "optimum_allocation", strata = "Species",
  y = "Sepal.Length",
  nsample = 15,
  method = "WrightII"
)

# or, we can establish function args in the metadata
get_data(MySurvey, phase = 2, slot = "metadata") <- list(
  strata = "Species",
  nsample = 15,
  y = "Sepal.Length",
  method = "WrightII"
)

# which allows the function to be run without specifying the args
MySurvey <- apply_multiwave(MySurvey,
  phase = 2, wave = 1,
  fun = "optimum_allocation"
)
```

get_data *Access and Write Slots of a Multiwave Object*

Description

get_data is the accessor function for objects of class Multiwave. It can be used to access or write slots.

Usage

```
get_data(
  x,
  phase = 1,
  wave = NA,
  slot = c("data", "design", "metadata", "samples", "sampled_data")
)

get_data(
  x,
  phase = 1,
  wave = NA,
  slot = c("data", "design", "metadata", "samples", "sampled_data")
) <- value
```

Arguments

x	an object of class 'Multiwave'
phase	a numeric value specifying the phase that should be accessed. To access the overall metadata, set phase = NA. Defaults to 1.
wave	a numeric value specifying the wave that should be accessed. Ta access phase metadata, set wave = NA. Defaults to NA.
slot	a character value specifying the name of the slot to be accessed. Must be one of "metadata", "design", "samples", "sampled_data", "data". Defaults to "data". See class documentation or package vignettes for more information about slots.
value	value to assign to specified slot

Value

If accessing a multiwave object slot, returns the specified slot.

Functions

- get_data(): access slot of multiwave object
- get_data(x, phase = 1, wave = NA, slot = c("data", "design", "metadata", "samples", "sampled_data")) <- value: assign value to slot of a multiwave object

Examples

```
# Intiate multiwave object
MySurvey <- new_multiwave(phases = 2, waves = c(1, 3))

# To access overall metadata
get_data(MySurvey, phase = NA, slot = "metadata")

# To write overall metadata
get_data(MySurvey, phase = NA, slot = "metadata") <- list(
  title = "Maternal Weight Survey"
)

# To access Phase 2 metadata
get_data(MySurvey, phase = 2, slot = "metadata")

# To access Phase 2, Wave 2 design
get_data(MySurvey, phase = 2, wave = 2, slot = "design")
```

`MatWgt_Sim` *Example Dataset: Maternal Weights*

Description

This SIMULATED dataset contains data on demographic characteristics and clinical data related to childhood obesity for 10335 mother-child pairs. It is used to generate the workflow in the main package vignette. It is based on a study that used multi-wave adaptive sampling to validate electronic health records that target factors related to childhood obesity (see https://www.pcori.org/research-results/2017/developin-methods-estimate-and-address-errors-studies-using-electronic-health).

Format

`MatWgt_Sim`: a data frame with 10335 rows and 6 columns

`id` unique ID for each mother-child pair

`mat_weight_true` true (but unknown in phase 1) mother weight change during pregnancy

`mat_weight_est` estimated mother weight change during pregnancy based on error-prone phase-1 measurement

`race` specifies mother's race

`diabetes` binary indicator for diabetes in the mother

`obesity` binary indicator for childhood obesity in child

Details

See package vignettes for more details.

merge_samples *Merge Sampled Data based on IDs*

Description

In an object of class "Mutiwave", merge_samples creates a dataframe in the "data" slot of the specified wave by merging the dataframe in the "sampled data" slot with the dataframe in the "data" slot of the previous wave.

Usage

```
merge_samples(x, phase, wave, id = NULL, sampled_ind = "already_sampled_ind")
```

Arguments

x	an object of class "Multiwave".
phase	A numeric value specifying the phase of the Multiwave object that the specified wave is in. Cannot be phase 1.
wave	A numeric value specifying the wave of the Multiwave object that the merge should be performed in. This wave must have a valid dataframe in the "sampled data" slot. The previous wave, taken as the final wave of the previous phase if wave = 1, must have a valid dataframe in the "data" slot.
id	A character value specifying the name of the column holding unit ids. Taken from wave, phase, or overall metadata (searched for in that order) if NULL. Defaults to NULL.
sampled_ind	a character value specifying the name of the column that should hold the indicator of whether each unit has already been sampled in the current phase.

Details

If a column name in the "sampled data" matches a column name in the "data" slot of the previous wave, these columns will be merged into one column with the same name in the output dataframe. For ids that have non-missing values in both columns of the merge, the value from "sampled_data" will overwrite the previous value and a warning will be printed. All ids present in the "data" from the previous wave but missing from "sampled_data" will be given NA values for the newly merged variables

Columns in "sampled_data" that do not match names of the "data" from the previous wave will be added as new columns in the output dataframe. All ids that do not appear in "sampled_data" will receive NA values for these new variables.

Value

A Multiwave object with the merged dataframe in the "data" slot of the specified wave.

Examples

```
library(datasets)
iris <- data.frame(iris, id = 1:150)

MySurvey <- new_multiwave(phases = 2, waves = c(1, 3))
get_data(MySurvey, phase = 1, slot = "data") <-
  data.frame(dplyr::select(iris, -Sepal.Width))
get_data(MySurvey, phase = 2, wave = 1, slot = "sampled_data") <-
  dplyr::select(iris, id, Sepal.Width)[1:40, ]
MySurvey <- merge_samples(MySurvey, phase = 2, wave = 1, id = "id")
```

---

merge_strata *Merge Strata*

Description

Merges multiple pre-defined sampling strata into a single stratum.

Usage

```
merge_strata(data, strata, merge, name = NULL)
```

Arguments

data	a dataframe or matrix with one row for each sampling unit, one column, strata, specifying each unit's current stratum, and any other relevant columns.
strata	a character string specifying the name of the column that defines each unit's current strata.
merge	the names of the strata to be merged, exactly as they appear in strata.
name	a character name for the new stratum. Defaults to NULL, which pastes the old strata names together to create the new stratum name.

Value

Returns the input dataframe with a new column named 'new_strata' that holds the name of the stratum that each sample belongs to after the merge. The column containing the previous strata names is retained and given the name 'old_strata'.

Examples

```
x <- merge_strata(iris,
  strata = "Species",
  merge = c("virginica", "versicolor"), name = "v_species"
)
```

Multiwave-class *Multiwave Class for Multi-Wave Sampling Organization*

Description

optimall defines three S4 classes for organizing the multi-wave sampling workflow: `Wave`, `Phase`, and `Multiwave`. An object of class `Multiwave` holds metadata and a list of objects of class `Phase`, which in turn holds metadata and a list of objects of class `Wave`. These three object classes are used together to organize the workflow of multi-wave sampling designs.

Slots

`metadata` A list of elements that describe the entire survey. The list is empty upon initialization of the multiwave object, but the user may add anything to it as they see fit. It may include a "title".

`phases` A list of objects of class `Phase` (see other class documentation).

multiwave_diagram *Print Summary Diagram of Multiwave Object*

Description

Takes a multiwave object as input and plots a diagram of its structure in the plotting window using `grViz()` from the `DiagrammeR` package. Red boxes indicate slots that have not yet been filled, blue boxes indicate that the slot is filled.

Usage

```
multiwave_diagram(x, height = NULL, width = NULL)
```

Arguments

`x`	An object of class `multiwave`.
`height`	The height in pixels of the diagram. Defaults to `NULL` , which produces default height.
`width`	The width in pixels of the diagram. Defaults to `NULL`, which produces the default width.

Value

Returns an object of class `htmlwidget` displaying the structure of the `x`.

Examples

```
MySurvey <- new_multiwave(phases = 2, waves = c(1, 3))
multiwave_diagram(MySurvey)
```

new_multiwave *Initialize a Multiwave Object*

Description

Creates an Object of Class `Multiwave` with the specified number of phases and waves. All contents will be NULL upon initialization, but the object contains a framework for contents to be added to during the survey design and sample collection process. Currently, multiwave objects may only have one wave in Phase 1.

Usage

```
new_multiwave(phases, waves, metadata = list(), phase1 = data.frame())
```

Arguments

`phases`	A numeric value specifying the number of phases in the survey design.
`waves`	A vector of numeric values specifying the number of waves in each phase of the survey design. Length must match the number of `phases` and the first
`metadata`	A list containing the survey metadata. Defaults to an empty list.
`phase1`	A dataframe containing the phase 1 data of the survey. Defaults to an empty dataframe.

Value

Returns an object of class `Multiwave` that stores all relevant data from the survey design in an organized and easy-to-access manner. See package vignettes or class documentation for more information.

Examples

```
# Initialize a multiwave object for a two-phase sampling design that will
# sample over three waves in the second phase
multiwave_object <- new_multiwave(phases = 2, waves = c(1, 3))

# If we already have the phase 1 data and want to add a title to the survey
# metadata, we can initialize the object with these included.

library(datasets)
multiwave_object <- new_multiwave(
  phases = 2, waves = c(1, 3),
  metadata = list(title = "my two-phase survey"), phase1 = iris
)
```

optimall_shiny *Run the shiny application*

Description

Launches an R Shiny application locally. This app can be used to interactively split strata and determine how the results affect optimum allocation of a fixed number of samples. It accepts .csv and .rds files as well as .rda files that contain a single dataset. See vignette titled "Splitting Strata with Optimall Shiny" for more information.

Usage

```
optimall_shiny(...)
```

Arguments

`...`	Optional arguments to pass to `shiny::runApp`. `display.mode` is already set to normal.

Value

Launches an R Shiny application locally.

optimum_allocation *Optimum Allocation*

Description

Determines the optimum sampling fraction and sample size for each stratum in a stratified random sample, which minimizes the variance of the sample mean according to Neyman Allocation or Exact Optimum Sample Allocation (Wright 2014).

Usage

```
optimum_allocation(
  data,
  strata,
  y = NULL,
  sd_h = NULL,
  N_h = NULL,
  nsample = NULL,
  ndigits = 2,
  method = c("WrightII", "WrightI", "Neyman"),
  allow.na = FALSE
)
```

Arguments

data	A data frame or matrix with at least one column specifying each unit's stratum, and either 1) a second column holding the value of the continuous variable for which the sample mean variance should be minimized (y) or 2) two columns: one holding the the within-stratum standard deviation for the variable of interest (sd_h) and another holding the stratum sample sizes (N_h). If data contains a column y holding values for the variable of interest, then data should have one row for each sampled unit. If data holds sd_h and N_h, the within-stratum standard deviations and population sizes, then data should have one row per stratum. Other columns are allowed but will be ignored.
strata	a character string or vector of character strings specifying the name(s) of columns which specify the stratum that each unit belongs to. If multiple column names are provided, each unique combination of values in these columns is taken to define one stratum.
y	a character string specifying the name of the continuous variable for which the variance should be minimized. Defaults to NULL and should be left as NULL when data holds stratum standard deviations and sample sizes instead of individual sampling units.
sd_h	a character string specifying the name of the column holding the within-stratum standard deviations for each stratum. Defaults to NULL and should be left as NULL when data holds individual sampling units.
N_h	a character string specifying the name of the column holding the population stratum sizes for each stratum. Defaults to NULL and should be left as NULL when data holds individual sampling units.
nsample	the desired total sample size. Defaults to NULL.
ndigits	a numeric value specifying the number of digits to which the standard deviation and stratum fraction should be rounded. Defaults to 2.
method	a character string specifying the method of optimum sample allocation to use. Must be one of: • "WrightII", the default, uses Algorithm II from Wright (2014) to determine the optimum allocation of a fixed sample size across the strata. It requires that at least two samples are allocated to each stratum. • "WrightI" uses Wright's Algorithm I to determine the optimum sample allocation. It only requires that at least one sample is allocated to each stratum, and can therefore lead to a biased variance estimate. • "Neyman" uses the standard method of Neyman Allocation to determine the optimum sample allocation. When nsample = NULL, the optimal sampling fraction is calculated and returned. When a numeric value is specified for nsample, then the number allocated to each stratum is the optimal sampling fraction times nsample rounded to the nearest integer, which may no longer be optimall.
allow.na	logical input specifying whether y should be allowed to have NA values. Defaults to FALSE.

Value

Returns a data frame with the specified total sample size, `nsample`, allocated across strata or the sampling fractions if nsample is NULL.

References

Wright, T. (2014). A Simple Method of Exact Optimal Sample Allocation under Stratification with any Mixed Constraint Patterns, Research Report Series (Statistics #2014-07), Center for Statistical Research and Methodology, U.S. Bureau of the Census, Washington, D.C.

Examples

```
optimum_allocation(
  data = iris, strata = "Species", y = "Sepal.Length",
  nsample = 40, method = "WrightII"
)

# Or if input data is summary of strata sd and N:
iris_summary <- data.frame(
  strata = unique(iris$Species),
  size = c(50, 50, 50),
  sd = c(0.3791, 0.3138, 0.3225)
)

optimum_allocation(
  data = iris_summary, strata = "strata",
  sd_h = "sd", N_h = "size",
  nsample = 40, method = "WrightII"
)
```

Phase-class *Phase Class for Multi-Wave Sampling Organization*

Description

optimall defines three S4 classes for organizing the multi-wave sampling workflow: `Wave`, `Phase`, and `Multiwave`. An object of class `Multiwave` holds metadata and a list of objects of class `Phase`, which in turn holds metadata and a list of objects of class `Wave`. These three object classes are used together to organize the workflow of multi-wave sampling designs.

Slots

`metadata` A list containing the phase metadata

`waves` A list of objects of class `Wave`, each element representing one wave of the phase

`sample_strata` *Select Sampling Units based on Stratified Random Sampling*

Description

Requires two dataframes or matrices: `data` with a column `strata` which specifies stratum membership for each unit in the population and a second dataframe `design_data` with one row per strata level with a column `design_strata` that indicates the unique levels of `strata` in `data` and `n_allocated` that specifies the number to be sampled from each stratum. `sample_strata` selects the units to sample by selecting a random sample of the desired size within each stratum. The second dataframe can be the output of `allocate_wave()` or `optimum_allocation()`.

Usage

```
sample_strata(
  data,
  strata,
  id,
  already_sampled = NULL,
  design_data,
  design_strata = "strata",
  n_allocated = "n_to_sample"
)
```

Arguments

`data`	A data frame or matrix with one row for each sampling unit in the population, one column specifying each unit's stratum, and one column with a unique identifier for each unit.
`strata`	a character string specifying the name of column in `data` which indicates stratum membership.
`id`	a character string specifying the name of the column in `data` that uniquely identifies each unit.
`already_sampled`	a character sting specifying the name of the column in `data` which indicates (1/0 or Y/N) whether a unit has already been sampled in a prior wave. Defaults to NULL which means that none have been sampled yet.
`design_data`	a dataframe or matrix with one row for each stratum that subdivides the population, one column specifying the stratum name, and one column indicating the number of samples allocated to each stratum.
`design_strata`	a character string specifying the name of the column in `design_data` that contains the stratum levels. Defaults to "strata".
`n_allocated`	a character string specifying the name of the column in `design_data` that indicates the n allocated to each stratum. Defaults to "n_to_sample".

Value

returns `data` as a dataframe with a new column named "sample_indicator" containing a binary (1/0) indicator of whether each unit should be sampled.

Examples

```
# Define a design dataframe
design <- data.frame(
  strata = c("setosa", "virginica", "versicolor"),
  n_to_sample = c(5, 5, 5)
)

# Make sure there is an id column
iris$id <- 1:nrow(iris)

# Run
sample_strata(
  data = iris, strata = "Species", id = "id",
  design_data = design, design_strata = "strata", n_allocated = "n_to_sample"
)

# If some units had already been sampled
iris$already_sampled <- rbinom(nrow(iris), 1, 0.25)

sample_strata(
  data = iris, strata = "Species", id = "id",
  already_sampled = "already_sampled",
  design_data = design, design_strata = "strata", n_allocated = "n_to_sample"
)
```

---

shiny_server *Server logic for Interactive Shiny for Optimall.*

Description

Server logic for Interactive Shiny for Optimall.

Usage

```
shiny_server(input, output, session)
```

Arguments

`input`	input for Shiny server.
`output`	output for by Shiny server.
`session`	session for Shiny server.

Value

Defines server logic for Shiny app that can be loaded with `optimall_shiny()`.

---

shiny_ui *UI for Shiny App for Splitting Strata with Optimum Allocation*

Description

UI for Shiny App for Splitting Strata with Optimum Allocation

Usage

```
shiny_ui()
```

Value

Creates the UI for the Shiny app that is loaded with `optimall_shiny`.

---

split_strata *Split Strata*

Description

Splits pre-defined sampling strata based on values of a continuous or categorical variable.

Usage

```
split_strata(
  data,
  strata,
  split = NULL,
  split_var,
  type = "global quantile",
  split_at = 0.5,
  trunc = NULL
)
```

Arguments

`data`	a dataframe or matrix with one row for each sampling unit, one column specifying each unit's current stratum, one column containing the continuous or categorical values that will define the split, and any other relevant columns.
`strata`	a character string specifying the name of the column that defines each unit's current strata.

split — the name of the stratum or strata to be split, exactly as they appear in strata. Defaults to NULL, which indicates that all strata in strata will be split.

split_var — a character string specifying the name of the column that should be used to define the strata splits.

type — a character string specifying how the function should interpret the split_at argument. Must be one of:

- "global quantile", the default, splits the strata at the quantiles specified in split_at defined along the entire, unfiltered split_var column.
- "local quantile" splits the strata at the quantiles specified in split_at defined along the filtered split_var column which only includes units in the stratum being split.
- "value" splits the strata at the values specified in split_at along split_var column.
- "categorical" splits the strata into two new strata, one that contains each unit where split_var matches an input of split_at, and a second that contains every other unit.

split_at — the percentile, value, or name(s) which split_var should be split at. The interpretation of this input depends on type. For "quantile" types, input must be between 0 and 1. Defaults to 0.5 (median). For "categorical" type, the input should be a vector of values or names in split_var that define the new stratum.

trunc — A numeric or character value specifying how the name of the split_var should be truncated when naming the new strata. If numeric, the new strata name will only include the first 'n' characters of the split_var name. If character, the specified string will be used to name the new strata instead of the split_var name. Defaults to NULL, which creates the new strata name using the entire name of the split_var column.

Details

For splits on continuous variables, the new strata are defined on left-open intervals. The only exception is the first interval, which must include the overall minimum value. The names of the newly created strata for a split generated from a continuous value are the split_var column name with the range of values defining that stratum appended to the old strata name. For a categorical split, the new strata names are the split_var column name appended to the 1/0 logical flag specifying whether the unit is in split at, all appended to the old strata name. If the split_var column name is long, the user can specify a value for trunc to prevent the new strata names from being inconveniently long.

Value

Returns the input dataframe with a new column named 'new_strata' that holds the name of the stratum that each sample belongs to after the split. The column containing the previous strata names is retained and given the name "old_strata".

Examples

```
x <- split_strata(iris, "Sepal.Length",
```

```
  strata = c("Species"),
  split = "setosa", split_var = "Sepal.Width",
  split_at = c(0.5), type = "global quantile"
)

# You can split at more than one quantile in one call.
# The above call splits the "setosa" stratum into three of equal size
x <- split_strata(iris, "Sepal.Length",
  strata = c("Species"),
  split = "setosa", split_var = "Sepal.Width", split_at = c(0.33, 0.66),
  type = "local quantile"
)

# Manually select split values with type = "value"
x <- split_strata(iris, "Sepal.Length",
  strata = "Species",
  split = "setosa", split_var = "Sepal.Width",
  split_at = c(3.1, 3.8), type = "value"
)

# Perform a categorical split.
iris$strata <- rep(c(rep(1, times = 25), rep(0, times = 25)), times = 3)
x <- split_strata(iris, "Sepal.Length",
  strata = "strata",
  split = NULL, split_var = "Species",
  split_at = c("virginica", "versicolor"), type = "categorical"
)
# Splits each initial strata 1 and 2 into one stratum with "virginia"
# and "versicolor" species and one stratum with all of the other species
# not specified in the split_at argument.
```

`summary,Multiwave-method`

Method for summary for class Multiwave

Description

Method for summary for class Multiwave

Usage

```
## S4 method for signature 'Multiwave'
summary(object)
```

Arguments

`object`	object of class "Multiwave"

Value

Prints a summary of the specified multiwave object in the console.

Wave-class *Wave Class for Multi-Wave Sampling Organization*

Description

optimall defines three S4 classes for organizing the multi-wave sampling workflow: `Wave`, `Phase`, and `Multiwave`. An object of class `Multiwave` holds metadata and a list of objects of class `Phase`, which in turn holds metadata and a list of objects of class `Wave`. These three object classes are used together to organize the workflow of multi-wave sampling designs.

Slots

`metadata` A list containing the metadata for the wave.

`design` a dataframe specifying the design of the wave. Is often the output of `allocate_wave`.

`samples` A character vector containing the ids of the units sampled in the wave.

`sampled_data` A dataframe holding the data, with ids, *collected* in this wave of sampling

`data` A dataframe holding the updated full data set with all of the Phase 1 sampling units including the samples collected in this wave.

Index