Package ‘splithalfr’

September 29, 2021

Title Estimate Split-Half Reliabilities

Version 2.2.0

Date 2021-09-28

Author Thomas Pronk [aut, cre]

Description Estimates split-half reliabilities for scoring algorithms of cognitive tasks and questionnaires. The 'splithalfr' supports researcher-provided scoring algorithms, with six vignettes illustrating how on included datasets. The package provides four splitting methods (first-second, odd-even, permutated, Monte Carlo), the option to stratify splits by task design, a number of reliability coefficients, and the option to sub-sample data.

Depends R (>= 3.6.0)

License GPL-3

Encoding UTF-8

LazyData true

Suggests knitr (>= 1.20), rmarkdown (>= 1.10), testthat (>= 2.1.0), MASS (>= 7.3.51)

Imports dplyr (>= 1.0.7), tibble (>= 2.1.1), psych (>= 1.8.12), bcaboot (>= 0.2.1), rlang (>= 0.4.0)

RoxygenNote 7.1.1

VignetteBuilder knitr

URL https://github.com/tpronk/splithalfr

BugReports https://github.com/tpronk/splithalfr/issues

NeedsCompilation no

Maintainer Thomas Pronk <pronkthomas@gmail.com>

Repository CRAN

Date/Publication 2021-09-29 04:30:01 UTC

R topics documented:

`angoff_feldt` *Calculate Angoff-Feldt coefficient*

Description

Angoff-Feldt reliability coefficient. Formula obtained from Warrens (2015) <doi: 10.1007/s11634-01501986>

Usage

```
angoff_feldt(x, y)
```

Arguments

x	(vector) a numeric vector
y	(vector) a numeric vector with compatible dimensions to x

Value

(numeric) Angoff-Feldt cefficient

See Also

Other splithalfr coefficients: assmd(), flanagan_rulon(), sdregi(), short_icc(), spearman_brown()

Examples

```
# Generate two variables with different means, variances and a correlation of about 0.5
library(MASS)
vars = mvrnorm(30, mu = c(0, 2), Sigma = matrix(c(5, 2, 2, 3), ncol = 2), empirical = TRUE)
# Calculate coefficient
angoff_feldt(vars[,1], vars[,2])
```

---

apply_split_indexes_to_strata

Split each element of a list of strata based on a list of indexes

---

Description

Splits each element of `strata` into two parts based on a list of indexes. For more information about splitting options, and an extensive list of examples, see get_split_indexes_from_stratum.

Usage

```
apply_split_indexes_to_strata(strata, indexes)
```

Arguments

strata	(list) list of strata to split
indexes	(list) list of indexes, which can be generated via get_split_indexes_from_strata

Value

(list) A list with two elements, containing the first and second split of `strata`.

See Also

Other splitting functions: apply_split_indexes_to_stratum(), check_strata(), get_split_indexes_from_strata(), get_split_indexes_from_stratum(), split_df(), split_strata(), split_stratum(), stratify()

Examples

```
# Stratify a data frame, then split it odd-even
ds <- data.frame(condition = rep(c("a", "b"), each = 4), score = 1 : 8)
strata <- stratify(ds, ds$condition)
split_indexes <- get_split_indexes_from_strata(strata, method = "odd_even")
apply_split_indexes_to_strata(strata, split_indexes)
```

apply_split_indexes_to_stratum

Split a stratum based on a list of indexes

Description

Splits stratum into two parts based on a list of indexes. For more information about splitting options, and an extensive list of examples, see get_split_indexes_from_stratum.

Usage

```
apply_split_indexes_to_stratum(stratum, indexes_1, indexes_2)
```

Arguments

stratum	(data frame, tibble, list, or vector) stratum to split
indexes_1	(vector) indexes for first split, which can be generated via get_split_indexes_from_stratum
indexes_2	(vector) indexes for second split, which can be generated via get_split_indexes_from_stratum

Value

(list) List with two elements that contain stratum split in two parts.

See Also

Other splitting functions: apply_split_indexes_to_strata(), check_strata(), get_split_indexes_from_strata(), get_split_indexes_from_stratum(), split_df(), split_strata(), split_stratum(), stratify()

Examples

```
# Random split-half. One of the splits gets 4 elements and the other 5
stratum =  letters[1:9]
indexes = get_split_indexes_from_stratum(stratum)
apply_split_indexes_to_stratum(stratum, indexes[[1]], indexes[[2]])
```

assmd

Calculate Absolute Strictly Standardized Mean Difference (ASSMD)

Description

Returns the absolute difference of the mean of x and y divided by their shared standard deviation. Since the resulting difference is absolute, the larger of the two means is always used as minuend and the smallest as subtrahend. Based on Zhang (2012) <doi: 10.1016/j.ygeno.2006.12.014>

Usage

```
assmd(x, y)
```

Arguments

x	(vector) a numeric vector
y	(vector) a numeric vector with compatible dimensions to x

Value

(numeric) Absolute SSMD

See Also

Other splithalfr coefficients: `angoff_feldt()`, `flanagan_rulon()`, `sdregi()`, `short_icc()`, `spearman_brown()`

Examples

```
# Generate two variables with different means, variances and a correlation of about 0.5
library(MASS)
vars = mvrnorm(30, mu = c(0, 2), Sigma = matrix(c(5, 2, 2, 3), ncol = 2), empirical = TRUE)
# Calculate Absolute SSMD
assmd(vars[,1], vars[,2])
```

---

by_split   *Calculate split scores per participant*

Description

Calculates split scores, by applying `fn_score` to subsets of `data` as specified via `participants`. It provides a range of additional arguments for different splitting methods and to support parallel processing. To learn more about writing scoring algorithms for use with the `splithalfr`, see the included vignettes. `by_split` is modeled after the `by` function, accepting similar values for the first three arguments (`data`, `INDICES`, `FUN`). For more information about different metods for splitting data, see `get_split_indexes_from_stratum`. For more information about stratification, see `split_df`

Usage

```
by_split(
  data,
  participants,
  fn_score,
  stratification = NULL,
  replications = 1,
  method = c("random", "odd_even", "first_second"),
  replace = FALSE,
```

```
  split_p = 0.5,
  subsample_p = 1,
  subsample_n = NULL,
  careful = TRUE,
  match_participants = FALSE,
  ncores = detectCores(),
  seed = NULL,
  verbose = TRUE
)
```

Arguments

- `data` — (data frame) data frame containing data to score. Data should be in long format, with one row per combination of participant and trial or item.
- `participants` — (vector) Vector that identifies participants in `data`.
- `fn_score` — (function) receives full or split sets, should return a single number.
- `stratification` — (vector). Vector that identifies which subsets of `data` should be split separately (denoted as strata in splitting functions) in order to ensure they are evenly distributed between parts. By default, the dataset of a participant formes a single stratum.
- `replications` — (numeric) Number of replications that split scores are calculated.
- `method` — (character) Splitting method. Note that `first_second` and `odd_even` splitting method will only deliver a valid split with default settings for other arguments (`split_p = 0.5,replace = FALSE,subsample_p = 1`)
- `replace` — (logical) If TRUE, stratum is sampled with replacement.
- `split_p` — (numeric) Desired length of both parts, expressed as a proportion of the length of the data per participant. If `split_p` is larger than 1 and `careful` is FALSE, then parts are automatically sampled with replacement
- `subsample_p` — (numeric) Subsample a proportion of `stratum` before splitting.
- `subsample_n` — (numeric) Subsample a number of participants before splitting.
- `careful` — (boolean) If TRUE, stop with an error when called with arguments that may yield unexpected splits
- `match_participants` — (logical) Default FALSE. If FALSE, the split-halves are newly randomized for each iteration and participant. If TRUE, the split-halves are newly randomized for each replication, but within a replication the same randomization is applied across participants. If the order of rows of datasets per participant denotes similar observations (such as items in a questionnaire), `match_participants` can be set to TRUE to ensure that per iteration, the same items are assigned to each part of the split-halves across participants. If `method` is "odd_even" or "first_second", splits are based on row number, so `match_participants` generally has little effects. If TRUE, each stratum should have the same number of rows, as checked via `check_strata`.
- `ncores` — (integer). By default, all available CPU cores are used. If 1, split replications are executed serially (via `lapply`). If greater than 1, split replications are executed in parallel, via (via `parLapply`).

seed (integer). When split replications are exectured in parallel, seed can be used to specificy a random seet to generate random seeds from for each worker via clusterSetRNGStream.

verbose (logical) If TRUE, reports progress. Note that progress across split replications is not displayed when these are executed in parallel.

Value

(data frame) Returns a data frame with a column for participant, a column replication that counts split replications, and score_1 and score_2 for the score calculated of each part via fn_score.

Examples

```
# N.B. This example uses R script from the vignette: "rapi_sum"
data("ds_rapi", package = "splithalfr")
# Convert to long format
ds_long <- reshape(
  ds_rapi,
  varying = paste("V", 1 : 23, sep = ""),
  v.names = "answer",
  direction = "long",
  idvar = "twnr",
  timevar = "item"
)
# Function for RAPI sum score
rapi_fn_score <- function (data) {
  return (sum(data$answer))
}
# Calculate scores on full data
by(
  ds_long,
  ds_long$twnr,
  rapi_fn_score
)
# Permutation split, one iteration, items matched across participants
split_scores <- by_split(
  ds_long,
  ds_long$twnr,
  rapi_fn_score,
  ncores = 1,
  match_participants = TRUE
)
# Mean flanagan-rulon coefficient across splits
fr <- mean(split_coefs(split_scores, flanagan_rulon))
```

---

check_strata *Check whether two strata have the same structure*

Description

Checks `strata` against `strata_left`. Each element of `strata_left` should also be present in `strata`, be of a similar type (data frame/tibble or list/vector), and be of similar size (`nrow` for data frames/tibbles or `length` for lists/vectors). Stops with an arror if any checks fail.

Usage

```
check_strata(strata_left, strata_right)
```

Arguments

`strata_left`	(list) strata to check against
`strata_right`	(list) strata to check

Value

None

See Also

Other splitting functions: `apply_split_indexes_to_strata()`, `apply_split_indexes_to_stratum()`, `get_split_indexes_from_strata()`, `get_split_indexes_from_stratum()`, `split_df()`, `split_strata()`, `split_stratum()`, `stratify()`

Examples

```
check_strata(list(1 : 4), list(1 : 4))
```

`ds_aat` *Example Approach Avoidance Task (AAT) Measurement Data in JASMIN2 Format*

Description

The JASMIN1 AAT was an irrelevant feature task, in which participants were instructed to approach/avoid left/right rotated stimuli. This particular AAT was administered (and described in detail) in doi: 10.1111/add.14071Boffo et al., 2018. Participants were presented stimuli from a "test" category, which were gambling-related pictures, and from a "control" category, which were pictures unrelated to gambling. It registered approach responses by participants pressing (and holding) the arrow down key, while avoid responses were given via the arrow up key. Upon a response, the stimulus zoomed in or out, until it disappeared from the screen. The first response to a stimulus was logged. The dataset contains one row per trial. This dataset was graciously provided by Eva Schmitz.

Usage

```
ds_aat
```

Format

An object of class `data.frame` with 6528 rows and 12 columns.

Details

Overview of columns:

- UserID. Identifies participants
- approach_tilt. If "left", participants were instructed to approach left rotated stimuli. If "right", participants were instructed to approach right rotated stimuli.
- block_type. Type of block: "practice" for practice trials with neutral stimuli, "assess" for assessment trials with salient stimuli
- block. Counts blocks, starting at zero
- trial. Counts trials in blocks, starting at zero
- appr. If "yes", this trial was an approach trial. If "no", this trial was an avoid trial.
- tilt. Whether the stimulus was rotated to the "left" or to the "right"
- cat. Stimulus category: "practice", "test", or "control"
- stim. Stimulus ID
- response. Response; 1 = correct, 2 = incorrect, 3 = timeout (no response in 4000 ms), 4 = invalid key
- rt. Response time in milliseconds
- sust. Was approach or avoid response sustained until the stimulus was completely zoomed in or out?

`ds_gng` *Example Go/No Go data*

Description

The Go/No Go is a task in which participants respond to one set of stimuli, but withhold a response to another set of stimuli. This particular dataset is from the first session of a study that is described in detail in doi: 10.1016/j.tate.2019.102887Hedge, Powell, and Sumner (2018). It was graciously provided by Craig Hedge and can be obtained from https://osf.io/cwzds.

Usage

```
ds_gng
```

Format

An object of class `data.frame` with 28200 rows and 7 columns.

Details

Overview of columns:

- block. Block number
- trial. Trial number
- stim. Stimuli set used in that block
- condition. 0 = go, 2 = no go
- response. Correct (1) or incorrect (0)
- rt. Reaction time (seconds)
- participant. Participant ID

---

ds_iat *Example Implicit Association Task (IAT) Data in JASMIN2 Format*

Description

The JASMIN2 IAT closely followed the original IAT procedure (Greenwald, McGhee, & Schwartz, 1998), except that target and attribute trials did not alternate. Upon a correct response, the next trial started. Upon an incorrect response, the current trial was repeated. The response to each trial was logged. This particular dataset is from a Ethnicity-Valence IAT, which was administered (and described in detail) in doi: 10.1016/j.tate.2019.102887Abacioglu and colleagues (2019). This dataset was graciously provided by Fadie Hanna and Marjolein Zee.

Usage

```
ds_iat
```

Format

An object of class `data.frame` with 9696 rows and 11 columns.

Details

Overview of columns:

- participation_id Identifies participants
- t1_left. If TRUE, the first combination block had target 1 on the left (and target 2 on the right)
- a1_left. If TRUE, the first combination block had attribute 1 on the left (and attribute 2 on the right)
- block_type. Type of block
- block. Counts blocks, starting at zero
- trial. Counts trials in blocks, starting at zero
- attempt. Counts attempts (responses) in trials, starting at zero

- cat. Category that stimulus belonged to
- stim. Stimulus
- response. Response; 1 = correct, 2 = incorrect, 3 = timeout (no response in 4000 ms), 4 = invalid key
- rt. Response time in milliseconds. Note that some response times may exceed the timeout window due to clock errors on the computer that the IAT was administered

The variable block_type can have these values:

- tar_discr: target discrimination
- att_discr: attribute discrimination
- tar1att1_1: target 1 with attitude 1, practice block
- tar1att1_2: target 1 with attitude 1, test block
- tar_rev: reverse target discrimination
- tar1att2_1: target 1 with attitude 2, practice block
- tar1att2_2: target 1 with attitude 2, test block

`ds_rapi` *Example 23-item Rutgers Alcohol Problem Inventory (RAPI) data*

Description

The RAPI is a questionnaire which asks how often a participant experienced each of 23 alcohol-related problems within the last year (doi: 10.15288/jsa.1989.50.30White & Labouvie, 1989). The dataset contains one row per participant.

Usage

```
ds_rapi
```

Format

An object of class `data.frame` with 426 rows and 24 columns.

Details

The dataset contains the following columns:

- twnr. Identifies participants
- V1 to V23. Answers on each of the 23 RAPI items

Each item is answered on a four-point scale with the following answer options:

- 0 = None
- 1 = 1-2 times
- 2 = 3-5
- 3 = More than 5 times

ds_sst *Example Stop Signal Task data*

Description

The Stop Signal Task is a task in which participants responded whether a stimulus was a square or a circle. On 25 This particular dataset is from the first session of a study that is described in detail in doi: 10.1016/j.tate.2019.102887Hedge, Powell, and Sumner (2018). It was graciously provided by Craig Hedge and can be obtained from https://osf.io/cwzds.

Usage

```
ds_sst
```

Format

An object of class `data.frame` with 27000 rows and 7 columns.

Details

Overview of columns:

- block. Block number
- trial. Trial number
- ssd. Stop signal delay
- condition. 0 = go, 1 = stop
- response. Correct (1) or incorrect (0)
- rt. Reaction time (milliseconds)
- participant. Participant ID

ds_vpt *Example Visual Probe Task (VPT) Measurement Data in JASMIN1 Format*

Description

The JASMIN1 VPT distinguished between "test" stimuli, which are in some way assumed to be salient to the participant and "control" stimuli, which are not. Test and control stimuli were presented in pairs, with one left and one right, followed by a probe that was an arrow pointing up or down. Participants needed to indicate whether the arrow pointed up or down. Upon a correct response the next trial started and upon an incorrect response the current trial was repeated. Only the first response to a new trial was logged. This particular VPT was part of the pre-measurement of a cognitive bias modification study. The "test" stimuli were alcoholic beverages and the "control" stimuli were non-alcoholic beverages, selected from the Amsterdam Beverage Picture Set doi: 10.1111/acer.12853(Pronk, Deursen, Beraha, Larsen, & Wiers, 2015). The dataset contains one row per trial. This dataset was graciously provided by Marilisa Boffo.

Usage

```
ds_vpt
```

Format

An object of class `data.frame` with 19520 rows and 12 columns.

Details

Overview of columns:

- UserID. Identifies participants
- patt. Probe-at-test. If "yes", the probe was positioned at the test stimulus. If "no", the probe was positioned at the control stimulus.
- phor. Probe horizontal position. Values: "left" or "right"
- thor. Test horizontal position. Values: "left" or "right"
- keep. If "yes" the probe was superimposed on the stimuli. If "no" the probe replaced the stimuli.
- pdir. Probe direction. Values: "up" or "down"
- stim. Stimulus
- response. Response; 1 = correct, 2 = incorrect, 3 and NA = timeout (no response in 5000 ms), 4 = invalid key
- rt. Response time in milliseconds
- block. Counts blocks, starting at zero
- trial. Counts trials in blocks, starting at zero
- block_type. Type of block: "assess" for assessment trials with salient stimuli

---

`flanagan_rulon` *Calculate Flanagon-Rulon coefficient*

Description

Flanagon-Rulon reliability coefficient. Formula obtained from Warrens (2015) <doi: 10.1007/s1163401501986>

Usage

```
flanagan_rulon(x, y)
```

Arguments

`x`	(vector) a numeric vector
`y`	(vector) a numeric vector with compatible dimensions to x

Value

(numeric) Flanagon-Rulon coefficient

See Also

Other splithalfr coefficients: angoff_feldt(), assmd(), sdregi(), short_icc(), spearman_brown()

Examples

```
# Generate two variables with different means, variances and a correlation of about 0.5
library(MASS)
vars = mvrnorm(30, mu = c(0, 2), Sigma = matrix(c(5, 2, 2, 3), ncol = 2), empirical = TRUE)
# Calculate coefficient
flanagan_rulon(vars[,1], vars[,2])
```

---

get_split_indexes_from_strata

Generate indexes for splitting strata

Description

Generates indexes for splitting each element of `strata` into two parts. For more information about splitting options, and an extensive list of examples, see get_split_indexes_from_stratum.

Usage

```
get_split_indexes_from_strata(strata, ...)
```

Arguments

`strata`	(list) Strata to split
`...`	Arguments passed on to get_split_indexes_from_stratum

- `method` (character) Splitting method. Note that `first_second` and `odd_even` splitting method will only deliver a valid split with default settings for other arguments (`subsample_p = 1`, `split_p = 1`, `replace = TRUE`)
- `replace` (logical) If FALSE, splits are constructed by sampling from stratum without replacement. If TRUE, stratum is sampled with replacement.
- `split_p` (numeric) Desired joint size of both parts, expressed as a proportion of the size of the subsampled `stratum`. If `split_p` is larger than 1, and `careful` is FALSE, then parts are automatically sampled with replacement
- `subsample_p` (numeric) Subsample a proportion of `stratum` to be used in the split.
- `careful` (boolean) If TRUE, stop with an error when called with arguments that may yield unexpected splits

Value

(list) A list with two elements, containing the first and second part of `strata`.

See Also

Other splitting functions: `apply_split_indexes_to_strata()`, `apply_split_indexes_to_stratum()`, `check_strata()`, `get_split_indexes_from_stratum()`, `split_df()`, `split_strata()`, `split_stratum()`, `stratify()`

Examples

```
# Stratify a data frame, then split it odd-even
ds <- data.frame(condition = rep(c("a", "b"), each = 4), score = 1 : 8)
strata <- stratify(ds, ds$condition)
split_indexes <- get_split_indexes_from_strata(strata, method = "odd_even")
apply_split_indexes_to_strata(strata, split_indexes)
```

`get_split_indexes_from_stratum`

Generate indexes that can be used to split a stratum into two parts

---

Description

`get_split_indexes_from_stratum` returns a list with indexes for splitting its `stratum` argument in two parts. The splits differ at most by one in size. With default arguments, a random split-half is returned, which samples elements for each part from `stratum` without replacement. Via additional arguments to `get_split_indexes_from_stratum` a range of other splitting methods can be applied.

Usage

```
get_split_indexes_from_stratum(
  stratum,
  method = c("random", "odd_even", "first_second"),
  replace = FALSE,
  split_p = 0.5,
  subsample_p = 1,
  careful = TRUE
)
```

Arguments

`stratum`	(data frame, tibble, list, or vector) Object to split; dataframes and tibbles are counted and split by row. All other data types are counted and split by element
`method`	(character) Splitting method. Note that `first_second` and `odd_even` splitting method will only deliver a valid split with default settings for other arguments (`subsample_p = 1, split_p = 1, replace = TRUE`)

`replace` (logical) If FALSE, splits are constructed by sampling from stratum without replacement. If TRUE, stratum is sampled with replacement.

`split_p` (numeric) Desired joint size of both parts, expressed as a proportion of the size of the subsampled `stratum`. If `split_p` is larger than 1, and `careful` is FALSE, then parts are automatically sampled with replacement

`subsample_p` (numeric) Subsample a proportion of `stratum` to be used in the split.

`careful` (boolean) If TRUE, stop with an error when called with arguments that may yield unexpected splits

Details

The following rounding rules apply to subsample size and split size:

- If the size of the subsample, calculated as `subsample_p` times size of `stratum`, is a fraction, then subsample size is rounded up.
- If the joint size of the two parts, calculated as 2 * `split_p` times size of the subsampled `stratum`, is a fraction, the part size is rounded up.
- If the joint size of the two parts is odd and `replace` is FALSE, then one of the parts randomly gets one more element than the other part.
- If the joint size of the two parts is odd and `replace` is TRUE, part size is rounded up to the next whole number, so each of the splits has the same size.

Value

(list) List with two elements that contain indexes that can be used to split the stratum in two parts two splits of stratum.

See Also

Other splitting functions: `apply_split_indexes_to_strata()`, `apply_split_indexes_to_stratum()`, `check_strata()`, `get_split_indexes_from_strata()`, `split_df()`, `split_strata()`, `split_stratum()`, `stratify()`

Examples

```
# Split-half. One of the splits gets 4 elements and the other 5
stratum = letters[1:9]
indexes = get_split_indexes_from_stratum(stratum)
apply_split_indexes_to_stratum(stratum, indexes[[1]], indexes[[2]])
```

sdregi — *SD ratio of equalities or greater inequalities*

Description

Returns the ratio of the SDs of x and y, using the largest SD of the two as denominator. Hence, the result is always 1 (ratio of equalities) or greater than 1 (ratio of greater inequalities). If x or y have less than two elements, NA is returned.

Usage

```
sdregi(x, y)
```

Arguments

`x`	(vector) a numeric vector
`y`	(vector) a numeric vector with compatible dimensions to x

Value

(numeric) SD ratio

See Also

Other splithalfr coefficients: `angoff_feldt()`, `assmd()`, `flanagan_rulon()`, `short_icc()`, `spearman_brown()`

Examples

```
# Generate two variables with different means, variances and a correlation of about 0.5
library(MASS)
vars = mvrnorm(30, mu = c(0, 2), Sigma = matrix(c(5, 2, 2, 3), ncol = 2), empirical = TRUE)
# Calculate SD ratio of left and right variables
sdregi(vars[,1], vars[,2])
# Calculate SD ratio of right and left variables; should give same result
sdregi(vars[,1], vars[,2])
```

short_icc — *Calculate Intraclass Correlation Coefficient (ICC)*

Description

Wrapper for ICCs calculated via `ICC`. If x or y have less than two elements, NA is returned.

Usage

```
short_icc(
  x,
  y,
  type = c("ICC1", "ICC2", "ICC3", "ICC1k", "ICC2k", "ICC3k"),
  ...
)
```

Arguments

`x`	(vector) a numeric vector
`y`	(vector) a numeric vector with compatible dimensions to x
`type`	(character) type of ICC to calculate, see `ICC`
`...`	Arguments passed to `ICC`

Value

(numeric) Value of ICC coefficient

See Also

Other splithalfr coefficients: `angoff_feldt()`, `assmd()`, `flanagan_rulon()`, `sdregi()`, `spearman_brown()`

Examples

```
# Generate two variables with different means, variances and a correlation of about 0.5
library(MASS)
vars = mvrnorm(30, mu = c(0, 2), Sigma = matrix(c(5, 2, 2, 3), ncol = 2), empirical = TRUE)
# Calculate ICC1
short_icc(vars[,1], vars[,2], type = "ICC1", lmer = FALSE)
```

`spearman_brown` *Calculate Spearman-brown coefficient*

Description

Spearman-Brown reliability coefficient for doubling test length. Formula obtained from Warrens (2015) <doi: 10.1007/s1163401501986>

Usage

```
spearman_brown(x, y, fn_cor = cor, ...)
```

Arguments

`x`	(vector) a numeric vector
`y`	(vector) a numeric vector with compatible dimensions to x
`fn_cor`	(function) a function returning a correlation coefficient
`...`	Arguments passed to `fn_cor`

Value

(numeric) Spearman-Brown coefficient

See Also

Other splithalfr coefficients: `angoff_feldt()`, `assmd()`, `flanagan_rulon()`, `sdregi()`, `short_icc()`

Examples

```
# Generate two variables with different means, variances and a correlation of about 0.5
library(MASS)
vars = mvrnorm(30, mu = c(0, 2), Sigma = matrix(c(5, 2, 2, 3), ncol = 2), empirical = TRUE)
# Calculate coefficient based on Pearson correlation
spearman_brown(vars[,1], vars[,2])
# Calculate coefficient based on ICC, two-way, random effects, absolute agreement, single rater
spearman_brown(vars[,1], vars[,2], short_icc, type = "ICC1", lmer = FALSE)
```

---

splithalfr — *splithalfr: Split-Half Reliabilities*

Description

Estimates split-half reliabilities for scoring algorithms of cognitive tasks and questionnaires.

Getting started

We've got six short vignettes to help you get started. You can open a vignette bij running the corresponding code snippets (`vignette(...)`) in the R console.

- `vignette("rapi_sum")` Sum-score for data of the 23-item version of the Rutgers Alcohol Problem Index (doi: 10.15288/jsa.1989.50.30White & Labouvie, 1989)
- `vignette("vpt_diff_of_means")` Difference of mean RTs for correct responses, after removing RTs below 200 ms and above 520 ms, on Visual Probe Task data (Mogg & Bradley, 1999 <doi: 10.1080/026999399379050>)
- `vignette("aat_double_diff_of_medians")` Double difference of medians for correct responses on Approach Avoidance Task data (Heuer, Rinck, & Becker, 2007 <doi: 10.1016/j.brat.2007.08.010>)
- `vignette("iat_dscore_ri")` Improved d-score algorithm for data of an Implicit Association Task that requires a correct response in order to continue to the next trial (Greenwald, Nosek, & Banaji, 2003)

- `vignette("sst_ssrti")` Stop-Signal Reaction Time integration method for data of a Stop Signal Task (Logan, 1981)
- `vignette("gng_dprime")` D-prime for data of a Go/No Go task (Miller, 1996 <doi: 10.3758/BF03205476>)

Splitting methods

The splithalfr supports a variety of methods for splitting your data. We review and assess each method in the compendium paper (Pronk et al., 2021 <doi: 10.3758/s13423021019483>). This vignette illustrates how to apply each splitting method via the splithalfr: `vignette("splitting_methods")`

- first-second and odd-even (Green et al., 2016 <doi: 10.3758/s1342301509683>; Webb, Shavelson, & Haertel, 1996 <doi: 10.1016/S01697161(06)260048>; Williams & Kaufmann, 2012 <doi: 10.1016/j.jesp.2012.03.001>)
- stratified (Green et al., 2016 <doi: 10.3758/s1342301509683>)
- permutated/bootstrapped/random sample of split halves (Kopp, Lange, & Steinke, 2021 <doi: 10.1177/1073191119866257>, Parsons, Kruijt, & Fox, 2019 <doi: 10.1177/2515245919879695>; Williams & Kaufmann, 2012 <doi: 10.1016/j.jesp.2012.03.001>)
- Monte Carlo (Williams & Kaufmann, 2012 <doi: 10.1016/j.jesp.2012.03.001>)

Validation of split-half estimations

Part of the splithalfr algorithm has been validated via a set of simulations that are not included in this package. The R script for these simulations can be found here.

Related packages

These R packages offer bootstrapped split-half reliabilities for specific scoring algorithms and are available via CRAN at the time of this writing: multicon, psych, and splithalf.

Acknowledgments

I would like to thank Craig Hedge, Eva Schmitz, Fadie Hanna, Helle Larsen, Marilisa Boffo, and Marjolein Zee for making datasets available for inclusion in the splithalfr. Additionally, I would like to thank Craig Hedge and Benedict Williams for sharing R-scripts with scoring algorithms that were adapted for splithalfr vignettes. Finally, I would like to thank Mae Nuys and Maren Sera for spotting bugs in earlier versions of this package.

`split_ci` *Calculate nonparametric bias-corrected and accelerated bootstrap confidence intervals for coefficients averaged across split replications*

Description

Calculates nonparametric bias-corrected and accelerated bootstrap confidence intervals via `bcajack`. Coefficients are `ds` should be a data frame as returned by `by_split`: Each unique value of the column `participant` is considered a independent sample of the target population. For each unique value of the column `split` in `ds`, it selects the corresponding rows in `ds`, and passes the values in the columns `score_1` and `score_2` as the first and second argument to `fn_coef`. Any row in `ds` for which `score_1` or `score_2` is NA is pairwise removed before passing the data to `fn_coef`. Any coefficient that is NA is removed before passing the data to to `fn_summary`.

Usage

```
split_ci(
  ds,
  fn_coef,
  fn_average = function(values) {     mean(values, na.rm = TRUE) },
  replications = 1000,
  ...
)
```

Arguments

`ds`	(data frame) a data frame with columns `split`, `score_1`, and `score_2`
`fn_coef`	(function) a function that calculates a bivariate coefficient.
`fn_average`	(function) a function that calculates an average across coefficients
`replications`	(integer) number of bootstrap replications
`...`	Additional arguments passed to `bcajack`

Details

For averaging internal consistency coefficients, see Feldt and Charter (2006). For more information about bias-corrected and accelerated bootstrap confidence intervals, see Efron (1987).

Value

Confidence interval

References

Efron, B. (1987). Better bootstrap confidence intervals. *Journal of the American statistical Association*, 82(397), 171-185. doi: 10.1080/01621459.1987.10478410

Feldt, L. S., & Charter, R. A. (2006). Averaging internal consistency reliability coefficients. *Educational and Psychological Measurement*, 66(2), 215-227. doi: 10.1177/0013164404273947

See Also

Other split aggregation functions: `split_coefs()`

Examples

```
# Generate five splits with scores that are correlated 0.00, 0.25, 0.5, 0.75, and 1.00
library(MASS)
ds_splits = data.frame(V1 = numeric(), V2 = numeric(), split = numeric())
for (r in 0:4) {
 vars = mvrnorm(10, mu = c(0, 0), Sigma = matrix(c(10, 3, 3, 2), ncol = 2), empirical = FALSE)
  ds_splits = rbind(ds_splits, cbind(vars, r, 1 : 10))
}
names(ds_splits) = c("score_1", "score_2", "replication", "participant")
# Calculate confidence interval
split_ci(ds_splits, cor)
```

---

split_coefs *Calculate a bivariate coefficient for each split-half replication*

Description

Calculates a bivariate coefficient across participants for each split-half replication and returns their values calculated across replications. `ds` should be a data frame as returned by `by_split`: For each unique value of the column `split` in `ds`, it selects the corresponding rows in `ds`, and passes the values in the columns `score_1` and `score_2` as the first and second argument to `fn_coef`. Any row in `ds` for which `score_1` or `score_2` is NA is pairwise removed before passing the data to `fn_coef`. For averaging internal consistency coefficients, see Feldt and Charter (2006).

Usage

```
split_coefs(ds, fn_coef, ...)
```

Arguments

`ds`	(data frame) a data frame with columns `split`, `score_1`, and `score_2`
`fn_coef`	(function) a function that calculates a bivariate coefficient.
`...`	Additional arguments passed to `fn_coef`

Value

Coefficients per split calculated via `fn_coef`.

References

Feldt, L. S., & Charter, R. A. (2006). Averaging internal consistency reliability coefficients. *Educational and Psychological Measurement*, 66(2), 215-227. doi: 10.1177/0013164404273947

See Also

Other split aggregation functions: `split_ci()`

Examples

```
# Generate five splits with scores that are correlated 0.00, 0.25, 0.5, 0.75, and 1.00
library(MASS)
ds_splits = data.frame(score_1 = numeric(), score_2 = numeric(), replication = numeric())
for (r in 0:4) {
 vars = mvrnorm(10, mu = c(0, 0), Sigma = matrix(c(10, 3, 3, 2), ncol = 2), empirical = FALSE)
  ds_splits = rbind(ds_splits, cbind(vars, r))
}
names(ds_splits) = c("score_1", "score_2", "replication")
# Pearson correlations
split_coefs(ds_splits, cor)
# Spearman-brown corrected Pearson correlations
split_coefs(ds_splits, spearman_brown)
# Flanagan-Rulon coefficient
split_coefs(ds_splits, flanagan_rulon)
# Angoff-Feldt coefficient
split_coefs(ds_splits, angoff_feldt)
# Spearman-Brown corrected ICCs
split_coefs(
  ds_splits,
  spearman_brown,
  short_icc,
  type = "ICC1",
  lmer = FALSE
)
```

---

split_df *Split a data frame into two parts*

Description

Splits data, Applies a stratified split to a data frame and returns each part. For more information about splitting options, and an extensive list of examples, see `get_split_indexes_from_stratum`.

Usage

```
split_df(data, stratification = NULL, ...)
```

Arguments

`data`	(data frame) Data to split, in long format, with one row per observation.
`stratification`	(vector). Vector that identifies which subsets of `data` should be split separately (denoted as strata in splitting functions) in order to ensure they are evenly distributed between parts. If NULL, all `data` is considered a single stratum.
`...`	Arguments passed on to `get_split_indexes_from_stratum`
	`method` (character) Splitting method. Note that `first_second` and `odd_even` splitting method will only deliver a valid split with default settings for other arguments (`subsample_p = 1, split_p = 1, replace = TRUE`)

`replace` (logical) If FALSE, splits are constructed by sampling from stratum without replacement. If TRUE, stratum is sampled with replacement.

`split_p` (numeric) Desired joint size of both parts, expressed as a proportion of the size of the subsampled `stratum`. If `split_p` is larger than 1, and `careful` is FALSE, then parts are automatically sampled with replacement

`subsample_p` (numeric) Subsample a proportion of `stratum` to be used in the split.

`careful` (boolean) If TRUE, stop with an error when called with arguments that may yield unexpected splits

Value

(list) List with two elements that each contain one of two parts.

See Also

Other splitting functions: `apply_split_indexes_to_strata()`, `apply_split_indexes_to_stratum()`, `check_strata()`, `get_split_indexes_from_strata()`, `get_split_indexes_from_stratum()`, `split_strata()`, `split_stratum()`, `stratify()`

Examples

```
ds <- data.frame(condition = rep(c("a", "b"), each = 4), score = 1 : 8)
split_df(ds, method = "random")
split_df(ds, method = "odd_even")
split_df(ds, method = "first_second")
split_df(ds, stratification = ds$condition, method = "random")
split_df(ds, stratification = ds$condition, method = "odd_even")
split_df(ds, stratification = ds$condition, method = "first_second")
ds <- data.frame(condition = rep(c("a", "b"), 4), score = 1 : 8)
split_df(ds, method = "random")
split_df(ds, method = "odd_even")
split_df(ds, method = "first_second")
split_df(ds, stratification = ds$condition, method = "random")
split_df(ds, stratification = ds$condition, method = "odd_even")
split_df(ds, stratification = ds$condition, method = "first_second")
```

`split_strata` *Split each stratum into two parts*

Description

Splits each element of `strata` into two parts. For more information about splitting options, and an extensive list of examples, see `get_split_indexes_from_stratum`.

Usage

```
split_strata(strata, ...)
```

Arguments

strata
: (list) list of strata to split

...
: Arguments passed on to get_split_indexes_from_stratum

  method (character) Splitting method. Note that first_second and odd_even splitting method will only deliver a valid split with default settings for other arguments (subsample_p = 1, split_p = 1, replace = TRUE)

  replace (logical) If FALSE, splits are constructed by sampling from stratum without replacement. If TRUE, stratum is sampled with replacement.

  split_p (numeric) Desired joint size of both parts, expressed as a proportion of the size of the subsampled stratum. If split_p is larger than 1, and careful is FALSE, then parts are automatically sampled with replacement

  subsample_p (numeric) Subsample a proportion of stratum to be used in the split.

  careful (boolean) If TRUE, stop with an error when called with arguments that may yield unexpected splits

Value

(list) A list with two elements, containing the first and second split of strata.

See Also

Other splitting functions: apply_split_indexes_to_strata(), apply_split_indexes_to_stratum(), check_strata(), get_split_indexes_from_strata(), get_split_indexes_from_stratum(), split_df(), split_stratum(), stratify()

Examples

```
# Stratify a data frame, then split it odd-even
ds <- data.frame(condition = rep(c("a", "b"), each = 4), score = 1 : 8)
strata <- stratify(ds, ds$condition)
split_strata(strata, method = "odd_even")
```

---

split_stratum *Split a stratum into two parts*

Description

Splits stratum into two parts. For more information about splitting options, and an extensive list of examples, see get_split_indexes_from_stratum.

Usage

```
split_stratum(stratum, ...)
```

Arguments

- `stratum` (data frame, tibble, list, or vector) Stratum to split; dataframes and tibbles are counted and split by row. All other data types are counted and split by element
- `...` Arguments passed on to `get_split_indexes_from_stratum`
  - `method` (character) Splitting method. Note that `first_second` and `odd_even` splitting method will only deliver a valid split with default settings for other arguments (`subsample_p = 1, split_p = 1, replace = TRUE`)
  - `replace` (logical) If FALSE, splits are constructed by sampling from stratum without replacement. If TRUE, stratum is sampled with replacement.
  - `split_p` (numeric) Desired joint size of both parts, expressed as a proportion of the size of the subsampled `stratum`. If `split_p` is larger than 1, and `careful` is FALSE, then parts are automatically sampled with replacement
  - `subsample_p` (numeric) Subsample a proportion of `stratum` to be used in the split.
  - `careful` (boolean) If TRUE, stop with an error when called with arguments that may yield unexpected splits

Value

(list) List with two elements that contain each of the two parts of stratum split in two.

See Also

Other splitting functions: `apply_split_indexes_to_strata()`, `apply_split_indexes_to_stratum()`, `check_strata()`, `get_split_indexes_from_strata()`, `get_split_indexes_from_stratum()`, `split_df()`, `split_strata()`, `stratify()`

Examples

```
# Split stratum odd-even
ds <- data.frame(condition = rep(c("a", "b"), each = 4), score = 1 : 8)
split_stratum(ds, method = "odd_even")
```

---

stratify *Stratify a data frame*

Description

Split a dataframe into strata formed by each a unique value of `stratification`.

Usage

```
stratify(ds, stratification = NULL)
```

Arguments

`ds`	(data frame) data to split into strata
`stratification`	(vector). Vector that identifies which subsets of `data` should be split separately (denoted as strata in splitting functions) in order to ensure they are evenly distributed between patrs. If NULL, all `data` is considered a single stratum.

Value

(list) List of strata

See Also

Other splitting functions: `apply_split_indexes_to_strata()`, `apply_split_indexes_to_stratum()`, `check_strata()`, `get_split_indexes_from_strata()`, `get_split_indexes_from_stratum()`, `split_df()`, `split_strata()`, `split_stratum()`

Examples

```
# Stratify a data frame, then split it odd-even
ds <- data.frame(condition = rep(c("a", "b"), each = 4), score = 1 : 8)
strata <- stratify(ds, ds$condition)
split_strata(strata, method = "odd_even")
```

Index