

Package ‘rtprep’

October 5, 2026

Title Screening, Trimming, and Aggregating Response Time Data

Version 0.1.0

Description A uniform interface to common response time preprocessing decisions that precede analysing aggregated response times or fitting an evidence accumulation model. Screening rules from different preprocessing routines -- absolute cutoffs, standard deviation and median absolute deviation criteria, recursive moving criteria, and model-based mixture flagging -- all return the same per-trial object, so that consequences of a preprocessing choice can be compared rather than assumed. The package also provides aggregation into EZ-diffusion summary statistics, diagnostics reporting what each rule removed and where rules disagree. Finally, generators for response time data with contaminants of known type are provided, so that a chosen pipeline can be tested against ground truth. Screening criteria follow Van Selst and Jolicoeur (1994) [<doi:10.1080/14640749408401131>](https://doi.org/10.1080/14640749408401131), the contaminant mixture Ratcliff and Tuerlinckx (2002) [<doi:10.3758/BF03196302>](https://doi.org/10.3758/BF03196302), and the EZ-diffusion equations Wagenmakers, van der Maas and Grasman (2007) [<doi:10.3758/BF03194023>](https://doi.org/10.3758/BF03194023).

License GPL (>= 2)

Encoding UTF-8

Depends R (>= 4.1.0)

Imports graphics, stats, utils

Suggests testthat (>= 3.0.0), bmm, trimr, rtdists, SimDesign, dplyr, tibble, ggplot2, knitr, rmarkdown, spelling, withr

Config/testthat/edition 3

URL <https://github.com/GidonFrischkorn/rtprep>,
<https://www.gfrischkorn.org/rtprep/>

BugReports <https://github.com/GidonFrischkorn/rtprep/issues>

Config/Needs/website rmarkdown, ggplot2, dplyr, tidyr

VignetteBuilder knitr

LazyData true

Language en-GB
RoxygenNote 7.3.3
NeedsCompilation no
Author Gidon T. Frischkorn [aut, cre, cph] (ORCID:
 <https://orcid.org/0000-0002-5055-9764>)
Maintainer Gidon T. Frischkorn <gfrischkorn@icloud.com>
Repository CRAN
Date/Publication 2026-10-05 16:30:02 UTC

Contents

adjust_accuracy	2
check_guessing	4
extending	6
ez_ddm	10
report_screening	11
rtprep-glossary	13
rtprep_report	14
rtprep_screen	15
rt_example	16
rt_keep	18
rt_screen	19
rt_summary	21
rules	24
rules_compose	31
rule_custom	33
rule_hierarchical	34
r_contaminated	36
screen_compare	38
screen_fits	40
Index	42

adjust_accuracy	<i>Correct accuracy counts for contamination</i>
-----------------	--

Description

Removes the estimated contaminant trials from the accuracy counts, on the assumption that contaminants respond correctly at guess_rate. A port of `bmm::adjust_ezdm_accuracy()`.

Usage

```
adjust_accuracy(n_upper, n_trials, contaminant_prop, guess_rate = 0.5)
```

Arguments

<code>n_upper</code>	Count of upper-boundary (correct) responses. Vectorised: the three count and proportion arguments recycle to a common length, one row per element, so the function takes the columns of a summary table directly.
<code>n_trials</code>	Total number of trials.
<code>contaminant_prop</code>	Estimated contaminant proportion, typically the <code>contaminant_prop</code> column of <code>rt_summary()</code> . NA or ≤ 0 returns that row's counts unchanged.
<code>guess_rate</code>	Accuracy assumed for a contaminant response, known from the design. 0.5 for a two-alternative task. One value for every row.

Details

Stochastic by design. How many trials were contaminants, and how many of those happened to be correct, are both binomial draws, so repeated calls differ. That is faithful to the uncertainty in a mixture estimate, which a point estimate would understate, and it matches `bmm`. There is no `set.seed()` anywhere in `rtprep`; reproducibility is the caller's.

Each row draws independently. For a single row the two draws are made in the same order as `bmm::adjust_ezdm_accuracy()`, so the two functions give the same answer from the same random seed.

Value

A `data.frame` with integer `n_upper_adj` and `n_trials_adj`, one row per input element. A row whose `n_upper` or `n_trials` is NA comes back NA.

See Also

`rt_summary()` for the counts and the proportion, `ez_ddm()` for what to do with them.

Examples

```
set.seed(42)
adjust_accuracy(n_upper = 80, n_trials = 100, contaminant_prop = 0.1)

# one row per cell of a summary table
cells <- data.frame(
  n_upper = c(80, 45), n_trials = c(100, 50), contaminant_prop = c(0.1, 0.2)
)
adjust_accuracy(cells$n_upper, cells$n_trials, cells$contaminant_prop)
```

check_guessing

*Test whether the fast trials a rule removed were really guesses***Description**

A screening rule tells you which trials it removed. It does not tell you whether it was right. This is one check that it was: if the fast trials a rule excluded really were guesses, their accuracy should be at chance.

A Beta-Binomial test with a Savage–Dickey Bayes factor, ported from `bmm::validate_fast_guesses()`.

Usage

```
check_guessing(
  keep,
  rt,
  response,
  threshold_type = c("quantile", "absolute"),
  rt_threshold = 0.25,
  chance = 0.5,
  prior_alpha = 1,
  prior_beta = 1,
  credible_mass = 0.95
)
```

Arguments

<code>keep</code>	Logical vector of keep decisions, typically the <code>.keep</code> column of <code>rt_screen()</code> . Note this is the <i>keep</i> flag, not a contaminant flag, which is <code>rtprep</code> 's convention throughout, so a screen's output drops straight in.
<code>rt</code>	Numeric vector of response times, in seconds.
<code>response</code>	Response coding of the same length, in any form <code>rt_screen()</code> accepts.
<code>threshold_type</code>	Whether <code>rt_threshold</code> is a quantile of the response time distribution or an absolute value in seconds.
<code>rt_threshold</code>	The cut defining "fast": a quantile in $(0, 1)$, or a positive number of seconds.
<code>chance</code>	Accuracy expected from a guess, known from the design.
<code>prior_alpha, prior_beta</code>	Beta prior on the proportion correct. The default 1, 1 is uniform.
<code>credible_mass</code>	Mass of the highest-density interval.

Details

The test looks only at trials that were **both excluded and fast**. Slow exclusions are a different claim: a slow contaminant is an attention lapse, not a guess, and there is no reason to expect chance accuracy from one.

This is the diagnostic counterpart of `rule_mixture(use_accuracy = TRUE)`: one checks accuracy after flagging, the other uses it during. The package's own tests found that the latter collapses on overlapping contamination and actively hurts when contaminants keep their accuracy, which makes this currently the safer of the two instruments.

When no trial is both excluded and fast, the row comes back with `n_tested = 0` and NA statistics rather than an error. In a simulation that cell is common, and informative: it means the rule removed nothing fast.

Value

A one-row `data.frame`, rather than `bmm`'s list, because in practice this goes into a results table:

`prop_upper` observed proportion correct among the tested trials.

`hdi_lower`, `hdi_upper` highest-density interval on that proportion.

`bf_01` Savage–Dickey Bayes factor for guessing against not.

`guess_in_hdi` whether chance falls inside the interval.

`bf_evidence` the Bayes factor on Jeffreys' scale.

`posterior_alpha`, `posterior_beta`, `n_tested`, `rt_threshold`, `threshold_type`, `credible_mass`, `mean_rt_tested` the inputs and intermediates, so a results table is self-describing.

References

Jeffreys, H. (1961). *Theory of Probability* (3rd ed.). Oxford University Press.

See Also

`rt_screen()` for the keep vector, `rule_ewma()` for a rule that uses accuracy to screen rather than to check.

Examples

```
set.seed(3)
rt <- c(runif(20, 0.15, 0.30), rgamma(80, 5, 10) + 0.2)
response <- c(rbinom(20, 1, 0.5), rbinom(80, 1, 0.85))
scr <- rt_screen(rt, rule = rule_cutoff(0.35, 3))

check_guessing(scr$.keep, rt, response)
```

Description

A rule is a parameter object; the engine applies it. Adding one takes a single call to `new_rule()` with `fun =`, the function that decides which trials to keep. The function travels on the rule object, so there is no S3 method to write and nothing to register, and `rt_screen()`, `rt_keep()`, `screen_fits()` and `screen_compare()` all pick the rule up unchanged.

Usage

```
new_rule(
  subclass,
  label,
  ...,
  fun = NULL,
  description = NULL,
  reason = "contaminant",
  needs_response = NULL,
  per_trial = character(),
  grouped = FALSE
)

apply_rule(rule, rt, response = NULL)

apply_rule_grouped(rule, rt, response = NULL, idx_by_group)
```

Arguments

<code>subclass</code>	A string naming the rule family. The object's class becomes <code>c("rtprep_rule_<subclass>", "rtprep_rule")</code> , with <code>"rtprep_rule_fun"</code> between them when the rule carries its own function.
<code>label</code>	A string identifying the rule and its settings, used as the <code>.rule</code> column and by <code>print()</code> . Conventionally <code>"family(setting, setting)"</code> .
<code>...</code>	Named parameters stored on the rule, passed to <code>fun</code> by name and available to an <code>apply_rule()</code> method as <code>rule\$name</code> .
<code>fun</code>	The screening function, or <code>NULL</code> to write an <code>apply_rule()</code> method instead. See "What the screening function receives" and "What the screening function must return".
<code>description</code>	A sentence saying what the rule does, shown by <code>print()</code> .
<code>reason</code>	A string naming what a dropped trial was dropped for, used for the <code>.reason</code> column when <code>fun</code> returns a bare vector. <code>rtprep</code> 's own rules use <code>"too_fast"</code> , <code>"too_slow"</code> and <code>"contaminant"</code> .
<code>needs_response</code> , <code>per_trial</code> , <code>grouped</code>	See "Declaring what a rule needs".

<code>rule</code>	A rule object, as built by <code>new_rule()</code> .
<code>rt</code>	Numeric vector of response times for one group, in seconds, with missing values already removed.
<code>response</code>	Numeric 0/1 vector of the same length, or NULL.
<code>idx_by_group</code>	A list of integer vectors, one per group, giving each group's positions in <code>rt</code> .

Details

`rule_custom()` does the same thing in one call, without a constructor, for a rule used once. A package shipping a family of rules can instead write a method for the `apply_rule()` generic; see "If you are writing a package".

Value

`new_rule()` returns an object of class `c("rtprep_rule_<subclass>", "rtprep_rule")`, with `"rtprep_rule_fun"` inserted before the last when `fun` is given. `apply_rule()` returns the three-element list described under "If you are writing a package".

What the screening function receives

The function declares what it needs by name, and the engine passes exactly that and nothing else. It can supply:

`rt` the response times of one group, in seconds, with missing values already removed and never empty. The function therefore does no validation of its own.

`response` accuracy for the same trials, already coerced to 0/1, or NULL. Declaring it makes it mandatory; see "Declaring what a rule needs".

`rule` the rule object itself, for a function that would rather read `rule$name` than take parameters one by one.

`idx_by_group` for a `grouped = TRUE` rule only: a list of integer vectors giving each group's positions in `rt`.

any parameter stored on the rule everything passed to `new_rule()` through `...`, by the name it was given there.

A formal the engine cannot supply is an error when the rule is built, rather than in the middle of a screen — unless it has a default, in which case the engine leaves it alone and the default applies. Declaring `...` means "and everything else": the function then receives the whole lot.

What the screening function must return

The answer is always in *keep* terms: TRUE, or a probability near 1, means the trial stays. That is the direction of the `.keep` and `.prob` columns and the opposite of how an exclusion criterion is usually phrased, so it is worth checking once on data whose answer you know.

There are three ways to say it, and a rule can start with the first and move to the third without anything else changing.

1. A **logical vector**, one value per trial, TRUE for a trial to keep. The simplest rule, and what most rules are. `.prob` becomes 0 or 1, and every dropped trial is labelled with the rule's reason.

2. A **numeric vector**, one value per trial, in $[0, 1]$: the probability that the trial came from the decision process. Use this when the rule is a model rather than a cutoff, as `rule_mixture()` is. `rt_screen()` turns it into a decision with its own policy and threshold, so the function does not decide and must not round.
3. A **list of prob, reason and fit**, the full contract under "If you are writing a package". Use it when different trials are dropped for different reasons, or when the rule estimates something per group — a criterion, a pair of bounds, whether a fit converged — that should become a column of `screen_fits()`. `reason` is then ignored: the function owns the `reason` column.

Whichever shape it returns, it returns one value per trial, in the order `rt` came in, and no NA. A rule that cannot evaluate a trial or a whole group, because it has too few trials, a spread of zero, or a fit that did not converge, returns TRUE, or 1, and removes nothing. Silence is not evidence of contamination.

Declaring what a rule needs

Three arguments change how the engine calls the rule.

`needs_response = TRUE` makes `response` mandatory: `rt_screen()` then errors when it is not supplied, instead of the function receiving NULL. Left unset, it is TRUE for a function that declares a `response` argument.

`per_trial` names the elements of `...` that hold one value per trial rather than a parameter. The engine checks their length against `rt` up front and subsets them to the group before dispatch, so the function sees only its own group's values. `rule_oracle()` uses this to carry ground truth.

`grouped = TRUE` marks a rule that has to see every group at once, one that pools information across participants, say. The engine then calls it once, with all groups, and hands it `idx_by_group`.

If you are writing a package

A package shipping a family of rules can write a method for the `apply_rule()` generic instead of carrying a function on the object. The two routes are equivalent; a registered `apply_rule.rtprep_rule_<subclass>()` method takes precedence, and a fun on the same rule is then ignored.

`apply_rule()` is called once per group, with `rt` already stripped of missing values and guaranteed non-empty, and `response` already coerced to 0/1 (or NULL). Methods therefore do no validation of their own. A method returns a list of three elements:

`prob` numeric, `length(rt)`, in $[0, 1]$: the probability that the trial came from the decision process.

Note the direction: this is $P(\text{valid})$, not $P(\text{contaminant})$. A deterministic rule returns 0 and 1.

`reason` character, `length(rt)`, NA wherever the trial is valid. `rtprep`'s own rules use "too_fast", "too_slow" and "contaminant"; a new rule may use any label.

`fit` a one-row `data.frame` of whatever the rule estimated for that group (bounds, criteria, convergence), or NULL if it estimated nothing. These become the extra columns of `screen_fits()`.

A rule that cannot be evaluated on a group returns `prob = rep(1, length(rt))` and removes nothing, for the same reason a screening function returns TRUE.

A `grouped = TRUE` rule gets a method for `apply_rule_grouped()` instead, called once with every group, and returns `fit` as a list of one-row data frames, one per group.

Saving a rule

The function travels with the rule, and a screen carries the rule that made it, so saving either saves the function *and the environment it was written in*. A function defined at the top level costs nothing. One defined inside another function drags whatever that function was holding into the file with it, which is how a rule ends up megabytes wide. Define screening functions at the top level, and pass what they need through

See Also

`rule_custom()` for a rule in one call, `rules` for the families `rtprep` ships.

Examples

```
# a rule that keeps the middle 90% of each group by quantile
rule_middle <- function(p = 0.05) {
  new_rule(
    "middle",
    label = paste0("middle(", p, ")"),
    p = p,
    description = "Keep the middle 90% of each group by quantile.",
    fun = function(rt, p) {
      b <- stats::quantile(rt, c(p, 1 - p), names = FALSE)
      rt >= b[1] & rt <= b[2]
    }
  )
}
rule_middle()
screen_fits(rt_example$rt, rule_middle(), .by = rt_example$id)

# a graded rule returns a probability and lets rt_screen() decide
shrinking <- rule_custom(
  "shrinking",
  function(rt) pmin(1, 0.2 / rt),
  description = "The slower the trial, the less likely it is a decision."
)
head(rt_screen(rt_example$rt, shrinking, .by = rt_example$id))

# the package way: a method, for a family with diagnostics to report
rule_bounded <- function(p = 0.05) {
  new_rule("bounded", label = paste0("bounded(", p, ")"), p = p)
}
apply_rule.rtprep_rule_bounded <- function(rule, rt, response = NULL) {
  b <- stats::quantile(rt, c(rule$p, 1 - rule$p), names = FALSE)
  list(
    prob = as.numeric(rt >= b[1] & rt <= b[2]),
    reason = ifelse(
      rt < b[1], "too_fast", ifelse(rt > b[2], "too_slow", NA)
    ),
    fit = data.frame(lower = b[1], upper = b[2])
  )
}
registerS3method(
```

```

    "apply_rule", "rtprep_rule_bounded", apply_rule.rtprep_rule_bounded
  )
  screen_fits(rt_example$rt, rule_bounded(), .by = rt_example$id)

```

 ez_ddm

Invert summary statistics into diffusion parameters

Description

The closed-form EZ-diffusion equations of Wagenmakers, van der Maas and Grasman (2007): mean response time, response time variance, and accuracy in; drift rate, boundary separation, and non-decision time out.

Exported so that the whole pipeline-to-parameters check runs with only `rtprep` installed: a reader can screen, aggregate, and estimate without reaching for a model-fitting package.

Usage

```
ez_ddm(mean_rt, var_rt, accuracy, n_trials, s = 1)
```

Arguments

<code>mean_rt, var_rt</code>	Mean and variance of the response times, in seconds. Wagenmakers et al. define these on <i>correct</i> responses; version = "3par" of <code>rt_summary()</code> pools both boundaries, which is equivalent under an unbiased diffusion and not otherwise. Use version = "4par" if the starting point may be off centre.
<code>accuracy</code>	Proportion of upper-boundary (correct) responses, in $[0, 1]$.
<code>n_trials</code>	Number of trials the statistics came from. Required: it sets the size of the edge correction.
<code>s</code>	Scaling constant. 1 here; Wagenmakers et al. use 0.1. This is a units convention, not a modelling one: drift and bound scale linearly with <code>s</code> and <code>ndt</code> does not, so a drift of 0.1 at <code>s = 0.1</code> and a drift of 1.0 at <code>s = 1</code> describe the same process.

Details

The equations divide by `logit(accuracy)` and break down at accuracies of 0, 0.5, and 1. Wagenmakers et al.'s edge correction moves the offending value by $1 / (2 * n_trials)$: 1 becomes $1 - 1/(2n)$, 0 becomes $1/(2n)$, and 0.5 becomes $0.5 + 1/(2n)$. It is applied silently, because it is the published behaviour and a warning per cell would bury a simulation run. Which cells were corrected comes back in the `edge_corrected` column, so a script can count them. It is a column rather than an attribute so that it survives `[, rbind()]`, and the `dplyr` verbs.

EZ is fragile under contamination: a handful of fast guesses moves the drift estimate a long way (Ratcliff, 2008). That fragility is the point of the comparison this package exists to support, not a reason to avoid the estimator.

Value

A data.frame with drift, bound, ndt, and a logical edge_corrected, one row per input element (inputs recycle to a common length). edge_corrected flags the rows that needed the correction below.

References

Wagenmakers, E.-J., van der Maas, H. L. J., & Grasman, R. P. P. P. (2007). An EZ-diffusion model for response time and accuracy. *Psychonomic Bulletin & Review*, 14(1), 3–22. doi:10.3758/bf03194023

Ratcliff, R. (2008). The EZ diffusion method: Too EZ? *Psychonomic Bulletin & Review*, 15(6), 1218–1228. doi:10.3758/pbr.15.6.1218

See Also

`rt_summary()`, which produces exactly the inputs this takes.

Examples

```
# the worked example from Wagenmakers et al. (2007)
ez_ddm(
  mean_rt = 0.723, var_rt = 0.112, accuracy = 0.802,
  n_trials = 100, s = 0.1
)
```

report_screening

Report a screening procedure in prose

Description

Turns a screen into a paragraph that can go into a Methods section: which rule at which setting, the grouping the criterion was computed within, how much it removed in total and per cell, what it caught, whether it read accuracy, the keep policy, and the references for the criteria used.

Every number and every description is derived from the screen, so editing the rule and forgetting to edit the paragraph is not a way to publish a wrong Methods section.

Usage

```
report_screening(x, ...)

## S3 method for class 'rtprep_screen'
report_screening(x, rule = NULL, groups = NULL, max_words = 300L, ...)

## S3 method for class 'data.frame'
report_screening(
  x,
```

```

    rule,
    groups = NULL,
    policy = c("threshold", "probabilistic"),
    threshold = 0.5,
    max_words = 300L,
    ...
  )

```

Arguments

<code>x</code>	A screening result from <code>rt_screen()</code> , or a data frame carrying its four columns.
<code>...</code>	Ignored.
<code>rule</code>	The rule that produced the screen. Optional for an <code>rt_screen()</code> result, which carries it; required for a data frame, where it cannot be recovered.
<code>groups</code>	Character vector naming the grouping variables, for the sentence that says what the criterion was computed within. Optional for an <code>rt_screen()</code> result when <code>.by</code> was a data frame or a named list, which supplies them. For a data frame these must be columns of <code>x</code> , since their values are needed to rebuild the cells.
<code>max_words</code>	A guard on the length of the paragraph. Optional clauses are dropped, lowest priority first, rather than any sentence being truncated.
<code>policy, threshold</code>	The exclusion policy the screen was run under. Read from the object where it carries them; pass them for a data frame if they were not left at the defaults.

Details

The counts separate what the rule excluded from what was never screened. Trials with a missing response time, a missing grouping key, or (for a rule that reads accuracy) a missing response, come back with `.keep = FALSE` and `.reason = "missing"`, so `sum(!.keep)` overstates the rule's work. The percentages quoted for the rule are out of the trials it actually saw, and the missing trials get their own sentence.

Value

An object of class `rtprep_report`: a list whose text element is the paragraph, carrying alongside it every number the paragraph quotes (`n_trials`, `n_missing`, `n_screened`, `n_excluded`, `prop_excluded`, `reasons`, `cells`, `words`), the fits table it read them from, the rule, the reference keys and their bibliography, and any notes.

Which object to pass

`rt_screen()` attaches the rule and the grouping to its result, so `report_screening(scr)` needs nothing else. Inside `dplyr::mutate(rt_screen(rt, rule), .by = ...)` the four columns are spliced into the data frame and the attributes are lost, so the data frame method asks for the rule back and for groups, the names of the grouping columns. It recomputes the per-cell counts from `.keep` and those columns, which needs no refitting and gives the same answer.

What it does not claim

The reference list names the sources for the criteria used and the standard caveats on them, which are not the same thing: `rule_sd()` cites Miller (1991) on sample-size bias, and `rule_mad()` also cites Leys et al. (2013), whose argument is that the mean and standard deviation are the wrong choice. Read the sentence as a pointer to the literature, not as an endorsement. `toBibtex()` on the result gives the entries.

See Also

`rt_screen()` for the screen, `screen_fits()` for the per-cell table the counts come from, and `rt-prep-report` for the print and BibTeX methods.

Examples

```
scr <- rt_screen(rt_example$rt, rule_mad(2.5), .by = rt_example$id)
report_screening(scr, groups = "participant")

# every number in the paragraph, for a sentence written by hand
rep <- report_screening(scr)
rep$n_excluded
rep$cells
```

rtprep-glossary

Terms used in rtprep

Description

The vocabulary the rest of the documentation assumes, defined once.

Response times and what contaminates them

Contaminant A trial not produced by the decision process the experiment is about: a response made before the stimulus was read, one delayed by something outside the task, one made without using the evidence. The word is used for the *trial*, not for the statistical criterion that might catch it; "outlier" is kept for the criterion and for literature that uses it that way. `rtprep` names three processes, and `r_contaminated()` generates each: **leading-edge anticipations**, **delayed start-ups**, and **informationless responses**.

Leading edge The fast rising flank of a response time distribution, the short climb from the fastest response to the mode. It matters because a right-skewed distribution has almost no mass there, so a trial that arrives early sits well inside a criterion built around the mean and is not removed by one.

What the package does to them

Screening Classifying trials, deciding which came from the decision process. `rt_screen()` screens.

Trimming Screening that then removes what it flagged. Every trimming rule screens; not every screening rule trims, since a probability can be carried forward as a weight instead.

Aggregation Turning the surviving trials into summary statistics. `rt_summary()` aggregates. Kept separate from the two above because a screen and a summary fail in different ways, and the choice between them is a real one.

The model the summaries feed

Evidence accumulation model (EAM) A model of choice and response time in which a decision is made by gathering evidence over time until enough has arrived to commit. The **diffusion model (DDM)** is one member, a racing accumulator another.

Drift How fast evidence arrives, on average; the rate of the accumulation. Higher drift means faster and more accurate responding.

Bound How much evidence is required before committing, also called boundary separation. A wider bound means slower and more accurate responding, which is where speed-accuracy trade-offs live.

Non-decision time (ndt) Everything in a response time that is not evidence accumulation: encoding the stimulus at one end, executing the movement at the other.

EZ-diffusion A closed-form inversion from the mean response time, its variance, and accuracy to drift, bound and ndt, so no fitting is needed. `rt_summary()` produces the three inputs and `ez_ddm()` does the inversion.

See Also

[rtprep-package](#) for what the package is for.

rtprep_report

Methods for a screening report

Description

`report_screening()` returns the paragraph together with every number in it. `print()` wraps the paragraph to the console width and says how long it is; `as.character()` returns it unwrapped, for pasting into a document or an inline r chunk.

Usage

```
## S3 method for class 'rtprep_report'
format(x, width = 0.9 * getOption("width"), ...)

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

```
## S3 method for class 'rtprep_report'
as.character(x, ...)

## S3 method for class 'rtprep_report'
toBibtex(object, ...)
```

Arguments

x, object	A report from <code>report_screening()</code> .
width	Wrapping width for <code>print()</code> .
...	Ignored.

Value

`format()` returns the wrapped paragraph as a character vector, `print()` returns x invisibly, `as.character()` returns the paragraph as a single string, and `toBibtex()` returns the BibTeX entries for the references cited.

Examples

```
scr <- rt_screen(rt_example$rt, rule_mad(2.5), .by = rt_example$id)
rep <- report_screening(scr, groups = "participant")
rep

cat(as.character(rep))
utils::toBibtex(rep)
```

rtprep_screen	<i>Methods for a screening result</i>
---------------	---------------------------------------

Description

`rt_screen()` returns one row per trial with the per-group diagnostics attached as an attribute. `print()` reports what was removed and why before the rows, because the count is usually the answer wanted and the rows are usually too many to read.

Usage

```
## S3 method for class 'rtprep_screen'
format(x, ...)

## S3 method for class 'rtprep_screen'
print(x, n = 6L, ...)

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

```
## S3 method for class 'rtprep_screen'
x[i, j, drop = TRUE]
```

Arguments

x	A screening result from <code>rt_screen()</code> .
...	Ignored.
n	Number of trials to show. <code>print()</code> shows the screen summary in full whatever this is.
row.names, optional	Passed to <code>base::as.data.frame()</code> .
i, j	Row and column indices.
drop	Whether to drop to a vector when one column is selected.

Details

The `fits` attribute describes the screen, so it survives column subsetting and is dropped by row subsetting: after `scr[scr$.keep, ]` its `n_dropped` and `prop_dropped` count rows that are no longer there, and carrying it along would attach a table that quietly disagrees with the object. The class survives for as long as the four columns do.

The `rule` and `group_names` attributes, which `report_screening()` reads, name the screen rather than count it, so they survive row subsetting too.

`as.data.frame()` strips the class and the attributes, for when a plain frame is wanted.

Value

`print()` returns `x` invisibly. `format()` returns the summary as a character vector. `as.data.frame()` returns a plain `data.frame`.

Examples

```
scr <- rt_screen(rt_example$rt, rule_mad(2.5), .by = rt_example$id)
scr

# the per-group diagnostics the summary points at
head(attr(scr, "fits"))
```

rt_example

Simulated response times from four participants, with ground truth

Description

A small data set for the examples and the vignette: four participants, two conditions, 100 trials per cell, generated by `r_contaminated()` from a diffusion core with contaminants from all three processes. Because the data are simulated, every trial carries the truth a real data set cannot: whether it was a contaminant, and which process produced it.

Usage

```
rt_example
```

Format

A data.frame with 800 rows and 9 columns:

id participant, "p1" to "p4".

condition factor, "hard" or "easy"; the base drifts the two conditions are built from differ by 0.4, before the participant's ability factor multiplies both (see Details).

trial trial number within the cell, 1 to 100.

rt response time in seconds.

response 1 for a correct response, 0 for an error.

contaminant logical ground truth.

process "clean", "leading_edge", "delay", or "informationless".

true_drift the drift the cell was generated from.

contam_rate the participant's contamination rate: .02, .05, .10, and .15 for p1 to p4.

Details

The participants differ in two things at once. Their drift differs, by a fixed ability factor of 0.85, 0.95, 1.05, and 1.20 on a base drift of 1.5 (hard) and 1.9 (easy), with bound = 1.2 and ndt = 0.30 throughout. And their contamination rate differs, so that the error in a participant's estimate can be set against their own rate, as the get-started vignette does.

The generating script is data-raw/rt_example.R in the source repository; the seed is fixed there, not in the package.

See Also

[r_contaminated\(\)](#) to generate data shaped like your own task.

Examples

```
head(rt_example)
table(rt_example$id, rt_example$process)
```

rt_keep	<i>Keep vector from a screening rule</i>
---------	--

Description

The `.keep` column of `rt_screen()` on its own, for the case where a filter is all that is wanted. It reports how many trials it dropped, once per call, so that the exclusion count is logged next to the exclusion rather than reconstructed afterwards.

Usage

```
rt_keep(
  rt,
  rule,
  response = NULL,
  .by = NULL,
  policy = c("threshold", "probabilistic"),
  threshold = 0.5,
  quiet = FALSE
)
```

Arguments

rt	Numeric vector of response times in seconds . NA is allowed; non-positive values are an error.
rule	A rule object; see rules .
response	Optional response coding of the same length as <code>rt</code> , given as numeric 0/1, logical, or a character or factor using labels such as "correct"/"error" or "upper"/"lower". Required by <code>rule_ewma()</code> and by <code>rule_mixture()</code> with <code>use_accuracy = TRUE</code> .
.by	Optional grouping of the same length as <code>rt</code> : a vector, factor, list of vectors, or data frame. Rules are fitted separately within each group. NULL treats all trials as one group.
policy	Exclusion policy. "threshold" keeps a trial when its probability of validity exceeds threshold. "probabilistic" keeps it with that probability, drawing once per trial.
threshold	Cut for <code>policy = "threshold"</code> , in $[0, 1]$. Ignored under the probabilistic policy.
quiet	If FALSE (the default), one message states the rule, the number of trials dropped, and the proportion. TRUE suppresses it.

Details

Inside a grouped `filter()` the message fires once per group, because the function is called once per group. Pass `.by` to `rt_keep()` instead of to `filter()`: the keep vector is identical either way, and the count then covers the whole data set in one line. Or set `quiet = TRUE`.

Value

A logical vector the length of `rt`: TRUE for a trial to keep. Trials with a missing response time or grouping key are FALSE, as in `rt_screen()`.

See Also

`rt_screen()` for the probability and the reason alongside the decision; `screen_fits()` for the per-group diagnostics.

Examples

```
rt <- c(0.12, 0.31, 0.35, 0.38, 0.42, 0.47, 0.55, 2.90)
keep <- rt_keep(rt, rule_cutoff(0.18, 2.5))
rt[keep]

# with dplyr: dat |> filter(rt_keep(rt, rule_sd(2.5), .by = id))
```

rt_screen

Screen response times with any rule

Description

Applies a screening rule and returns one row per input trial: the keep decision, the probability behind it, the rule's label, and the reason for a flag. The four columns are the same whichever rule went in, so changing the rule changes one word and nothing around it.

Usage

```
rt_screen(
  rt,
  rule,
  response = NULL,
  .by = NULL,
  policy = c("threshold", "probabilistic"),
  threshold = 0.5
)
```

Arguments

<code>rt</code>	Numeric vector of response times in seconds . NA is allowed; non-positive values are an error.
<code>rule</code>	A rule object; see rules .
<code>response</code>	Optional response coding of the same length as <code>rt</code> , given as numeric 0/1, logical, or a character or factor using labels such as "correct"/"error" or "upper"/"lower". Required by <code>rule_ewma()</code> and by <code>rule_mixture()</code> with <code>use_accuracy = TRUE</code> .

<code>.by</code>	Optional grouping of the same length as <code>rt</code> : a vector, factor, list of vectors, or data frame. Rules are fitted separately within each group. <code>NULL</code> treats all trials as one group.
<code>policy</code>	Exclusion policy. <code>"threshold"</code> keeps a trial when its probability of validity exceeds threshold. <code>"probabilistic"</code> keeps it with that probability, drawing once per trial.
<code>threshold</code>	Cut for <code>policy = "threshold"</code> , in <code>[0, 1]</code> . Ignored under the probabilistic policy.

Details

Separating the probability from the decision is deliberate. A mixture rule produces a genuine posterior probability; a cutoff produces a degenerate one. Keeping both in the same object means a probabilistic screen can feed `rt_summary()` as a weight vector without being forced through a threshold first, and that the same comparison machinery covers both families.

Trials with a missing response time, or a missing value in any `.by` component, are excluded from every rule's computation and returned with `.keep = FALSE`, `.prob = NA`, and `.reason = "missing"`.

Under `policy = "probabilistic"` the decision is stochastic by design. There is no `set.seed()` anywhere in `rtprep`; reproducibility is the caller's.

Value

A `data.frame` with one row per element of `rt`, in input order:

- `.keep` logical; keep this trial under the stated policy.
- `.prob` numeric; the probability that the trial came from the decision process, that is **P(valid)**. Deterministic rules return 0 or 1. Note that `bmm::flag_contaminant_rts()` returns the complement.
- `.rule` character; the rule's label.
- `.reason` character; why the **rule** flagged the trial: `"too_fast"`, `"too_slow"`, `"contaminant"`, or `"missing"`. `NA` whenever the rule did not flag it, which includes trials the keep policy dropped anyway (at `threshold = 1`, or on a probabilistic draw against a fractional `.prob`). A kept trial never carries a reason.

Per-group fit diagnostics are attached as `attr(x, "fits")`: one row per group with `.group`, `n_trials`, `n_dropped`, and `prop_dropped`, plus whatever the rule reports (bounds, iterations, EM convergence). `screen_fits()` returns that table on its own.

Inside a data-frame pipeline

The function takes vectors and returns a data frame, so it drops into `dplyr::mutate()` unchanged: `mutate(rt_screen(rt, rule_sd(2.5)), .by = id)` splices the four columns in, and `filter(.keep)` then drops the flagged trials. Two things to know. `attr(x, "fits")` does not survive `mutate()`; use `screen_fits()` when the per-group table is what you want. And `dplyr` reads `.keep` = as its own argument, so assign the column by splicing rather than by name. When only the filter is needed, `rt_keep()` returns the logical vector directly.

See Also

[rules](#) for the rules themselves; [rt_keep\(\)](#) for the keep vector alone; [screen_fits\(\)](#) for the per-group table alone; [screen_compare\(\)](#) to apply several rules at once; [rt_summary\(\)](#) to aggregate what survives.

Examples

```
rt <- c(0.12, 0.31, 0.35, 0.38, 0.42, 0.47, 0.55, 2.90)
rt_screen(rt, rule_cutoff(0.18, 2.5))

# rules are group-aware without the package depending on dplyr
id <- rep(c("a", "b"), each = 4)
scr <- rt_screen(rt, rule_sd(2), .by = id)
attr(scr, "fits")

# a rule that reads accuracy takes it by name
correct <- c(0, 1, 1, 1, 0, 1, 1, 1)
rt_screen(rt, rule_ewma(lambda = 0.1), response = correct)
```

rt_summary

Aggregate response times into EZ-diffusion summary statistics

Description

Screening decides which trials survive; aggregation decides what the survivors are summarised *as*. They fail differently, so rtprep keeps them apart and makes both comparable: the same trials summarised three ways can give three different parameter estimates, and that is a preprocessing choice as consequential as the exclusion rule.

Usage

```
rt_summary(
  rt,
  response = NULL,
  method = c("simple", "robust", "trimmed", "winsorized", "mixture"),
  version = c("3par", "4par"),
  distribution = c("exgaussian", "lognormal", "invgaussian"),
  robust_scale = c("iqr", "mad"),
  trim = 0.1,
  weights = NULL,
  min_trials = 10,
  ...
)
```

Arguments

rt	Numeric vector of response times in seconds .
response	Optional response coding of the same length as rt, in any form rt_screen() accepts. Required for version = "4par"; without it n_upper is NA. A trial whose response is missing belongs to neither boundary and is left out of both, but still counts towards n_trials, as in bmm, so n_upper / n_trials understates accuracy when responses are missing. Drop those trials first if that matters.
method	How the moments are computed. • "simple" gives the sample mean and variance. The baseline, and what almost everyone does. • "robust" gives the median, with the variance from the interquartile range (divided by 1.349) or the median absolute deviation. Statistic-level robustness, after Chávez De la Peña et al. (2026). • "trimmed" gives the mean of the middle $1 - 2 * \text{trim}$ of the trials, with the variance from the Winsorized sample. The robust-statistics answer. • "winsorized" gives the same variance, with the mean of the Winsorized sample rather than the trimmed one. • "mixture" gives the analytic moments of the response time component of a fitted contaminant mixture, and its estimated contaminant proportion.
version	"3par" pools the two response boundaries; "4par" summarises each separately.
distribution	Core distribution for method = "mixture"; see rule_mixture() .
robust_scale	Spread statistic for method = "robust".
trim	Proportion of trials cut from <i>each</i> tail by method = "trimmed" and method = "winsorized", as in base::mean() . $\text{floor}(n * \text{trim})$ trials go from each end.
weights	Optional per-trial weights, typically the .prob column of rt_screen() . Defined for method = "simple" only.
min_trials	Below this many trials the moments are NA rather than noise; n_trials is still reported. With weights, the comparison uses Kish's effective sample size $\text{sum}(w)^2 / \text{sum}(w^2)$, so two unit weights among ninety-eight zeros count as two trials rather than a hundred.
...	Passed to the mixture fit: bound, init, max_prop, maxit, tol, as documented in rule_mixture() and with the same defaults, including maxit = 500 where bmm::ezdm_summary_stats() uses 100.

Value

A one-row data.frame holding the inputs [ez_ddm\(\)](#) needs.

version = "3par": mean_rt, var_rt, n_upper, n_trials, contaminant_prop.

version = "4par": mean_rt_upper, mean_rt_lower, var_rt_upper, var_rt_lower, n_upper, n_trials, contaminant_prop_upper, contaminant_prop_lower.

contaminant_prop is NA for every method but "mixture", which is the only one that estimates it.

Two ways of not letting contaminants count

method = "mixture" reads its moments from the *parametric component*, not from the data, so a trial the uniform component owns contributes nothing at all. weights instead computes weighted sample moments, so such a trial contributes a little. They are two models of the same doubt, which is why both are here and why combining them is an error rather than a convenience.

Weighted variances use the reliability-weight denominator $\text{sum}(w) - \text{sum}(w^2) / \text{sum}(w)$, which reduces to $n - 1$ when the weights are equal. Frequency weights would use $\text{sum}(w) - 1$ and are the wrong model: .prob is a probability, not a count.

Trimming and Winsorizing

Both cut the same $\text{floor}(n * \text{trim})$ trials from each tail. Trimming drops them; Winsorizing replaces each with the nearest surviving value, so the count stays the same and the extremes stop pulling. The variance comes from the Winsorized sample either way, rescaled so that it estimates the variance of the response time distribution rather than the variance of the trimmed mean. It is the same kind of correction as the 1.349 that robust_scale = "iqr" applies, and it is necessary because ez_ddm() reads var_rt as a moment of the distribution.

The divisor is not the familiar $(1 - 2 * \text{trim})^2$ of Tukey and McLaughlin (1963). That one estimates n times the variance of the trimmed *mean*, and using it here would report a variance 6% high at trim = 0.1 and 14% high at trim = 0.2, which ez_ddm() would read as a slower drift.

contaminant_prop stays NA. trim is the proportion removed, not an estimate of the proportion contaminated, and adjust_accuracy() would apply a second correction to counts that have already been trimmed.

Under version = "4par" the trim applies within each boundary, so the surviving count per boundary is about $(1 - 2 * \text{trim})$ of what arrived; set min_trials with that in mind.

Differences from bmm

bmm: ezdm_summary_stats() defaults to method = "mixture"; this function defaults to "simple". A package about preprocessing choices should not make one of the choices silently.

For version = "4par" with method = "mixture", the contaminant bounds are resolved once on the pooled response times before the split. Resolving them per boundary would make the uniform component's density depend on which side happened to have the wider range, so the two halves would be fitted against different models.

When the mixture fit fails the moments fall back to "robust" with a warning, as in bmm. rt_screen() resolves the analogous failure the other way and keeps every trial (see extending). The two layers differ because a screen can decline to act, and a summary still has to return a number.

References

- Wagenmakers, E.-J., van der Maas, H. L. J., Dolan, C. V., & Grasman, R. P. P. (2008). EZ does it! Extensions of the EZ-diffusion model. *Psychonomic Bulletin & Review*, 15(6), 1229–1235. doi:10.3758/pbr.15.6.1229
- Chávez De La Peña, A. F., Shin, E., & Vandekerckhove, J. (2026). Robust Bayesian hypothesis testing with the hierarchical EZ-DDM. *Behavior Research Methods*, 58(7), 177. doi:10.3758/s13428026030661

See Also

[ez_ddm\(\)](#) to invert these statistics into parameters, [adjust_accuracy\(\)](#) to correct the counts for contamination, [rt_screen\(\)](#) to decide which trials get here.

Examples

```
rt <- c(0.32, 0.35, 0.38, 0.41, 0.44, 0.47, 0.50, 0.55, 0.62, 0.71, 1.90)
correct <- c(1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0)

rt_summary(rt, correct, min_trials = 5)
rt_summary(rt, correct, method = "robust", min_trials = 5)

# a probabilistic screen can feed aggregation without a threshold
scr <- rt_screen(rt, rule = rule_sd(2))
rt_summary(rt, correct, weights = scr$.prob, min_trials = 5)
```

rules

Screening rules

Description

Rule constructors are parameter objects: they validate their arguments and describe themselves, but perform no computation on data. Pass one to [rt_screen\(\)](#), which applies it and returns the same four per-trial columns whichever rule was used.

Rules from incompatible families therefore become directly comparable:

- [rule_cutoff\(\)](#) applies fixed absolute bounds.
- [rule_sd\(\)](#) applies a location/spread criterion, covering both the SD criterion and the median absolute deviation criterion.
- [rule_mad\(\)](#) is a thin alias of [rule_sd\(center = "median", scale = "mad"\)](#).
- [rule_iqr\(\)](#) applies Tukey's quartile fences, asymmetric by construction.
- [rule_recursive\(\)](#) applies the sample-size-dependent criteria of van Selst and Jolicoeur (1994).
- [rule_ewma\(\)](#) applies the accuracy control chart of Vandekerckhove and Tuerlinckx (2007).
- [rule_mixture\(\)](#) fits a uniform-contaminant mixture by EM.
- [rule_none\(\)](#) is a pass-through baseline.
- [rule_oracle\(\)](#) is perfect exclusion, for generated data with known truth.

Usage

```
rule_cutoff(min = 0, max = Inf)

rule_sd(n_sd = 2.5, center = c("mean", "median"), scale = c("sd", "mad"))

rule_mad(n_mad = 2.5)
```

```

rule_iqr(k = 1.5)

rule_recursive(type = c("moving", "modified", "hybrid"), include_max = TRUE)

rule_ewma(lambda = 0.01, L = 1.5, chance = 0.5)

rule_mixture(
  distribution = c("exgaussian", "lognormal", "invgaussian"),
  bound = c("min", "max"),
  use_accuracy = FALSE,
  chance = 0.5,
  init = 0.05,
  max_prop = 0.5,
  maxit = 500,
  tol = 1e-06
)

rule_none()

rule_oracle(contaminant)

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

```

Arguments

min, max	Absolute bounds in seconds. Bounds are inclusive : a trial is flagged only when it falls strictly outside. max = Inf leaves the upper tail untouched.
n_sd, n_mad	Multiplier applied to the spread statistic.
center	Location statistic, "mean" or "median".
scale	Spread statistic, "sd" or "mad". "mad" uses <code>stats::mad()</code> and so carries the usual consistency constant of 1.4826.
k	Multiplier applied to the interquartile range when placing Tukey's fences. 1.5 marks an outlier, 3 a far-out point.
type	Which recursive criterion to use; see Details.
include_max	Whether the largest response time enters the mean and standard deviation from which the criterion is built. Applies to type = "moving" only.
lambda	Smoothing weight of the exponentially weighted moving average, in (0, 1]. Smaller values average over more trials.
L	Control-limit multiplier; the chart signals when the average departs from chance by more than L standard errors.
chance	Accuracy expected from a contaminant response, known from the design (0.5 for a two-alternative task).
distribution	Parametric distribution for the valid response time component of the mixture.

<code>bound</code>	Length-2 bounds of the uniform contaminant component. Each element may be a number, or "min"/"max" for a data-driven bound.
<code>use_accuracy</code>	Whether accuracy enters the mixture likelihood. Experimental ; see Details.
<code>init</code>	Starting value for the contaminant proportion.
<code>max_prop</code>	Upper bound on the estimated contaminant proportion.
<code>maxit</code>	Maximum number of EM iterations. A fit that reaches <code>maxit</code> is reported as un-converged and keeps every trial in its group. <code>bmm::flag_contaminant_rts()</code> uses 100; pass <code>maxit</code> explicitly when comparing the two packages.
<code>tol</code>	Convergence tolerance on the log-likelihood.
<code>contaminant</code>	Logical vector, one value per trial, marking which trials are contaminants. NA means unknown, and an unknown trial is kept.
<code>x</code>	A rule object.
<code>...</code>	Ignored.

Value

An object of class `c("rtprep_rule_<name>", "rtprep_rule")`: a list of validated parameters plus a `label` element used for the `.rule` column of `rt_screen()`.

Absolute cutoffs

The oldest and still the most common screen: discard anything faster than a plausible minimum or slower than a plausible maximum. Ratcliff (1993) is the standard treatment; Ulrich and Miller (1994) is the counterweight, showing that truncation biases the surviving distribution even when it removes real contaminants.

Bounds are inclusive, so `rule_cutoff(min = 0.18)` keeps a response time of exactly 180 ms, since 180 ms is not *below* 180 ms. Note that `trimr` uses strict comparisons, so the two implementations can disagree on a trial sitting exactly on a bound.

Location and spread criteria

`rule_sd()` flags trials further than `n_sd` spread units from a centre, recomputed within each group of `rt_screen()`. `center = "mean", scale = "sd"` is the standard deviation criterion, modal practice in the field; `center = "median", scale = "mad"` is the median absolute deviation criterion recommended by Leys et al. (2013), also available as `rule_mad()`.

One constructor covers both because they are one family: the same algorithm with a different location/spread pair. Miller (1991) is the standard warning about the standard deviation criterion: the proportion of a skewed distribution that survives a fixed multiplier depends on sample size, so the criterion silently changes what it removes as trial counts vary. That dependence is what `rule_recursive()` was designed to remove.

When the spread statistic cannot be used, because a group holds fewer than two observed trials or because the spread is zero, nothing is flagged; see [extending](#) for the contract this follows from.

Quartile fences

`rule_iqr()` flags trials outside Tukey's fences, $Q1 - k * IQR$ and $Q3 + k * IQR$, with quartiles at R's default type 7. $k = 1.5$ is Tukey's (1977) value and $k = 3$ his marker for a far-out point. It is the criterion a boxplot draws, and so the one behind "I removed the points outside the whiskers".

Not quite, though: `grDevices::boxplot.stats()` places the fences at `fivenum()` hinges rather than at type-7 quartiles, and the two differ at some sample sizes – for $n = 10$ and $n = 50$ in a quick check, not for $n = 9, 11$ or 51 . `rule_iqr()` uses the quartiles, which is what `stats::quantile()` and `ggplot2::geom_boxplot()` use.

It belongs to neither family above. `rule_cutoff()` fixes its bounds in advance; `rule_sd()` places them symmetrically around a centre. Tukey's fences are estimated from the data like the second and asymmetric like neither, so on a right-skewed response time distribution the upper fence sits further from the median than the lower one. That asymmetry is the reason to have it: a symmetric criterion on skewed data spends its budget in the tail the distribution is thin in.

Fewer than four observed trials in a group, or an interquartile range of zero, flags nothing.

Recursive and moving criteria

Van Selst and Jolicoeur (1994) answered Miller's (1991) sample-size problem by making the multiplier itself depend on the number of trials. `rtprep` ships their published criteria (their Table 4, tabulated at 4 to 15, 20, 25, 30, 35, 50, and 100 trials), linearly interpolated between the tabulated sample sizes as the table's note instructs and as `trimr` does. Below 4 no criterion exists and nothing is flagged; above 100 the value for 100 is used.

- `type = "moving"` is their non-recursive moving criterion: one pass, with the multiplier read off the table for the group's trial count.
- `type = "modified"` is the modified recursive procedure: the largest remaining response time is *temporarily set aside* while the mean and standard deviation are computed, the most extreme trial at each end is removed if it falls outside the resulting bounds, and the procedure repeats until nothing is removed or fewer than five trials remain. The temporary exclusion is what makes the rule bite, so it applies whatever `include_max` says; `include_max` governs `type = "moving"` only.
- `type = "hybrid"` averages the two. Per trial, `.prob` is the mean of the two rules' decisions and so takes the value 0, 0.5, or 1; under the default keep policy a trial survives only if both rules keep it.

Note that van Selst and Jolicoeur's published hybrid statistic is the mean of the two *condition means*, which no single per-trial keep vector can reproduce, because the two means have different denominators. To recover the published statistic, summarise the two rules separately and average:

```
keep_moving <- rt_screen(rt, rule = rule_recursive("moving"))$.keep
moving <- rt_summary(rt[keep_moving])
keep_modif <- rt_screen(rt, rule = rule_recursive("modified"))$.keep
modif <- rt_summary(rt[keep_modif])
(moving$mean_rt + modif$mean_rt) / 2
```

Accuracy control chart

`rule_ewma()` is the exponentially weighted moving average cutoff introduced with DMAT (Vandekerckhove & Tuerlinckx, 2007), and the only *published* screen that uses accuracy rather than response time alone. Trials are ordered from fastest to slowest and an exponentially weighted average of accuracy is accumulated, starting from chance. The control limit at trial i is

$$UCL_i = \gamma + L \sigma \sqrt{\frac{\lambda}{2 - \lambda} (1 - (1 - \lambda)^{2i})},$$

with γ the chance rate and $\sigma = \sqrt{\gamma(1 - \gamma)}$ the standard deviation of a guess. The cutoff is the response time at which the average first rises above the limit: below it, accuracy is indistinguishable from guessing, so those trials are flagged. If the average never crosses, nothing is flagged.

The chart lags, and the lag is a cost rather than an implementation detail: the average needs several trials above chance before it clears the limit, so the cutoff lands past the point where accuracy actually rose and valid trials just above it are flagged along with the guesses. Smaller lambda averages over more trials and overshoots further. Ratcliff and Kang (2021) note that this rule has not found much use; `rtprep` ships it as the published accuracy-based comparator, not as a recommendation.

This rule requires response, coded as correct/error rather than as an upper/lower boundary.

Mixture flagging

`rule_mixture()` fits, per group, a two-component mixture of a uniform contaminant distribution over bound and a parametric response time distribution, by expectation maximisation (Ratcliff & Tuerlinckx, 2002). `.prob` is then the posterior probability that a trial came from the response time component. This is the one rule in the package that returns something other than 0 and 1, and the reason `rt_screen()` separates the probability from the keep decision at all.

The bounds of the uniform component are buffered outward from the observed range by half its width. Without that buffer the uniform's edges sit exactly on data points and the mixture is barely identifiable.

Which core distribution you choose matters more than it looks. On a tight block of fast guesses the ex-Gaussian can absorb the block by widening its Gaussian part and driving tau to zero, reporting no contamination at all, where the lognormal and inverse Gaussian cores find it. The same thing happens in `bmm`'s implementation, so it is a property of the model rather than of either package. It is still a reason to check `attr(x, "fits")$contaminant_prop` against what you expected rather than trusting the default.

When the EM does not converge, or a group has fewer than five trials inside the bounds, **nothing is flagged** and `attr(x, "fits")$converged` is FALSE. `rt_screen()` warns once for the whole call, naming how many groups failed, rather than once per group. Note that `bmm` returns NA probabilities in this situation; `rtprep` keeps the data, because an NA would propagate into `.keep`.

Accuracy inside the likelihood

`use_accuracy = TRUE` is **experimental**. It puts accuracy inside the mixture likelihood rather than using it only afterwards, on the reasoning that a fast trial that is *correct* is less likely to be a guess than a fast trial that is an error, which an RT-only mixture cannot see. With y the accuracy indicator, γ the chance rate known from the design, and p_c the estimated accuracy of the decision process:

$$f(rt, y) = (1 - \pi)f_{RT}(rt | \theta)p_c^y(1 - p_c)^{1-y} + \pi U(rt | a, b)\gamma^y(1 - \gamma)^{1-y}.$$

γ is fixed, not estimated. The M-step for p_c is the responsibility-weighted mean of y , so the extension costs the EM almost nothing; the fitted value comes back as `p_correct` in `attr(x, "fits")`.

What is known so far, and it is not all good:

The staging is deliberate, and the reasons are concrete. Three things the package's own tests establish:

- **It can order overlapping guesses better than response time alone.** Where contaminants fall inside the valid distribution's range, which is the case RT-only detection fails at, the joint posterior ranks them more accurately.
- **But the fit tends to collapse.** A contaminant proportion of zero is a fixed point of this EM, and the accuracy factor widens its basin because it favours the valid component on every correct trial. On exactly the overlapping case above, the fit converges cleanly with $\pi \approx 10^{-7}$ and the rule removes nothing at all: the better ordering is one the keep policy never gets to act on. Check `collapsed` and `contaminant_prop` in `attr(x, "fits")` against what you expected.
- **It actively hurts when contaminants are as accurate as valid trials.** A delayed start-up still runs the decision process, so it is usually correct, and every correct contaminant has its contaminant evidence attenuated by γ/p_c . This is not neutrality: in the package's own test the joint model loses a large part of the sensitivity the RT-only model had.

The same deflation applies whenever observed accuracy is well above chance, which in most response time paradigms is always. Treat a contaminant proportion below the RT-only estimate as expected rather than as evidence of a cleaner data set.

Two things the method cannot enforce for you:

- It assumes contaminants respond at chance. Get chance wrong, by screening a four-alternative task at 0.5, and detection degrades sharply.
- response must be coded **correct/error**, not upper/lower boundary. Both are 0/1, so `rtprep` cannot tell them apart.

Nothing forces the valid component to be the accurate one either. When a fit comes back with p_c below chance the labels have swapped, which usually means the two components are not separable at that contamination rate. `rtprep` reports rather than constrains: `accuracy_inverted` in `attr(x, "fits")`, plus one warning per call.

No screening

`rule_none()` keeps every trial. It exists so that "no preprocessing" is an entry in the roster rather than a missing row, and so that pipelines can be compared against it without a special case.

Perfect exclusion

`rule_oracle()` removes exactly the trials you tell it are contaminants and nothing else. It is not a method: on real data nobody has the vector it needs. It is the ceiling the others are read against.

The reason to have it is that a drop rate and a hit rate do not say what a pipeline costs. Running the oracle through the same aggregation and the same estimation as a real rule gives the error that remains when screening is perfect, and the difference between the two is what the screening

decision actually bought. `r_contaminated()` returns the contaminant column this takes, so the comparison is two calls.

contaminant holds one value per trial and is subset alongside `rt`, so the rule works under `.by` grouping like any other.

References

- Ratcliff, R. (1993). Methods for dealing with reaction time outliers. *Psychological Bulletin*, 114(3), 510–532. doi:10.1037/00332909.114.3.510
- Ulrich, R., & Miller, J. (1994). Effects of truncation on reaction time analysis. *Journal of Experimental Psychology: General*, 123(1), 34–80. doi:10.1037/00963445.123.1.34
- Leys, C., Ley, C., Klein, O., Bernard, P., & Licata, L. (2013). Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *Journal of Experimental Social Psychology*, 49(4), 764–766. doi:10.1016/j.jesp.2013.03.013
- Miller, J. (1991). Reaction time analysis with outlier exclusion: Bias varies with sample size. *The Quarterly Journal of Experimental Psychology Section A*, 43(4), 907–912. doi:10.1080/14640749108400962
- Tukey, J. W. (1977). *Exploratory data analysis*. Addison-Wesley.
- Van Selst, M., & Jolicoeur, P. (1994). A solution to the effect of sample size on outlier elimination. *The Quarterly Journal of Experimental Psychology Section A*, 47(3), 631–650. doi:10.1080/14640749408401131
- Cousineau, D., & Chartier, S. (2010). Outliers detection and treatment: A review. *International Journal of Psychological Research*, 3(1), 58–67. doi:10.21500/20112084.844
- Vandekerckhove, J., & Tuerlinckx, F. (2007). Fitting the Ratcliff diffusion model to experimental data. *Psychonomic Bulletin & Review*, 14(6), 1011–1026. doi:10.3758/bf03193087
- Ratcliff, R., & Kang, I. (2021). Qualitative speed-accuracy tradeoff effects can be explained by a diffusion/fast-guess mixture model. *Scientific Reports*, 11, 15169. doi:10.1038/s41598021944517
- Liu, Y., Cheng, Y., & Liu, H. (2020). Identifying effortful individuals with mixture modeling response accuracy and response time simultaneously to improve item parameter estimation. *Educational and Psychological Measurement*, 80(4), 775–807. doi:10.1177/0013164419895068
- Ratcliff, R., & Tuerlinckx, F. (2002). Estimating parameters of the diffusion model: Approaches to dealing with contaminant reaction times and parameter variability. *Psychonomic Bulletin & Review*, 9(3), 438–481. doi:10.3758/bf03196302

See Also

`rt_screen()` to apply a rule; `screen_compare()` to apply several; `rule_all()` to combine them; `rule_hierarchical()` for a criterion pooled across groups. Two further, experimental rules are unexported and documented in `?rules_experimental`.

Examples

```
rule_cutoff(0.18, 3)

rule_sd(2.5)
rule_mad(3)
```

```

rule_iqr()
rule_iqr(3)

rule_recursive("modified")

rule_ewma(lambda = 0.05)

rule_mixture("lognormal")

rule_none()

truth <- rt_example$contaminant
oracle <- rule_oracle(truth)
oracle

# what a real rule leaves behind, against what perfect exclusion leaves
screen_compare(
  rt_example$rt,
  list(mad = rule_mad(2.5), oracle = oracle),
  .by = rt_example$id
)

```

rules_compose	<i>Combine screening rules</i>
---------------	--------------------------------

Description

A composite is a rule, so it goes wherever a rule goes: [rt_screen\(\)](#), [rt_keep\(\)](#), [screen_fits\(\)](#), [screen_compare\(\)](#).

Usage

```

rule_all(...)

rule_any(...)

rule_then(..., threshold = 0.5)

```

Arguments

...	Two or more rule objects.
threshold	Probability above which a trial survives a stage and is passed to the next. Only rule_then() takes this: which trials the later stages see is part of the rule, not part of the exclusion policy applied afterwards by rt_screen() .

Details

`rule_all()` keeps a trial only when every component keeps it, so the trials it removes are the *union* of what the components remove. `rule_any()` keeps a trial that any component keeps, so it removes only the *intersection*. `rule_then()` runs the components in order, each estimated on the trials the previous ones left.

Value

An object of class `c("rtprep_rule_<name>", "rtprep_rule")` holding the components in rules.

Which one you want

Screening rules disagree because they look in different places. The location and spread criteria and the recursive criteria read the slow tail; on anticipations at the leading edge of a right-skewed distribution they remove nothing, because those trials sit well inside a criterion built around the mean. `rule_ewma()` reads accuracy and finds the leading edge, and is blind to the slow tail. Combining one of each with `rule_all()` covers both, which no single conventional rule does.

`rule_then()` is the other idiom, and it is what a two-stage description like "trials below 200 ms were discarded, then trials more than 2.5 SD from each participant's mean" actually means: the standard deviation is computed after the floor, on the survivors. `trimr::sdTrim(minRT = , sd = )` does exactly this, and `rule_then(rule_cutoff(0.2), rule_sd(2.5))` reproduces it. Writing the two stages in parallel instead gives a different answer, because the fast trials are still inflating the standard deviation when it is estimated.

What `.prob` becomes

`rtprep` keeps the probability that a trial is valid separate from the decision to drop it, and a composite has to preserve that. `rule_all()` takes the smallest of the component probabilities and `rule_any()` the largest, so the composite's `.prob` is still a probability of validity and still drives `policy = "probabilistic"` and `rt_summary(weights = )`. For components that return 0 and 1 this reduces to the logical operation, and a mixture combined with a deterministic rule keeps its posterior wherever the deterministic rule does not veto.

`rule_then()` reports the probability from the stage that flagged the trial, or from the last stage to see it if none did. `.reason` comes from the first component to flag, in the order given.

What `screen_fits()` reports

One row per group, as always, with each component's diagnostics prefixed by its position: `r1_lower`, `r2_criterion`, and so on. `rule_then()` adds `r1_n_flagged`, `r2_n_flagged`, ... so it is visible how much work each stage did, which is the number a staged pipeline usually wants and rarely reports.

See Also

[rules](#) for the components.

Examples

```
# the slow tail and the leading edge, which no single rule covers
rule_all(rule_mad(2.5), rule_ewma())

# a floor, then a criterion estimated on what the floor left
staged <- rule_then(rule_cutoff(0.2), rule_sd(2.5))
staged

screen_fits(rt_example$rt, staged, .by = rt_example$id)
```

rule_custom	<i>A screening rule from a function, in one call</i>
-------------	--

Description

The short form of [new_rule\(\)](#): pass the function that decides which trials to keep, and get a rule back. No constructor, no S3 method, nothing to register. The result goes anywhere a rule goes — [rt_screen\(\)](#), [rt_keep\(\)](#), [screen_fits\(\)](#), [screen_compare\(\)](#), and inside [rule_all\(\)](#) and its relatives.

Usage

```
rule_custom(
  label,
  fun,
  ...,
  description = NULL,
  reason = "contaminant",
  needs_response = NULL,
  per_trial = character(),
  grouped = FALSE,
  subclass = "custom"
)
```

Arguments

label	A string identifying the rule, used as the <code>.rule</code> column, by <code>print()</code> , and by screen_compare() , which needs the rules it compares to be named apart. Conventionally "family(setting, setting)".
fun	The screening function. What it may take is listed under "What the screening function receives" in extending ; what it may return, under "What the screening function must return".
...	Named parameters stored on the rule and passed to <code>fun</code> by name.
description, reason, needs_response, per_trial, grouped	As in new_rule() .

subclass The rule family, for the rare case of wanting a class to hang a method on later. The default is shared by every `rule_custom()` rule, which costs nothing: the function travels on the object, not on the class, so two rules built this way never collide.

Value

An object of class `c("rtprep_rule_custom", "rtprep_rule_fun", "rtprep_rule")`.

See Also

[extending](#) for the full contract, [new_rule\(\)](#) to wrap a rule of your own in a constructor.

Examples

```
fast <- rule_custom(
  "fast(0.35)",
  function(rt, cut) rt >= cut,
  cut = 0.35,
  description = "Exclude trials faster than 350 ms.",
  reason = "too_fast"
)
fast
rt_screen(rt_example$rt, fast, .by = rt_example$id)

# against one of the rules rtprep ships
screen_compare(
  rt_example$rt,
  list(fast = fast, mad = rule_mad(2.5)),
  .by = rt_example$id
)
```

rule_hierarchical	<i>Hierarchical screening</i>
-------------------	-------------------------------

Description

A location and spread criterion whose centre and spread are shrunk towards the values pooled over all groups, rather than estimated from each group alone.

Usage

```
rule_hierarchical(
  n_sd = 2.5,
  n0 = 20,
  center = c("mean", "median"),
  scale = c("sd", "mad")
)
```

Arguments

n_sd	Multiplier applied to the shrunk spread.
n0	Trials at which a group is weighted equally between its own estimate and the pooled one. Larger values shrink harder. Inf gives one criterion for every group.
center	Location statistic, "mean" or "median".
scale	Spread statistic, "sd" or "mad".

Details

Experimental. No published convention exists for it, so it has no conventional setting to cite, and it is not a recommendation.

A per-participant criterion is estimated from the very data it is meant to clean. A participant with a handful of very slow trials has a mean and a standard deviation pulled outward by exactly those trials, so the criterion widens to admit them: the more contaminated a participant is, the less their own criterion removes. Estimating one criterion for everyone instead trades that for a different error, since participants really do differ in speed and variability.

Shrinkage sits between the two. Each group's centre is $w * \text{centre_group} + (1 - w) * \text{centre_pooled}$ with $w = n / (n + n0)$, and its spread is shrunk the same way on the log scale, which keeps it positive. $n0$ is the number of trials at which a group is weighted equally between its own estimate and the pooled one: $n0 = 0$ gives each group its own criterion, exactly as `rule_sd()` under the same grouping, and $n0 = \text{Inf}$ gives every group one common criterion. A group whose own spread cannot be computed pools completely, which is the case the method exists for.

Value

An object of class `c("rtprep_rule_hierarchical", "rtprep_rule")`.

Grouping

`.by` in `rt_screen()` names the units that are shrunk towards each other, normally participants. Everything in one call is pooled, so a `.by` that crosses participants with an experimental condition pools across conditions as well, and a condition that is genuinely slower drags every centre towards it. Screen one condition at a time.

See Also

[rules](#) for the criteria estimated within each group.

Examples

```
rule_hierarchical()

# what shrinkage changes, against the same criterion estimated per group
screen_compare(
  rt_example$rt,
  list(
    per_group = rule_mad(2.5),
    shrunk = rule_hierarchical(
```

```

    2.5,
    n0 = 20, center = "median", scale = "mad"
  )
),
.by = rt_example$id
)

```

r_contaminated

Generate response time data with contaminants of known type

Description

Draws trials from an evidence accumulation core and replaces a known subset with contaminants generated by one of three psychologically motivated processes. Which trials were replaced, and by which process, comes back with the data. That is what a real data set cannot give you, and what scoring a screen requires: sensitivity and specificity need labels.

Usage

```

r_contaminated(
  n,
  generator = c("ddm", "rdm"),
  process = c("leading_edge", "delay", "informationless", "mixed"),
  rate = 0.05,
  par = list(),
  st0 = 0,
  anticipation_anchor = c("min", "q10"),
  anticipation_depth = 0.3,
  delay_min = 0,
  delay_max = 2,
  lapse_prop = 0,
  mix = c(leading_edge = 1, delay = 1, informationless = 1)/3,
  dt = 0.001
)

```

Arguments

n	Number of trials.
generator	The clean core. "ddm" is a first-principles diffusion (Euler–Maruyama); "rdm" is a racing diffusion, two independent single-boundary Wald accumulators with a shared bound, drawn exactly from the inverse Gaussian.
process	Which contaminant process replaces the selected trials: "leading_edge" generates anticipations. Uniform in a band around the observed minimum of the <i>clean</i> trials in this call (see anticipation_anchor), responding at chance. Anchoring to an observed quantity rather than to the

	generating non-decision time keeps the construct comparable across generators, whose fitted non-decision times can differ substantially at identical observables.
	• "delay" generates delayed start-ups. A genuine trial plus a uniform shift of delay_min to delay_max seconds; the response, and so the intact accuracy, is kept. • "informationless" generates lapses of evidence quality. The trial is re-drawn from the core with its evidence scaled by lapse_prop while the speed of processing is held fixed: lapse_prop = 0 is a complete lapse (chance accuracy, response times still in the core's range), lapse_prop = 1 the clean process, and values between are partial lapses on the same continuum. • "mixed" draws each contaminant trial's process from mix.
rate	Probability that a trial is a contaminant, in $[0, 1)$.
par	Generator parameters as a list. For "ddm": drift, bound, ndt, zr (defaults 1.5, 1.2, 0.30, 0.5). For "rdm": drift as c(correct, error), bound, ndt (defaults c(3, 1.5), 1.5, 0.30). The defaults are illustrative, not calibrated; simulation designs should set parameters from an observable target instead.
st0	Range of across-trial variability in non-decision time, in seconds. The per-trial non-decision time is uniform on $\text{ndt} \pm \text{st0} / 2$, the centred parameterization, so the mean response time is invariant and only the leading edge smears.
anticipation_anchor	Observable the anticipation band is centred on: the minimum ("min") or the 10th percentile ("q10") of the clean trials generated in this call.
anticipation_depth	Half-width of the anticipation band as a proportion of the anchor: response times are uniform on $\text{anchor} * (1 \pm \text{anticipation_depth})$.
delay_min, delay_max	Bounds of the uniform shift, in seconds, for process = "delay".
lapse_prop	Evidence-quality proportion for process = "informationless"; see above.
mix	Relative weights of leading_edge, delay, and informationless for process = "mixed", in that order. Normalized internally.
dt	Euler–Maruyama step size for the "ddm" generator, in seconds. Smaller steps are slower and more faithful: a discretised first-passage sampler misses boundary crossings that happen inside a step, which shifts the response times slightly late. The default of one millisecond was validated against <code>rtddists::rddiffusion()</code> in the package's tests, where the two agree in accuracy and in the response time quantiles.

Value

A data.frame with n rows:

- rt is the response time in seconds,
- response is 1 for a correct (upper-boundary / winning-accumulator) response and 0 otherwise,
- contaminant is the logical ground truth,
- process is "clean" or the generating contaminant process per trial.

Reproducibility

There is no `set.seed()` anywhere in `rtprep`; reproducibility belongs to the caller.

The three processes are clusters, not point definitions

Each named process is one implementation of a psychological cluster, whether premature responding, late starts or disengagement, and the arguments (`anticipation_anchor`, `anticipation_depth`, `delay_min/delay_max`, `lapse_prop`) sweep within the cluster. Which implementations a simulation uses, and how many per cluster, is a design decision for that simulation to record and defend, not a package default.

References

Ratcliff, R. (1993). Methods for dealing with reaction time outliers. *Psychological Bulletin*, 114(3), 510–532. doi:[10.1037/00332909.114.3.510](https://doi.org/10.1037/00332909.114.3.510)

Ratcliff, R., & Tuerlinckx, F. (2002). Estimating parameters of the diffusion model: Approaches to dealing with contaminant reaction times and parameter variability. *Psychonomic Bulletin & Review*, 9(3), 438–481. doi:[10.3758/bf03196302](https://doi.org/10.3758/bf03196302)

See Also

`rt_screen()` for what the flags make of these trials, `rt_summary()` and `ez_ddm()` for what the contamination does to the parameters.

Examples

```
set.seed(1)
d <- r_contaminated(
  200,
  generator = "ddm", process = "leading_edge", rate = 0.1,
  par = list(drift = 1.5, bound = 1.2, ndt = 0.30)
)
table(d$process)
# detection scored against ground truth
scr <- rt_screen(d$rt, rule = rule_sd(2.5))
table(flagged = !scr$.keep, truth = d$contaminant)
```

screen_compare

Compare what several screening rules would remove

Description

Applies every rule once and reports where they disagree. This is the question the package exists to make askable: *what would a different preprocessing choice have removed?* That is normally unanswerable, because each rule's implementation returns a different shape.

Usage

```

screen_compare(
  rt,
  rules,
  response = NULL,
  .by = NULL,
  policy = c("threshold", "probabilistic"),
  threshold = 0.5
)

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

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

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

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

```

Arguments

rt, response, .by, policy, threshold	As in rt_screen() , and forwarded unchanged.
rules	A <code>list()</code> of rule objects; see rules . Names become the labels in the output; unnamed entries take the rule's own label.
x	An <code>rtprep_comparison</code> , or for <code>print.rtprep_comparison_summary()</code> an <code>rtprep_comparison_summary</code> .
...	Ignored.
object	An <code>rtprep_comparison</code> .

Details

agree and jaccard answer different questions and diverge exactly where it matters. Two rules that each drop 2% of trials and never the same one agree on 96% of decisions, and have a Jaccard index of zero. Agreement alone would call them interchangeable. Jaccard is NA, not 1, when neither rule dropped anything: no overlap can be computed from two empty sets.

Value

An object of class `rtprep_comparison`: a list with

`keep`, `prob`, `reason` matrices, `length(rt)` rows by one column per rule, so downstream analysis needs nothing else.

`drops` one row per rule and group: `n_trials`, `n_dropped`, `prop_dropped`, and a count column per reason.

agreement one row per rule pair: agree, jaccard, and the counts behind them.

fits the per-group fit diagnostics, stacked, with a .rule column.

summary() returns an rtprep_comparison_summary: the drops and agreement tables, printed by their own method. It is a value, not a side effect, so `s <- summary(cmp)` is quiet and `s$drops` is the table. `print()` and `plot()` return their input invisibly. `plot()` draws with `ggplot2` when it is installed and with `graphics::barplot()` when it is not; the return value is the same either way.

See Also

`rt_screen()` for a single rule, `r_contaminated()` to score the comparison against known ground truth.

Examples

```
set.seed(2)
d <- r_contaminated(300, process = "mixed", rate = 0.1)
cmp <- screen_compare(
  d$rt,
  list(
    cutoff = rule_cutoff(0.18, 3),
    sd = rule_sd(2.5),
    recursive = rule_recursive("modified")
  )
)
cmp
cmp$agreement
```

screen_fits

Per-group fit diagnostics from a screening rule

Description

The fits table of `rt_screen()` on its own: one row per group with the bookkeeping columns and whatever the rule reports. It exists because an attribute does not survive `dplyr::mutate()`, so the table is otherwise out of reach inside a pipeline.

Usage

```
screen_fits(
  rt,
  rule,
  response = NULL,
  .by = NULL,
  policy = c("threshold", "probabilistic"),
  threshold = 0.5
)
```

Arguments

rt	Numeric vector of response times in seconds . NA is allowed; non-positive values are an error.
rule	A rule object; see rules .
response	Optional response coding of the same length as rt, given as numeric 0/1, logical, or a character or factor using labels such as "correct"/"error" or "upper"/"lower". Required by rule_ewma() and by rule_mixture() with use_accuracy = TRUE.
.by	Optional grouping of the same length as rt: a vector, factor, list of vectors, or data frame. Rules are fitted separately within each group. NULL treats all trials as one group.
policy	Exclusion policy. "threshold" keeps a trial when its probability of validity exceeds threshold. "probabilistic" keeps it with that probability, drawing once per trial.
threshold	Cut for policy = "threshold", in [0, 1]. Ignored under the probabilistic policy.

Value

A data.frame with one row per group, in group order: .group, n_trials, n_dropped, prop_dropped, then the rule's own columns (bounds, criterion, iterations, EM convergence, and so on; see [rules](#)). n_dropped and prop_dropped count under the stated policy and threshold, exactly as attr(rt_screen(...), "fits") would.

See Also

[rt_screen\(\)](#), whose per-trial result carries this table as an attribute; [screen_compare\(\)](#) for the same table stacked over several rules.

Examples

```
rt <- c(0.12, 0.31, 0.35, 0.38, 0.42, 0.47, 0.55, 2.90)
id <- rep(c("a", "b"), each = 4)
screen_fits(rt, rule_sd(2), .by = id)

# with dplyr: dat |> reframe(screen_fits(rt, rule_sd(2.5)), .by = id)
```

Index

- * **datasets**
 - rt_example, 16
- [.rtprep_screen (rtprep_screen), 15
- adjust_accuracy, 2
- adjust_accuracy(), 23, 24
- apply_rule (extending), 6
- apply_rule(), 6–8
- apply_rule_grouped (extending), 6
- apply_rule_grouped(), 8
- as.character.rtprep_report (rtprep_report), 14
- as.data.frame.rtprep_screen (rtprep_screen), 15
- base::as.data.frame(), 16
- base::mean(), 22
- check_guessing, 4
- extending, 6, 23, 26, 33, 34
- ez_ddm, 10
- ez_ddm(), 3, 14, 22–24, 38
- format.rtprep_report (rtprep_report), 14
- format.rtprep_screen (rtprep_screen), 15
- glossary (rtprep-glossary), 13
- graphics::barplot(), 40
- grDevices::boxplot.stats(), 27
- new_rule (extending), 6
- new_rule(), 33, 34
- plot.rtprep_comparison (screen_compare), 38
- print.rtprep_comparison (screen_compare), 38
- print.rtprep_comparison_summary (screen_compare), 38
- print.rtprep_report (rtprep_report), 14
- print.rtprep_rule (rules), 24
- print.rtprep_screen (rtprep_screen), 15
- r_contaminated, 36
- r_contaminated(), 13, 16, 17, 30, 40
- report_screening, 11
- report_screening(), 14–16
- rt_example, 16
- rt_keep, 18
- rt_keep(), 6, 20, 21, 31, 33
- rt_screen, 19
- rt_screen(), 4–6, 8, 12–16, 18, 19, 22–24, 26, 28, 30, 31, 33, 35, 38–41
- rt_summary, 21
- rt_summary(), 3, 10, 11, 14, 21, 38
- rtprep-glossary, 13
- rtprep-package, 14
- rtprep_report, 13, 14
- rtprep_screen, 15
- rule_all (rules_compose), 31
- rule_all(), 30, 33
- rule_any (rules_compose), 31
- rule_custom, 33
- rule_custom(), 7, 9
- rule_cutoff (rules), 24
- rule_ewma (rules), 24
- rule_ewma(), 5, 18, 19, 41
- rule_hierarchical, 34
- rule_hierarchical(), 30
- rule_iqr (rules), 24
- rule_mad (rules), 24
- rule_mixture (rules), 24
- rule_mixture(), 8, 18, 19, 22, 41
- rule_none (rules), 24
- rule_oracle (rules), 24
- rule_oracle(), 8
- rule_recursive (rules), 24
- rule_sd (rules), 24
- rule_then (rules_compose), 31
- rules, 9, 18, 19, 21, 24, 32, 35, 39, 41

rules_compose, [31](#)

screen_compare, [38](#)

screen_compare(), [6](#), [21](#), [30](#), [31](#), [33](#), [41](#)

screen_fits, [40](#)

screen_fits(), [6](#), [8](#), [13](#), [19–21](#), [31](#), [33](#)

stats::mad(), [25](#)

stats::quantile(), [27](#)

summary.rtprep_comparison
 (screen_compare), [38](#)

toBibtex.rtprep_report(rtprep_report),
 [14](#)