Package: sandtablR (via r-universe)

November 3, 2026

Title Mass-Casualty Incident Simulation and Machine Learning Workbench

Version 0.0.0.9000

Description A generic framework for mass-casualty incident tabletop exercise simulation, decision-path analysis, and machine learning. Provides a discrete-event patient-flow engine, a tidy data model for scenarios, patient cards and decision paths, synthetic cohort generation, supervised triage and therapy-concept classification spanning interpretable and black-box models, sequence analysis of participant decision paths, and a model-agnostic explainability layer. An interactive 'shiny' workbench ties the pieces together for scenario building, simulation, model training, comparison and visualisation.

License MIT + file LICENSE

URL https://github.com/cttir/sandtablR, https://cttir.github.io/sandtablR/

BugReports https://github.com/cttir/sandtablR/issues

Depends R (>= 4.5.0)

Imports cli, dplyr, ggplot2, grDevices, methods, rlang, simmer, stats, tibble, utils, withr

Suggests bslib, chromote, covr, DALEX, DALEXtra, dials, earth, digest, DT, future, future.apply, ggiraph, glmmTMB, glmnet, hardhat, htmltools, jsonlite, keras3, kernlab, knitr, lme4, nanoparquet, nnet, parallelly, parsnip, plotly, quarto, ranger, reactable, readxl, recipes, rmarkdown, rpart, rsample, shiny, shinytest2, simstudy, spelling, stacks, synthpop, testthat (>= 3.2.0), TraMineR, tune, WeightedCluster, workflows, workflowsets, writexl, xgboost, yardstick, zip

VignetteBuilder knitr

Config/testthat/edition 3

Config/Needs/website CTTIR/themakR

Config/Needs/coverage covr

Encoding UTF-8

Language en-GB

Roxygen list(markdown = TRUE)

LazyData true

Config/roxygen2/version 8.0.0

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

Date/Publication 2026-10-29 11:42:48 UTC

RemoteUrl https://github.com/CTTIR/sandtablR

RemoteRef HEAD

RemoteSha 826ca2a990e63fef77b851a9bd88391fd256ce9f

Contents

as_tibble.scenario . . . 3
autoplot.scenario . . . 4
format.patient_card . . . 5
predict.sdt_model . . . 5
sdt_add_decisions . . . 6
sdt_app . . . 7
sdt_app_server . . . 8
sdt_app_ui . . . 8
sdt_apply_gold_standard . . . 9
sdt_area_labels . . . 10
sdt_as_patient_cards . . . 11
sdt_cohort . . . 12
sdt_consistency_score . . . 13
sdt_course_composition . . . 13
sdt_decision_consistency . . . 15
sdt_decision_paths . . . 16
sdt_describe . . . 17
sdt_describe_participants . . . 20
sdt_describe_render . . . 21
sdt_engines . . . 21
sdt_error_fidelity . . . 22
sdt_explain . . . 24
sdt_export_simulation . . . 25
sdt_fidelity . . . 27
sdt_fit . . . 30
sdt_fit_behaviour . . . 32
sdt_fit_triage_error . . . 35
sdt_importance . . . 38
sdt_metric_set . . . 39
sdt_metrics . . . 40
sdt_participant_dictionary . . . 40
sdt_participant_summary . . . 42

sdt_patient_card . . . 43
sdt_plot_board . . . 44
sdt_plot_end_state . . . 45
sdt_plot_fidelity . . . 46
sdt_plot_flow . . . 47
sdt_plot_sequences . . . 47
sdt_plot_triage_mix . . . 48
sdt_plot_utilisation . . . 49
sdt_predictions . . . 49
sdt_profile . . . 50
sdt_read_gold_standard . . . 51
sdt_read_participants . . . 52
sdt_reference . . . 54
sdt_reference_horizon . . . 55
sdt_reference_scenario . . . 55
sdt_roc . . . 56
sdt_run_app . . . 57
sdt_run_behaviour . . . 57
sdt_scenario . . . 59
sdt_seq_cluster . . . 59
sdt_seqdef . . . 60
sdt_shap . . . 61
sdt_simulate . . . 62
sdt_simulate_groups . . . 63
sdt_simulate_triage . . . 66
sdt_synthesis_utility . . . 68
sdt_synthesise . . . 68
sdt_synthetic_cohort . . . 69
sdt_transformer . . . 70
sdt_triage_accuracy . . . 71
sdt_vocabularies . . . 72
sdt_workflowset . . . 73
summary.game_run . . . 75
validate_cohort . . . 76
validate_patient_card . . . 77

Index **78**

`as_tibble.scenario` *Coerce a scenario to a tibble*

Description

Returns the scenario's casualty cohort. Provided so a scenario can be piped straight into any function that expects a data frame.

Usage

```
## S3 method for class 'scenario'
as_tibble(x, ...)
```

Arguments

x	A scenario object.
...	These dots are for future extensions and must be empty.

Value

The cohort tibble held by the scenario.

Examples

```
sc <- sdt_scenario(sdt_synthetic_cohort(n = 5, seed = 1))
tibble::as_tibble(sc)
```

---

autoplot.scenario *Plot a scenario*

Description

The ggplot2::autoplot() method for a scenario; dispatches to sdt_plot_triage_mix().

Usage

```
## S3 method for class 'scenario'
autoplot(object, ...)
```

Arguments

object	A scenario object.
...	These dots are for future extensions and must be empty.

Value

A **ggplot2** object.

Examples

```
sc <- sdt_scenario(sdt_synthetic_cohort(n = 20, seed = 1))
ggplot2::autoplot(sc)
```

format.patient_card *Format a patient card*

Description

Renders a `patient_card` as a single-line character summary. `print()` shows exactly this string, so the two cannot drift apart.

Usage

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

Arguments

`x`	A `patient_card` object.
`...`	These dots are for future extensions and must be empty.

Value

A length-1 character vector.

Examples

```
card <- sdt_patient_card(
  "P01", 34, "m", 4L, TRUE, "thorax", "red", "DCS", 1L
)
format(card)
```

predict.sdt_model *Predict from a fitted mass-casualty model*

Description

Scores new casualties with an `sdt_fit()` model. The return is always a tibble, whatever the engine, so downstream code does not have to branch on which model produced it.

Usage

```
## S3 method for class 'sdt_model'
predict(object, new_data, ...)
```

Arguments

object	An sdt_model from sdt_fit().
new_data	A data frame of casualties to score.
...	Passed to the underlying stats::predict() method; type = "prob" returns class probabilities rather than a hard call.

Value

A tibble::tibble() with one row per row of new_data.

See Also

sdt_predictions() for the held-out predictions the model's own metrics were computed from.

Examples

```
# See ?sdt_fit for a runnable example.
sdt_engines()
```

---

sdt_add_decisions *Add participant decisions to a cohort*

Description

Simulates the sighting category and therapy concept a participant assigns to each casualty, using the published per-category hit rates. Where the participant is wrong, the miscall is drawn from the neighbouring categories, since confusing red with yellow is far more likely than confusing red with green.

Usage

```
sdt_add_decisions(cohort, hit = NULL, therapy_hit = 0.75, seed = NULL)
```

Arguments

cohort	A cohort from sdt_cohort() or sdt_synthetic_cohort().
hit	Named numeric vector of per-category hit rates. Defaults to the published triage_hit.
therapy_hit	Scalar probability that the therapy concept is called correctly.
seed	Optional integer seed, applied with withr::local_seed().

Value

The cohort with two added columns: called_triage and called_therapy, the participant's calls.

See Also

sdt_consistency_score() to score the calls against the truth.

Examples

```
co <- sdt_synthetic_cohort(n = 200, seed = 1)
called <- sdt_add_decisions(co, seed = 2)
sdt_consistency_score(called$triage, called$called_triage)

# Hit rate per true category, against the published targets.
table(called$triage, called$called_triage)
```

---

sdt_app *The sandtablR workbench as a Shiny app object*

Description

Returns a fully wired shiny::shinyApp() object: use it to deploy the workbench, to mount it behind a router, or to run it yourself.

Usage

```
sdt_app()
```

Details

Prefer this to assembling sdt_app_ui() and sdt_app_server() by hand. The package serves its own logo from an addResourcePath() mount, and that registration is process-local: an app object built in one process and served from another registers nothing in the process that answers the request. This constructor mounts the assets from onStart, which runs in the serving process before it takes any request.

Value

A shiny.appobj.

See Also

sdt_run_app() to launch it, sdt_app_ui() and sdt_app_server() for the pieces.

Examples

```
if (requireNamespace("shiny", quietly = TRUE) &&
    requireNamespace("bslib", quietly = TRUE)) {
  app <- sdt_app()
  class(app)
}
```

sdt_app_server *Server for the sandtablR workbench*

Description

All six modules share one `shiny::reactiveValues()` holding the scenario, the game run, the fitted models and the leaderboard. Nothing is passed between modules by any other route, so there is one place to look when the app and the console disagree.

Usage

```
sdt_app_server(input, output, session)
```

Arguments

`input`, `output`, `session`
: Supplied by **shiny**.

Value

Invisible `NULL`, called for its side effects.

See Also

`sdt_run_app()`, `sdt_app_ui()`.

Examples

```
# Driven headlessly in the package's own tests:
# shiny::testServer(sdt_app_server, { session$setInputs() })
sdt_app_ui
```

sdt_app_ui *User interface for the sandtablR workbench*

Description

User interface for the sandtablR workbench

Usage

```
sdt_app_ui()
```

Value

A **shiny** UI definition.

See Also

sdt_run_app(), sdt_app_server().

Examples

```
if (requireNamespace("shiny", quietly = TRUE) &&
    requireNamespace("bslib", quietly = TRUE)) {
  ui <- sdt_app_ui()
  class(ui)
}
```

---

sdt_apply_gold_standard *Overwrite a cohort's placeholders with the gold standard*

Description

Joins the gold-standard codebook onto a cohort, or onto any frame carrying a card id, and replaces the named columns with the deck's own values. The replaced columns are kept as <col>_original, so the provenance is not lost and the inference that was there before can still be scored against the truth that replaced it.

Usage

```
sdt_apply_gold_standard(
  cohort,
  gold,
  on = "card",
  replace = c("triage", "age", "sex"),
  keep_original = TRUE
)
```

Arguments

cohort	A cohort from sdt_cohort(), or any data frame with a card id column.
gold	An sdt_gold_standard from sdt_read_gold_standard().
on	Name of the card id column in cohort. Values may be padded ("07"), unpadded ("7") or numeric; all three join.
replace	Which columns to overwrite: any of "triage", "age" and "sex". Pass character() to join nothing and only stamp triage_source.
keep_original	Keep each replaced column as <col>_original.

Details

In the TDSC corpus the three replaceable columns start out as a circular surrogate (`triage`, the inter-team consensus call), a constant (`age`) and an inference from the log's gendered wording (`sex`). After this function they are the card's own values, and `triage_source` reads `"gold"` so a downstream result can say which label it routed on. The engine routes on `triage`, so which label sits there decides which question a simulation answers.

Value

`cohort` with the named columns replaced, the originals kept when asked for, and a `triage_source` column reading `"gold"`. A `cohort` comes back a validated `cohort`; anything else comes back a `tibble::tibble()`. The `gold_applied` attribute records what was done.

Provenance

`triage_source` names where the label sitting in `triage` came from, so it reads `"gold"` only when `triage` was among the columns replaced. Ask for a subset that leaves `triage` alone and any existing source label is left alone with it: a surrogate must never end up wearing the truth's label.

See Also

`sdt_read_gold_standard()`, `sdt_triage_accuracy()`.

Examples

```
## Not run:
gold <- sdt_read_gold_standard("Patienten Codebook.xlsx")
truth <- sdt_apply_gold_standard(pooled, gold)
mean(truth$triage == truth$triage_original)

## End(Not run)
```

---

sdt_area_labels *Display labels for board areas*

Description

Turns board-area names into the labels used on figures and in the workbench, so a legend never shows a column name such as `"icu_imc"`. Anything unrecognised is passed through unchanged rather than becoming `NA`, which keeps the function total over user-supplied vocabularies.

Usage

```
sdt_area_labels(areas = sdt_board_areas())
```

Arguments

`areas`	Character vector of board areas. Defaults to `sdt_board_areas()`.

Value

A character vector of labels, the same length as `areas`.

See Also

`sdt_board_areas()`, and `summary.game_run()`, whose `reference = TRUE` folds the board onto the published reporting locations.

Examples

```
sdt_area_labels()
sdt_area_labels(c("or", "morgue", "nowhere"))
```

---

sdt_as_patient_cards *Explode a cohort into patient cards*

Description

Turns each row of a `cohort` back into a `patient_card`. The inverse of stacking cards into a cohort; useful when code wants to work with one casualty at a time.

Usage

```
sdt_as_patient_cards(x)
```

Arguments

`x`	A `cohort` from `sdt_cohort()`.

Value

A list of `patient_card` objects, named by casualty `id`.

Examples

```
co <- sdt_cohort(sdt_synthetic_cohort(n = 3, seed = 1))
cards <- sdt_as_patient_cards(co)
cards[[1]]
```

`sdt_cohort` *Build a validated casualty cohort*

Description

Coerces a data frame of casualties into a `cohort`, a `tibble::tibble()` subclass carrying the column contract shared by `sdt_patient_card()` and `sdt_scenario()`. Validation is vectorised: every column is checked in one pass and *all* offending rows are reported, rather than aborting on the first bad one.

Usage

```
sdt_cohort(data, ...)
```

Arguments

`data`	A data frame with the columns produced by `sdt_patient_card()`: `id`, `age`, `sex`, `life_points`, `ventilated`, `injury_region`, `triage`, `therapy`, `arrival_order`. Extra columns are kept and carried through untouched.
`...`	These dots are for future extensions and must be empty.

Value

A `cohort` object: a tibble subclass with the contract columns first, ordered by `arrival_order`. A zero-row input yields a zero-row cohort with the correct column types.

See Also

`validate_cohort()` to check an existing cohort, `sdt_as_patient_cards()` to explode one into cards.

Examples

```
co <- sdt_cohort(sdt_synthetic_cohort(n = 5, seed = 1))
co

# Every bad row is named at once, not just the first.
bad <- sdt_synthetic_cohort(n = 4, seed = 1)
bad$triage[c(2, 4)] <- "purple"
try(sdt_cohort(bad))
```

sdt_consistency_score *Agreement between two labellings of the same casualties*

Description

Cohen's kappa between two sets of labels for the same units: a participant's calls against the card's true category, one participant against another, or a model's predictions across a repeated scenario. Kappa rather than raw agreement, because on a cohort that is 44% red, two people who both guess "red" every time agree 44% of the time while sharing no judgement at all.

Usage

```
sdt_consistency_score(truth, estimate, levels = NULL)
```

Arguments

`truth`	Reference labels.
`estimate`	Labels to compare against `truth`.
`levels`	Optional level set. Defaults to the union of both inputs, so a level nobody used does not distort the expected agreement.

Value

A one-row `tibble::tibble()` with `.metric`, `.estimator`, `.estimate` and `n`.

Examples

```
if (requireNamespace("yardstick", quietly = TRUE)) {
  truth <- c("red", "red", "yellow", "green", "green")
  called <- c("red", "yellow", "yellow", "green", "green")
  sdt_consistency_score(truth, called)
}
```

sdt_course_composition *Aggregate participant metadata to course composition*

Description

Collapses a `sdt_participants` register to one row per **course**, keyed by the JSON `file` name so it joins straight onto the corpus, and carries the composition covariates a model can use.

Usage

```
sdt_course_composition(
  x,
  impute = c("flag", "median", "none"),
  min_participants = 3L
)
```

Arguments

x	A sdt_participants object from sdt_read_participants().
impute	How to handle a covariate a course has no observation for: "flag", "median" or "none". See the details.
min_participants	Below this many recorded participants a course is marked as too small to average.

Value

A sdt_course_composition object: a tibble with one row per course, including courses with no participants at all, which come through with has_participants = FALSE, n_participants = 0 and every covariate NA or imputed-and-flagged. A course whose sheet was not read carries sheet_read = FALSE and n_participants = NA, so a sheets = subset never masquerades as a set of cancelled courses.

Why course level, and what that costs

The register has no group, team or "Gruppe" column, so a participant cannot be placed in group A rather than group B. Composition is therefore a block-level covariate: in y ~ card + (1 | block) + (1 | group) it can compete with the **session** random effect and can never explain the **team** random effect. Every group inside one course gets the same row. Fitting on this table is a between-course regression with as many points as there are courses, and a covariate that barely varies between courses cannot explain anything at all — check the spread before using one.

The two halves of the missingness policy

Every covariate comes with three companions: n_<covariate>, the number of participants the course mean was taken over; n_imputed_<covariate>, how many of those numbers came from a participant-level fill rather than an answer somebody wrote down (only the academic ladder has one by default); and <covariate>_missing, whether the course value itself was filled. The two count columns always tell the truth, whatever impute is set to.

- impute = "flag" (the default) fills a missing covariate with the corpus-wide value — the pooled mean over **all** participants, not the mean of course means — and sets the indicator. A model can then carry "this was missing" as information instead of pretending it was observed.
- impute = "median" fills with the between-course median and does **not** set the indicator. It exists because a downstream pipeline can refuse a column that is perfectly collinear with has_participants, which is exactly what the indicator is when the only empty courses are the ones with no participants; dropping the indicator then buys rank instead of information. It

is not the default because it makes an imputed value indistinguishable from an observed one, which is the thing this module exists to prevent. Only `n_<covariate>` still gives it away.

- `impute = "none"` leaves `NA` and still sets the indicator.

A course with fewer than `min_participants` recorded people is marked in `below_min_participants`. Its observed value is **kept**, not overwritten: the flag is what stops it being read as if it were solid, and `n_participants` is there to weight or filter on. A share computed from two people is not missing, it is noisy, and those are different problems.

See Also

`sdt_read_participants()`, `sdt_participant_summary()`.

Examples

```
## Not run:
p <- sdt_read_participants("TDSC_course_participant_relations.xlsx")
comp <- sdt_course_composition(p)
comp[, c("short_course_id", "n_participants", "share_consultant")]

# How much does each covariate actually vary between courses?
vapply(comp[, c("share_female", "share_consultant")],
       stats::sd, numeric(1), na.rm = TRUE)

## End(Not run)
```

---

sdt_decision_consistency

Score decision consistency

Description

Quantifies how consistently the same casualty is routed across replications. Two measures are offered, both returning a score in `[0, 1]` where 1 is perfectly consistent.

Usage

```
sdt_decision_consistency(run, method = c("om", "entropy"))
```

Arguments

`run`	A `game_run` from `sdt_simulate()`.
`method`	`"om"` (default) or `"entropy"`.

Details

"om" compares each casualty's paths with one another by optimal matching: the mean pairwise edit distance between its replication paths, normalised by the longer of each pair. This is the measure that answers "does this casualty get handled the same way twice?".

"entropy" scores each path on its own, as the normalised Shannon entropy of the time it spent in each area. A casualty that sat in one area all game scores 1; one moved through many areas scores lower. That describes the shape of a single trajectory rather than agreement between trajectories, so it is meaningful even with a single replication.

Value

A tibble::tibble() with one row per casualty: id, n_paths (how many replication paths contributed), n_distinct (how many of them differ), and consistency. With method = "om" and a single replication there is nothing to compare, so consistency is NA.

See Also

sdt_decision_paths(), sdt_seq_cluster().

Examples

```
co <- sdt_synthetic_cohort(n = 12, seed = 1)
run <- sdt_simulate(sdt_scenario(co), replications = 4, seed = 1,
                    verbose = FALSE)
sdt_decision_consistency(run)
sdt_decision_consistency(run, method = "entropy")
```

---

sdt_decision_paths *Build decision paths from a game run*

Description

Extracts, for each casualty in each replication, the ordered sequence of board areas visited.

Usage

```
sdt_decision_paths(run, format = c("long", "wide"))
```

Arguments

run	A game_run from sdt_simulate().
format	"long" for one row per casualty per step, or "wide" for one row per casualty per replication with a column per position. The wide form is the state-sequence (STS) layout sdt_seqdef() expects. Every casualty appears in every replication. One the game was called before reaching gets a single-step path of "not_seen" rather than being left out: never being screened is a pathway the exercise produces and the source reports, and a sequence analysis that silently dropped those casualties would be computed on a different denominator than the one it claims.

Value

A `tibble::tibble()`. Long: `replication`, `id`, `step`, `area`, `start`, `end`, `duration`. Wide: `replication`, `id`, then `step_1` to `step_k`, padded with `NA` for casualties whose path was shorter.

See Also

`sdt_seqdef()`, `sdt_decision_consistency()`, `sdt_plot_sequences()`.

Examples

```
co <- sdt_synthetic_cohort(n = 10, seed = 1)
run <- sdt_simulate(sdt_scenario(co), replications = 2, seed = 1,
                    verbose = FALSE)
sdt_decision_paths(run)
sdt_decision_paths(run, format = "wide")
```

---

sdt_describe *Describe a table without hiding what is missing*

Description

A descriptive table over a `sdt_participants` register, a `sdt_course_composition` table, or any data frame. Every variable reports its counts, its missingness and the summary its type deserves – counts and percents for a categorical variable, mean, SD, median, IQR and range for a numeric one – optionally split by a grouping column.

Usage

```
sdt_describe(
  x,
  by = NULL,
  vars = NULL,
  na_report = c("row", "column", "footnote"),
  digits = 2
)

## S3 method for class 'sdt_describe'
format(x, ..., max_groups = 8L, max_levels = 12L)

## S3 method for class 'sdt_describe'
print(x, ..., max_groups = 8L, max_levels = 12L)
```

Arguments

`x`	A `sdt_participants` object, a `sdt_course_composition` object, or any data frame.

by	Optional single column name to split by. NULL, the default, pools every row. Rows whose grouping value is missing form their own group rather than disappearing.
vars	Optional character vector of columns to describe. Defaults to a curated set for the two classes this package produces, and to every non-indicator column otherwise.
na_report	Where to surface missingness: "row", "column" or "footnote". It cannot be switched off.
digits	Decimal places for the printed statistics. Percentages use one fewer, since a percent already carries two significant figures more than the proportion it comes from. The returned tibble is never rounded.
...	These dots are for future extensions and must be empty.
max_groups	Groups to print before the rest are summarised in one line. The object keeps every group either way.
max_levels	Levels to print per categorical variable.

Value

A sdt_describe object: a tibble::tibble() subclass with one row per group, variable and level. Two families of counts share every row and are named apart so they cannot be confused. n, n_level_observed and n_level_imputed belong to the row – to one level of one categorical variable, or to the values of one numeric variable. n_observed, n_imputed, n_missing, n_valid and n_records belong to the variable, and are repeated on each of its rows, so that n_observed + n_imputed + n_missing == n_records holds on every row of the table. The three percents, pct_missing and the numeric statistics complete it. tibble::as_tibble() returns it bare; sdt_describe_render() returns the display-ready shape.

Every percent states its denominator

A categorical level carries three shares, and all three are in the returned tibble whatever the report style:

- pct_observed – over answered rows only. Its denominator excludes both missing rows and filled ones.
- pct_valid – over rows that carry a value. Its denominator excludes missing rows and includes filled ones.
- pct_total – over every row in the group. Its denominator includes everything, missing rows included.

pct_valid and pct_total are the two a reader usually means by "the percent", and they differ exactly when something is missing. pct_observed differs from pct_valid exactly when something was imputed.

Missingness cannot be switched off

na_report decides **where** missingness is surfaced, never whether it is:

- "row" – each categorical variable gains a (missing) row, and each numeric variable a missing line of its own.
- "column" – the missing count and percent appear beside every row.
- "footnote" – they are collected under each block, complete variables named as such so silence is never ambiguous.

n_missing and pct_missing are columns of the returned tibble in all three cases.

Imputed values are flagged, not absorbed

Where the input carries indicators – .missing_<field> and .imputed_<field> from sdt_read_participants(), <covariate>_missing and n_<covariate> from sdt_course_composition() – each row is placed in exactly one of three states: observed, imputed or missing. Imputed rows are counted separately, never as observations, and numeric variables carry mean_observed, sd_observed and median_observed alongside the all-values statistics so both are visible at once.

For a variable with a fill rule, n_imputed + n_missing is the number of rows nobody answered: some of them were filled and some were not, and the two are reported apart rather than added up for the reader. That is why title_level reports no missing values and 99 imputed ones on the TDSC register – an empty title box is filled with none, and the fill is visible in every share it touches.

A value that is present in a row whose own missing indicator is set is not an observation. It is counted as imputed when an imputation indicator or a zero denominator (n_<covariate> of 0) says it was filled, and as missing otherwise. That zero-denominator rule is what catches sdt_course_composition(impute = "median"), which fills without flagging: the fill is reported as imputed anyway, and named as unflagged.

Level, not group

The participant register has no group column, so a composition descriptive split by course is a between-course description with as many points as there are courses. There is no split by group to be had.

See Also

sdt_describe_participants() for the participant defaults, sdt_participant_summary() for the participant-only summary this generalises, sdt_course_composition() for the imputing half of the policy.

Examples

```
d <- data.frame(
  grade = c("consultant", "resident", "consultant", NA),
  years = c(11, 2, 7, NA),
  .missing_grade = c(FALSE, FALSE, FALSE, TRUE),
  .imputed_grade = c(FALSE, FALSE, FALSE, FALSE)
)
sdt_describe(d, vars = c("grade", "years"))
tibble::as_tibble(sdt_describe(d, vars = "grade"))
```

sdt_describe_participants *Describe a participant register*

Description

sdt_describe() with the defaults a sdt_participants register wants: split by course, and reporting the normalised covariates rather than the raw German free text they were derived from.

Usage

```
sdt_describe_participants(x, by = "short_course_id", ...)
```

Arguments

x	A sdt_participants object from sdt_read_participants().
by	Column to split by. Defaults to "short_course_id"; NULL pools the whole register.
...	Passed to sdt_describe(), including vars, na_report and digits.

Details

Courses with no linked participants have no rows in the register, so they cannot appear as groups. They are named in the printed header instead, read from the course_index attribute, so their absence is stated rather than left for the reader to notice.

Value

A sdt_describe object.

See Also

sdt_describe(), sdt_read_participants().

Examples

```
## Not run:
p <- sdt_read_participants("TDSC_course_participant_relations.xlsx")
sdt_describe_participants(p)
sdt_describe_participants(p, by = NULL, na_report = "footnote")

## End(Not run)
```

sdt_describe_render *The display-ready shape of a descriptive table*

Description

Renders a `sdt_describe()` object as a flat table of character columns, one row per level or statistic, ready for `knitr::kable()`, `DT` or any other table renderer. The numbers are formatted with the object's `digits`; the object itself keeps them unrounded.

Usage

```
sdt_describe_render(x)
```

Arguments

`x`	A `sdt_describe` object.

Details

Missingness follows the object's `na_report`: a `(missing)` row, a pair of `missing` columns, or a footnote. The legend naming each percent's denominator is always the first footnote, so a rendered table can never arrive without one.

Value

A `tibble::tibble()` of character columns with a `footnotes` attribute.

See Also

`sdt_describe()`.

Examples

```
d <- data.frame(grade = c("consultant", "resident", NA), years = c(9, 3, 5))
sdt_describe_render(sdt_describe(d))
```

sdt_engines *Model engines available to the workbench*

Description

Lists the supported engines, the package each needs, whether that package is installed, and how readable the fitted model is.

Usage

```
sdt_engines()
```

Details

`class_box` grades readability by where an explanation comes from: `"glass"` when the fitted model can simply be read, `"grey"` when its own structure yields exact per-feature attributions, `"black"` when attributions come only from perturbing the inputs, and `"closed"` when the model learns its own representation first, so a per-input attribution no longer describes what the model reasons in. Gradient boosting sits in the middle rather than with the black boxes, which is where a two-way split would put it.

Value

A `tibble::tibble()` with columns `engine`, `package`, `class_box` and `installed`.

Examples

```
sdt_engines()
```

---

sdt_error_fidelity *Score how faithfully a simulation reproduces triage error*

Description

Compares simulated triage calls against observed ones **per true category**, and reports two ratios for every statistic: the level and the between-team standard deviation, each as observed over simulated, each judged inside the same 0.90-1.15 band the rest of the package uses. Getting the average error rate right while making every team err identically is the failure this function exists to catch, so the level alone is never enough.

Usage

```
sdt_error_fidelity(
  sim,
  observed = NULL,
  truth = NULL,
  estimate = NULL,
  group = NULL,
  card = NULL,
  cards = NULL,
  min_coverage = NULL,
  n_boot = 200L,
  seed = NULL
)
```

Arguments

`sim`	Simulated calls: an `sdt_simulated_triage` object from `sdt_simulate_triage()`, or a data frame with one row per (team, card) carrying a true and a called category.

observed	The observed calls, in the same shape. Optional when sim carries its fitted model, in which case the model's own fitting frame is used.
truth, estimate, group, card	Column names. NULL takes them from the fitted model where there is one, and otherwise from the usual candidates.
cards	Optional character vector fixing the matched frame's cards. NULL chooses them, as described above.
min_coverage	Minimum number of observed teams a card must be seen by to enter the matched frame. NULL chooses it from the data.
n_boot	Bootstrap replicates for the intervals. 0 skips them.
seed	Optional integer seed, for a reproducible bootstrap.

Value

An S3 object of class sdt_error_fidelity, a tibble with one row per (statistic, measure): statistic, category, quantity, measure ("level" or "sd"), observed, simulated, ratio, ci_lower, ci_upper, verdict, control, n_obs, n_sim, location_z, overlap, the four percentile columns, reason, n_cards, n_obs_rows, n_sim_rows and n_sim_reps. Carries band, location_tol, frame, structural, group_rates and n_boot attributes. The column set is a superset of what sdt_plot_fidelity() needs, so the table can be drawn with it.

Why this is not an argument to sdt_fidelity()

sdt_fidelity() scores unconditional per-group rates, and enforces composition matching on the value: a group is dropped for a statistic unless its non-missing cells cover every shared card. A conditional statistic – the hit rate among the red cards – is undefined on every card that is not red, so under that rule every group is dropped on both sides. Handed such a statistic, sdt_fidelity() does not fail loudly; it returns the row with n_obs = 0, n_obs_rows = 0, an NA ratio and the reason "not evaluable", which is the correct answer to the question it was asked and no answer at all to this one. The conditional case also needs a different denominator per category, a per-category composition check, and two ratios per statistic rather than one. What the two share is the verdict: the band and the location tolerance are read from sdt_fidelity()'s own helpers, so they cannot drift apart.

What is scored

For each true category, and for the pooled "overall" row, the hit rate, the over-triage rate and the under-triage rate. Direction is the clinically meaningful part: over-triage spends a scarce resource, under-triage can cost a life, and a single accuracy figure hides which one a team is doing.

Two pairs are zero by construction and are excluded rather than failed: over-triage on the most severe category and under-triage on the least, since there is nothing above red or below green to move to. They are listed in the structural attribute. Note also that for those same two categories the remaining error rate is the exact complement of the hit rate, so red under and green over carry the same information as red hit and green hit; they are reported because the direction of error is what a reader is looking for, not because they are independent evidence.

The matched frame

Real play is truncated by the clock, so observed teams saw different subsets of the deck while every simulated team is dealt all of it. Keeping only the teams that saw the whole intersection would therefore keep the handful that finished, so the card set is chosen the way `sdt_fit_behaviour()` chooses it: the threshold that maximises cards times complete teams, with ties going to the frame that keeps more teams. Pass `cards` to fix the set by hand, or `min_coverage` to fix the threshold.

The two sides must then agree about which true category each kept card carries. A category left with fewer than three cards is warned about: a team's rate there can take only a handful of values, so its between-team SD is as much a property of the deck as of the teams.

See Also

`sdt_simulate_triage()`, `sdt_fidelity()`, `sdt_triage_accuracy()`.

Examples

```
set.seed(1)
cards <- sprintf("c%02d", seq_len(12))
truth <- rep(c("red", "yellow", "green"), each = 4)
d <- expand.grid(card = cards, group_id = sprintf("g%02d", seq_len(20)),
                 stringsAsFactors = FALSE)
d$session_id <- ifelse(d$group_id < "g11", "s1", "s2")
d$triage <- truth[match(d$card, cards)]
b <- stats::rnorm(20, sd = 1)
p <- stats::plogis(1 + b[match(d$group_id, unique(d$group_id))])
hit <- stats::runif(nrow(d)) < p
wrong <- vapply(seq_len(nrow(d)), function(i) {
  sample(setdiff(sdt_triage_levels(), d$triage[[i]]), 1)
}, character(1))
d$called_triage <- ifelse(hit, d$triage, wrong)
if (requireNamespace("lme4", quietly = TRUE)) {
  fit <- sdt_fit_triage_error(d, verbose = FALSE)
  sim <- sdt_simulate_triage(fit, n_groups = 20, seed = 1)
  sdt_error_fidelity(sim, n_boot = 50, seed = 1)
}
```

---

sdt_explain *Explain a fitted mass-casualty model*

Description

Wraps a fitted `sdt_fit()` model in a **DALEX** explainer, so permutation importance, dependence profiles and per-casualty attributions all work off one object.

Usage

```
sdt_explain(model, data = NULL, y = NULL)
```

Arguments

model	An sdt_model from sdt_fit().
data	Data to explain on. Defaults to the held-out half of the model's own split, which is the right choice: explaining a model on the rows it was fitted to flatters it.
y	Optional true outcome vector; taken from data when omitted.

Value

A **DALEX** explainer.

See Also

sdt_importance(), sdt_profile(), sdt_shap().

Examples

```
# See ?sdt_importance for a runnable example.
sdt_engines()
```

---

sdt_export_simulation *Export a simulation run as a self-describing bundle*

Description

Writes a simulated cohort, the fidelity metrics that judge it, the deck it was drawn over, a README and a manifest.json into one archive that can be handed to someone who was not in the room. The archive is the return value, so the whole function is a one-liner inside a Shiny downloadHandler().

Usage

```
sdt_export_simulation(
  x,
  path,
  format = c("csv", "parquet", "rds"),
  metrics = NULL,
  overwrite = FALSE,
  pseudonymise = TRUE,
  crosswalk = NULL,
  allow_git = FALSE,
  zip = TRUE,
  verbose = TRUE
)
```

Arguments

x	A sdt_simulated_groups table from sdt_simulate_groups(), or any data frame with sim_group and card columns. Model provenance is read from attr(x, "model") or attr(x, "meta") when present.
path	Destination. With zip = TRUE the archive is written at exactly this path — pass a .zip name in a script, or the file argument of a downloadHandler() unchanged. With zip = FALSE it is the directory to fill.
format	Storage format for the tables: "csv" (widest compatibility), "parquet" (via **nanoparquet**) or "rds" (lossless R). The README and manifest are always plain text.
metrics	A fidelity table from sdt_fidelity(). Defaults to attr(x, "fidelity") when that is present. A bundle without metrics is written, and says in as many words that it makes no fidelity claim.
overwrite	Overwrite an existing destination?
pseudonymise	Replace course identifiers with stable codes? Defaults to TRUE; see the section above before turning it off.
crosswalk	Optional path, outside the bundle, for the pseudonym crosswalk as CSV. NULL discards the mapping.
allow_git	Permit a destination inside a git working tree?
zip	Write a single archive (TRUE) or a directory (FALSE)?
verbose	If TRUE, report what was written via **cli**.

Value

Invisibly, the written paths: the archive when zip = TRUE, the member files otherwise. The archive path is always the first element, and carries the member names in attr(., "members").

What the bundle contains

- simulated-groups.* — one row per simulated group and card.
- fidelity-metrics.* — the output of sdt_fidelity(), when supplied.
- deck-cards.* — the cards the simulation ranged over.
- README.md — generated at write time; carries the caveats.
- manifest.json — versions, seed, frame, model terms, random-effect SDs, skipped outcomes, the privacy scheme, and a SHA-256 of every other file.

Refusals

The writer aborts rather than doing something quietly regrettable:

- an existing destination is never overwritten unless overwrite = TRUE;
- a destination inside a git working tree is refused unless allow_git = TRUE, because synthetic rows that get committed are read as real data by the next person;
- with pseudonymise = TRUE the written bytes are searched for the original identifiers before the file is closed, and a survivor is an error.

Pseudonymisation

Session identifiers and scenario names in this domain embed hospital, venue and city names and the exact course date. They are therefore mapped to `S01...Snn`, groups to `G001...Gnnn` and scenarios to `SC01...SCnn` by default. The crosswalk is written only if `crosswalk` names a path, and that path must lie outside the bundle; otherwise the mapping is discarded when the function returns and the bundle cannot be re-identified from itself.

See Also

`sdt_fidelity()` for the metrics, `sdt_simulate_groups()` for the input.

Examples

```
sim <- tibble::tibble(
  sim_group = rep(c("sim_01", "sim_02"), each = 2L),
  card = rep(c("01", "02"), 2L),
  red = c(TRUE, FALSE, TRUE, TRUE)
)
out <- sdt_export_simulation(
  sim,
  tempfile("demo-bundle-", fileext = ".zip"),
  verbose = FALSE
)
utils::unzip(out, list = TRUE)$Name

# In Shiny, the handler's `file` is the archive path:
# downloadHandler(
#   filename = function() "simulation-bundle.zip",
#   content = function(file) {
#     sdt_export_simulation(sim, file, metrics = fid, verbose = FALSE)
#   }
# )
```

---

sdt_fidelity *Score how faithful a simulation is to the real thing*

Description

Compares simulated groups against observed groups on the statistics that matter for a tabletop exercise, and reports one row per statistic. The headline is the **dispersion ratio**: the between-group standard deviation of a behavioural rate, observed divided by simulated. A simulator that reproduces the average behaviour but makes every team look the same fails here, which is the whole point of the function.

Usage

```
sdt_fidelity(x, observed = NULL, statistics = NULL, n_boot = 200L, seed = NULL)
```

Arguments

x	The simulated groups: an sdt_simulated_groups object from sdt_simulate_groups(), a game_run from sdt_simulate() (converted with sdt_run_behaviour()), or a plain data frame with one row per (group, card).
observed	The observed groups, in the same shape. Optional when x carries its fitted model, in which case the model's own training frame is used.
statistics	Which statistics to score. NULL (the default) scores every behavioural column the two frames share. A character vector selects columns by name. A named list of functions defines derived statistics; each is called with a data frame and must return one logical or numeric value per row, and each is applied to **both** frames, so the columns it uses must exist in both. A shared character or factor column is expanded into one statistic per level, named column=level.
n_boot	Bootstrap replicates for the interval on ratio. 0 skips the bootstrap and returns NA bounds.
seed	Optional integer seed, for a reproducible bootstrap.

Value

A tibble with one row per statistic, ordered with the failures first: statistic, observed, simulated, ratio, ci_lower, ci_upper, verdict, control, n_obs, n_sim, and then observed_mean, simulated_mean, location_z, overlap, the four percentile columns, reason, n_obs_rows, n_sim_rows, n_obs_missing, n_sim_missing and n_sim_reps. n_obs and n_sim count groups – the sample size of each standard deviation, with n_sim counted per simulated replicate – while the _rows columns count the (group, card) episodes behind them and the _missing columns count the episodes that statistic could not be computed on, so a rate built on a partly missing column cannot pass unnoticed.

What is compared

For each statistic, each group gets one number – the share of its patient cards on which the statistic was true. The spread of those group-level rates is the quantity of interest, and the columns report:

- observed, simulated: the between-group SD of the group rates.
- ratio: observed / simulated. Inf when the simulator produces no between-group spread at all, which is a real and reportable result.
- ci_lower, ci_upper: a percentile bootstrap interval on ratio, resampling groups on each side with the simulated cohort held fixed. It answers "how much would this move with a different sample of teams?".
- rep_lower, rep_upper, rep_covers_one: the interval across replicates – observed / SD(replicate) for each simulated cohort, at the 2.5th and 97.5th percentiles. It answers the other question, "how much would this move on a different RUN of the simulator?", which is the one that dominates for a rare behaviour and which the bootstrap cannot see. Needs at least ten replicates; NA below that.
- observed_mean, simulated_mean, location_z: the level check. location_z is the difference in pooled rates expressed in observed between-group SDs, so a model can also fail by getting the level wrong.

- `overlap`: the overlapping coefficient of the two distributions of group-level rates, from 0 (disjoint) to 1 (identical). Together with the `observed_p10`/`observed_p90`/`simulated_p10`/`simulated_p90` columns it shows how far the simulated teams reach, rather than only how wide they are on average.
- `verdict`: `"PASS"` only when the dispersion ratio lies inside 0.90 and 1.15 **and** `abs(location_z)` is at most 0.25. `reason` names whichever check failed.

The full per-group rates behind the summary are attached as the `"group_rates"` attribute of the result, and the frame the comparison ran on as `"frame"`.

The composition-matched frame

Between-group dispersion is only meaningful when every group is scored on the same cards. `sdt_fidelity()` enforces that: it intersects the card sets of the two frames, then keeps only groups that saw every card in the intersection, and records what it dropped in the `"frame"` attribute. On the real course corpus, skipping this step reports no team signal at all, because each truncated group saw a different deck and that composition variance swamps the behavioural one.

Negative controls

Outcomes fixed by the patient card rather than chosen by the team – `died` and `ventilated` by default – are marked `control = TRUE`. They are the pipeline's own alarm, and they read backwards from the other rows: a control is *expected* to pass even for a simulator with no team term at all, so a **failing control means the frame or the pipeline is wrong, not the model**. Check the matched frame before you believe anything else in the table. The one exception is a simulator with no team-level randomness at all: the discrete-event engine reads `died` and `ventilated` off the patient card and never redraws them, so its controls are dead flat and fail too. That is a finding about the engine – it has no team layer of any kind – rather than about the frame, and the way to tell the two apart is that a broken frame moves the controls a little while a missing team layer flattens them completely.

What the bootstrap resamples

Whole groups, with replacement, independently on each side, and the ratio is recomputed from the resampled group rates. The group is the unit the dispersion is a statement about and the unit that is exchangeable in both frames; resampling individual (group, card) episodes instead would break the very clustering being measured and would report an interval far too narrow. Where the simulated frame carries several replicates, each is resampled within itself and the resulting SDs averaged, exactly as the point estimate is formed.

See Also

`sdt_plot_fidelity()` to draw the result, `sdt_run_behaviour()` to score the discrete-event engine on the same statistics.

Examples

```
set.seed(1)
cards <- sprintf("c%02d", 1:6)
# Ten observed groups that genuinely differ from one another.
obs <- expand.grid(group_id = sprintf("g%02d", 1:10), card = cards,
```

```
                    stringsAsFactors = FALSE)
team <- stats::setNames(stats::runif(10, 0.1, 0.9), sprintf("g%02d", 1:10))
obs$surgery <- stats::runif(nrow(obs)) < team[obs$group_id]

# A simulator that gets the average right and the spread badly wrong.
sim <- expand.grid(sim_group = sprintf("s%02d", 1:10), card = cards,
                    stringsAsFactors = FALSE)
sim$surgery <- stats::runif(nrow(sim)) < mean(team)

sdt_fidelity(sim, obs, n_boot = 50, seed = 1)
```

`sdt_fit` *Fit a triage or therapy classification model*

Description

Trains a supervised classifier predicting either the sighting category or the therapy concept of a casualty from its card, through a full **tidymodels** path: a preprocessing recipe, stratified cross-validation, a tuning grid, and a final fit evaluated on a held-out split.

Usage

```
sdt_fit(
  data,
  target = c("triage", "therapy", "called_triage", "called_therapy"),
  engine = NULL,
  predictors = NULL,
  tune = TRUE,
  grid = 5L,
  folds = 5L,
  prop = 0.75,
  parallel = FALSE,
  workers = NULL,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

`data`	A cohort or data frame of casualties.
`target`	Outcome to predict. `"triage"` and `"therapy"` are the true values on the card; `"called_triage"` and `"called_therapy"` are what a participant assigned, as produced by `sdt_add_decisions()`. The true values are close to a deterministic function of the card, so every engine scores near 1 on them and the comparison is uninformative. The participant's call is the target worth benchmarking.
`engine`	One of `sdt_engines()`. `"multinom"`, `"glmnet"` and `"tree"` are glass-box; `"mars"` and `"xgboost"` are grey-box; `"ranger"`, `"svm_rbf"` and `"mlp"` are black-box.

predictors	Character vector of predictor columns. Defaults to the card fields, excluding identifiers and every outcome column, so a model for the participant's call cannot simply read the true category.
tune	If TRUE, tune hyperparameters over a grid; if FALSE, fit the specification's defaults once.
grid	Number of candidate hyperparameter settings.
folds	Number of cross-validation folds, stratified on the target.
prop	Proportion of the data used for training in the initial split.
parallel	If TRUE, fit resamples and grid points in parallel. Defaults to FALSE: a package that claims every core without being asked is a bad guest on a shared machine, and a sequential run needs no thought about RNG streams. It pays off on a tuned fit or an engine screen and costs a little on a single untuned one. The backend is set up for the duration of the call and the caller's plan is restored on exit.
workers	Number of parallel workers, or NULL to choose one: a core short of the machine, capped at 8.
seed	Optional integer seed, applied with withr::local_seed().
verbose	If TRUE, report progress via **cli**.

Details

The returned object keeps the split, the resamples and the tuning results, not only the fitted workflow, so the numbers reported later can be traced back to what produced them.

Value

An S3 object of class sdt_model with elements workflow, engine, target, predictors, class_box, split, resamples, tuning, best and metrics.

See Also

sdt_workflowset() to screen every engine at once, sdt_metrics(), sdt_roc(), sdt_explain().

Examples

```
co <- sdt_synthetic_cohort(n = 200, seed = 3)
needed <- c("parsnip", "recipes", "workflows", "rsample", "tune",
            "yardstick", "rpart")
if (all(vapply(needed, requireNamespace, logical(1), quietly = TRUE))) {
  fit <- sdt_fit(co, target = "triage", engine = "tree",
                 tune = FALSE, seed = 1, verbose = FALSE)
  fit
  sdt_metrics(fit)
}
```

`sdt_fit_behaviour` *Fit a hierarchical model of how teams behave*

Description

Fits one binomial mixed model per outcome to observed (group, card) decision episodes:

Usage

```
sdt_fit_behaviour(
  data,
  outcomes = NULL,
  group = "group_id",
  block = "session_id",
  card = "card",
  engine = "lme4",
  nAGQ = 1L,
  frame = c("matched", "all"),
  min_coverage = NULL,
  block_data = NULL,
  block_covariates = NULL,
  verbose = TRUE
)
```

Arguments

`data`	A data frame of decision episodes, one row per (group, card).
`outcomes`	Character vector of outcome names, or `NULL` to auto-detect.
`group`, `block`, `card`	Column names holding the team identifier, the session (block) the team played in, and the patient card. `block` may be `NULL` to fit without a session term. Group labels that repeat across blocks are nested inside their block, with a warning.
`engine`	`"lme4"` (default) or `"glmmTMB"`.
`nAGQ`	Quadrature points for the **lme4** engine. 1 (default) is the Laplace approximation. A larger value uses adaptive Gauss-Hermite quadrature, which removes most of the upward bias the Laplace approximation carries in the variance component when clusters hold very few events – the failure mode behind an over-dispersed simulated cohort for a rare behaviour. **lme4** supports it only for a model with one grouping factor, so it requires `block = NULL`; asking for it on a cross-classified model is an error rather than a silent downgrade.
`frame`	`"matched"` (default) for the composition-matched frame, or `"all"` for every row.
`min_coverage`	Minimum number of groups a card must be seen by to enter the matched frame. `NULL` chooses it from the data.

`block_data` Optional data frame with one row per block, keyed by the `block` column, holding block-level covariates – typically the output of `sdt_course_composition()`, which keys on `file`.

`block_covariates` Optional character vector naming the columns of `block_data` that enter the fixed part of the formula. Passed together with `block_data` or not at all.

`verbose` If `TRUE`, report progress via **cli**.

Details

```
y ~ card + (1 | block) + (1 | group)
```

The card fixed effect carries what the casualty is; the group random effect carries what the team is. That second term is the point. It is what lets `sdt_simulate_groups()` draw a *new* team rather than another replication of the ones already in the data, and it is what the discrete-event engine in `sdt_simulate()` has no equivalent of.

Value

An S3 object of class `sdt_behaviour_model`:

`fits` Named list of fitted models, one per modelled outcome.

`outcomes` Names of the outcomes that were modelled.

`frame` `"matched"` or `"all"`, as requested.

`terms` The formula template and the column names it was built from, including `block_covariates`.

`data` The frame actually fitted on, with the derived outcome columns attached.

`meta` Counts, the frame definition, engine, package versions, runtime, and `sds`: per outcome, the rows used, the rows dropped as missing, the prevalence, and the group and block standard deviations on the logit scale. With block covariates it also carries `composition` (the centring and scaling used) and `block_variance` (the block SD before and after, per outcome).

`skipped` One row per outcome not modelled, with the reason.

The composition-matched frame

Real play is truncated by the clock, so groups see different subsets of the deck. Any between-group statistic computed on that unbalanced frame mixes team behaviour with *which cards the team reached*, and the second term is much larger than the first. `frame = "matched"` therefore keeps only cards seen by at least `min_coverage` groups and only the groups that saw all of them, so every group in the fit answered the same questions. `frame = "all"` fits on everything and warns: a simulator deals every card to every group, so it cannot reproduce composition spread, and the dispersion ratio is then wrong for a reason that has nothing to do with the model. Measured on the validation corpus, the full deck reports 1.10 for TASC and 1.33 for eFAST where the matched frame built from the same rows reports 1.00 and 1.02.

With `min_coverage = NULL` the threshold is chosen to maximise cards times groups – the number of complete decision episodes – with ties going to the frame that keeps more groups.

Course composition as a block covariate

`block_data` is a table with one row per block, keyed by the `block` column – the shape `sdt_course_composition()` returns, which keys on `file`, so fit with `block = "file"` to join on it. `block_covariates` names the columns that enter the fixed part, giving

```
y ~ card + <covariates> + (1 | block) + (1 | group)
```

The random block intercept **stays**. The covariates are there to explain part of the session spread, not to replace the term, and with fifteen informative courses a handful of between-course slopes cannot be trusted to replace it.

They also cannot reach further than that. Participants are linked to courses, never to the groups inside them, so composition is constant within a block: it competes with the **session** random effect and can never explain the **team** random effect, which is the one `sdt_simulate_groups()` draws a new team from. A course-level covariate cannot make a simulated team more or less like any other team in its own course.

Numeric and logical covariates are accepted; each is centred and scaled over the distinct blocks, so a course counts once however many groups it ran, and the centring is recorded in `$meta$composition$covariates` so the simulator can invert it. A covariate that is constant across blocks, that is missing for a block being fitted, or that is collinear with another is refused, and every offender is named in one message.

Each outcome is then fitted **twice**, with the covariates and without them on exactly the same rows, and `$meta$block_variance` reports the block SD before and after, the share of block variance removed, and the deviance and AIC change. That doubles the fitting time and it is the point of the exercise: the answer is allowed to be "nothing", and on this corpus it usually is.

Outcomes

`outcomes = NULL` auto-detects everything the catalogue can build from the columns present: `tasc`, `dcs`, `etc` and `conservative` from `therapy` or the `did_*` flags; `efast`, `xray` and `ct` from `did_efast`, `did_xray` and `did_ct`; `red`, `yellow` and `green` from `called_triage`; `icu` and `stranded` from `general_location`; `ventilated` and `died` from the columns of those names. You may instead pass a character vector naming catalogue outcomes, logical columns of `data`, or both.

A consensus or modal triage label is deliberately *not* a source. It is derived from what the teams did, so modelling it would model the model.

What gets skipped

An outcome that cannot be built, is constant, has too few events or groups, or whose fit fails or does not converge is recorded in `$skipped` with its reason, never dropped in silence. A fit that converges to a zero team variance is *kept* and flagged `singular`: that is a finding about the outcome – teams did not differ – and hiding it would hide the negative result the fidelity report exists to surface. A fit whose team SD lands above 5 on the logit scale is flagged `boundary` for the mirror-image reason: that is separation, and simulated teams will be over-dispersed.

See Also

`sdt_simulate_groups()` to draw new teams from the fit.

Examples

```
set.seed(1)
d <- expand.grid(
  card = sprintf("c%02d", seq_len(8)),
  group_id = sprintf("g%02d", seq_len(12)),
  stringsAsFactors = FALSE
)
d$session_id <- ifelse(d$group_id < "g07", "s1", "s2")
b <- stats::rnorm(12, sd = 1.2)
p <- stats::plogis(-0.5 + b[match(d$group_id, unique(d$group_id))])
d$did_efast <- stats::runif(nrow(d)) < p
if (requireNamespace("lme4", quietly = TRUE)) {
  fit <- sdt_fit_behaviour(d, outcomes = "efast", verbose = FALSE)
  fit
}
```

---

sdt_fit_triage_error *Fit a model of triage error, with its between-team variance*

Description

Models the called sighting category **conditional on the true one**, so the object being fitted is a row of the confusion matrix rather than a marginal rate. Where `sdt_fit_behaviour()` answers "how often does a team call red", this answers "given that this casualty is red, what does a team call, and how far do teams differ in getting it wrong".

Usage

```
sdt_fit_triage_error(
  data,
  truth = "triage",
  estimate = "called_triage",
  group = "group_id",
  block = "session_id",
  card = "card",
  engine = "lme4",
  verbose = TRUE,
  card_effect = c(hit = "fixed", split = "none"),
  frame = c("all", "matched"),
  min_coverage = NULL,
  joint = FALSE
)
```

Arguments

`data`	A data frame of decision episodes, one row per (group, card), carrying both the true and the called category – the shape `sdt_apply_gold_standard()` returns.

`truth`, `estimate` Columns holding the true and the called sighting category. Both must hold `sdt_triage_levels()`.

`group`, `block`, `card` Columns holding the team, the session the team played in, and the patient card. `block` may be `NULL` to fit without a session term. Group labels that repeat across blocks are nested inside their block, with a warning.

`engine` `"lme4"` (default) or `"glmmTMB"`.

`verbose` If `TRUE`, report progress via **cli**.

`card_effect` How the card enters each part: one or two of `"fixed"`, `"random"` and `"none"`. See the sections above.

`frame` `"all"` (default) or `"matched"` for the composition-matched frame of `sdt_fit_behaviour()`. The default differs from that function on purpose: there dispersion is measured off the same frame that was fitted, while here the card term is in the model and the between-team dispersion is measured separately by `sdt_error_fidelity()`, which does its own matching. Matching here would cost a category its cards – on this corpus the matched frame keeps 2 of the 5 yellow cards.

`min_coverage` Passed to the matched frame; `NULL` chooses it.

`joint` If `TRUE`, also fit the full multinomial mixed model per category as a diagnostic. Slow, and not used by the simulator.

Value

An S3 object of class `sdt_triage_error_model`:

`cells` One row per (category, part) fitted, with `n_obs`, `prevalence`, the card effect actually used, `sd_group`, `sd_block`, `sd_card`, the null fit's `sd_block_null` and `aic_null`, `delta_aic`, `chisq`, `df`, `p_value`, and the `singular`, `boundary` and `separated` flags.

`fits`,`fits_null` Named lists of fitted models, keyed `"category.part"`, with and without the team term.

`joint` `NULL`, or one row per category with the joint fit's team correlation, per-part SDs, AIC and why it was rejected.

`terms` Column names, card levels, the card-to-truth lookup and the wrong-answer ordering the split refers to.

`data` The frame actually fitted on.

`meta` Counts, frame, engine, versions, runtime and the observed confusion matrix.

`skipped` One row per cell not modelled, with the reason.

The parameterisation

For each true category `k` the response has three levels and therefore two free logits. They are fitted as a sequential pair:

- **hit** – `P(called = k | true = k)`, over every row with truth `k`.
- **split** – `P(called = w1 | called != k, true = k)`, over the error rows only, where `w1` is the more severe of the two remaining categories.

Each is a binomial mixed model

```
y ~ card + (1 | block) + (1 | group)
```

This is an **exact reparameterisation** of the three-category multinomial, not an approximation to it: the multinomial likelihood factorises into the two binomial likelihoods, so fitting them separately fits the multinomial with the team random effects of the two logits held independent. The single parameter the decomposition gives up is the correlation between a team's accuracy and its error direction, and `joint = TRUE` estimates exactly that (see below).

Why the card term is load-bearing

Cards inside one true category are not alike. On the validation corpus red card 14 was called correctly by 26% of teams and 15 of the 30 red cards by every team. Removing the card term does not shrink the team effect, it **erases** it: the fitted team SD goes to exactly 0 for red and for yellow, against 0.544 and 0.639 with the card term, and the AIC is 473 and 137 worse. Every group saw nearly the same deck, so the spread of raw hit rates across teams is *narrower* than binomial noise around a single pooled probability, and a model with one probability per category can only account for that by setting the team variance to zero.

`card_effect` therefore defaults to `"fixed"` for the hit part. It takes one value for both parts, or two – `c("fixed", "none")` sets the hit part then the split part – or a named vector such as `c(hit = "fixed", split = "random")`.

The split part defaults to `"none"`, and that is measured rather than assumed. Its cells hold few error rows with a small minority (176, 124 and 529 rows with 10, 24 and 38 minority calls on this corpus); a card-fixed split runs the yellow team SD up to 41 on the logit scale, is worse by AIC than no card term for red, and leaves any card that no team ever miscalled with no coefficient at all. It also costs little, because for red both wrong answers are under-triage and for green both are over-triage: the split moves cells *inside* the confusion matrix without moving the over- and under-triage rates.

The ladder, and what gets skipped

A card term is attempted only where there are at least as many minority events as card levels, and a fit is rejected if it is rank deficient, fails to converge, or lands a random-effect SD above 5 on the logit scale – which is separation, not a tendency. Each rejection demotes the cell one rung down `"fixed"`, `"random"`, `"none"` and is recorded in the cell's `demotions`. A cell that runs out of rungs, or that is constant, or that has too few events or groups, appears in `$skipped` with its reason and is never dropped in silence. A fit that lands a zero team variance is **kept** and flagged `singular`: teams not differing is a finding, not a failure.

The full multinomial

`joint = TRUE` stacks a category's two parts and fits them together with an unstructured team covariance, which is the full multinomial mixed model for that category. It is a **diagnostic**: it measures the one parameter the default gives up, and `sdt_simulate_triage()` never reads it. Run on the validation corpus it converged for green only – correlation `-0.050`, and a joint AIC equal to the sum of the two separate ones – while red came back rank deficient with a team correlation of 0.99 and yellow ran its split SD to 33-40. The full multinomial does not fit reliably at this n, and where it does fit it says the constraint costs nothing.

The team term

Every cell is fitted **twice**: once as above and once with `(1 | group)` removed from exactly the same rows. `$cells` carries `sd_group`, the null fit's AIC, `delta_aic` and a likelihood-ratio test, so the necessity of the team term is a measured claim per cell rather than a stated one. The null fits are kept, and `sdt_simulate_triage()` can simulate from them with `team_effects = FALSE`.

See Also

`sdt_simulate_triage()` to draw new teams from the fit, `sdt_error_fidelity()` to score them, `sdt_triage_accuracy()` for the observed table.

Examples

```
set.seed(1)
cards <- sprintf("c%02d", seq_len(12))
truth <- rep(c("red", "yellow", "green"), each = 4)
d <- expand.grid(card = cards, group_id = sprintf("g%02d", seq_len(20)),
                 stringsAsFactors = FALSE)
d$session_id <- ifelse(d$group_id < "g11", "s1", "s2")
d$triage <- truth[match(d$card, cards)]
b <- stats::rnorm(20, sd = 1)
p <- stats::plogis(1 + b[match(d$group_id, unique(d$group_id))])
hit <- stats::runif(nrow(d)) < p
wrong <- vapply(seq_len(nrow(d)), function(i) {
  sample(setdiff(sdt_triage_levels(), d$triage[[i]]), 1)
}, character(1))
d$called_triage <- ifelse(hit, d$triage, wrong)
if (requireNamespace("lme4", quietly = TRUE)) {
  sdt_fit_triage_error(d, verbose = FALSE)
}
```

---

sdt_importance *Permutation variable importance*

Description

How much worse the model gets when each predictor is shuffled. The measure is model-agnostic, so a random forest and a decision tree are ranked on the same scale.

Usage

```
sdt_importance(x, data = NULL, n_permutations = 10L)
```

Arguments

`x`	An `sdt_model` from `sdt_fit()`, or an explainer from `sdt_explain()`.
`data`	Optional data to explain on, passed to `sdt_explain()`.
`n_permutations`	Number of permutation rounds. More is steadier and slower.

Value

A **ggplot2** object.

See Also

`sdt_profile()`, `sdt_shap()`.

Examples

```
co <- sdt_add_decisions(sdt_synthetic_cohort(n = 200, seed = 3), seed = 4)
needed <- c("parsnip", "recipes", "workflows", "rsample", "tune",
            "yardstick", "rpart", "DALEX", "DALEXtra")
if (all(vapply(needed, requireNamespace, logical(1), quietly = TRUE))) {
  fit <- sdt_fit(co, target = "called_triage", engine = "tree",
                 tune = FALSE, seed = 1, verbose = FALSE)
  sdt_importance(fit)
}
```

---

sdt_metric_set *The metric set the workbench scores on*

Description

Area under the ROC curve, accuracy, Cohen's kappa and macro F. Kappa is included deliberately: on a cohort that is 44% red, accuracy alone rewards a model for guessing the majority class.

Usage

```
sdt_metric_set()
```

Value

A **yardstick** metric set.

Examples

```
if (requireNamespace("yardstick", quietly = TRUE)) {
  sdt_metric_set()
}
```

sdt_metrics *Metrics for a fitted model*

Description

Returns the held-out performance of an `sdt_fit()` model, always with the same three columns whatever engine produced it.

Usage

```
sdt_metrics(model)
```

Arguments

`model`	An `sdt_model` from `sdt_fit()`.

Value

A `tibble::tibble()` with columns `.metric`, `.estimator` and `.estimate`.

Examples

```
# See ?sdt_fit for a runnable example.
sdt_metric_set
```

sdt_participant_dictionary *Normalisation dictionary for participant metadata*

Description

The TDSC participant register stores four covariates as German free text. This function holds the maps that fold that text onto controlled vocabularies **as data**, so extending the package for a spelling it has not seen is one argument rather than a code change: whatever you pass is `utils::modifyList()`-merged, recursively, onto the shipped default.

Usage

```
sdt_participant_dictionary(
  funktion = NULL,
  titel = NULL,
  fachbereich = NULL,
  anrede = NULL
)
```

Arguments

`funktion`, `titel`, `fachbereich`, `anrede`
: Optional named lists merged onto the corresponding shipped slot. Each slot has a `map` (normalised key to level), and, depending on the covariate, `patterns`, `levels`, `score` and `group`. Adding one spelling means passing `funktion = list(map = list("leitender oberarzt" = "consultant"))`.

Details

The four vocabularies are:

- `funktion` — an ordered seniority ladder, `resident < specialist < consultant < chief` (Assistenzarzt, Facharzt, Oberarzt, Chefarzt), scored 1–4. `other` ("Andere") is a real answer that sits off the ladder and scores `NA`; `not_stated` ("Nichts angegeben") is a declared non-answer and is treated as **missing**, not as a sixth grade.
- `titel` — the academic ladder `none < doctorate < habilitation < professor`, scored 0–3. Matching is exact on a punctuation- and case-insensitive key first, then falls back to `patterns`, where every rung is tested and the highest match wins. An empty title box is filled with `none` when `blank_is_none` is `TRUE` — and every filled row is flagged, see `sdt_read_participants()`.
- `fachbereich` — a controlled specialty list plus a surgical / non-surgical `group`. Multi-specialty free text ("Chirurgie, Unfallchirurgie") is collapsed by the stated rule *the primary specialty is the one mentioned first*: the pattern whose match starts earliest in the string wins.
- `anrede` — salutation to sex. This is a **proxy**: it records how a person was addressed on a registration form, not a gender identity.

Value

An S3 object of class `sdt_participant_dictionary`: a list of four slots, with `print()` and `format()` methods.

See Also

`sdt_read_participants()`, which applies it.

Examples

```
d <- sdt_participant_dictionary()
d
d$funktion$score

# One new spelling, no package patch.
d2 <- sdt_participant_dictionary(
  funktion = list(map = list("leitender oberarzt" = "consultant"))
)
d2$funktion$map[["leitender oberarzt"]]
```

`sdt_participant_summary` *Descriptives that state their denominator*

Description

Tabulates the normalised participant covariates without silently dropping or filling anything. Every row carries both denominators and both kinds of uncertainty, so a reader cannot see a share without seeing what it rests on:

Usage

```
sdt_participant_summary(x, fields = NULL, by = NULL)
```

Arguments

`x`	A `sdt_participants` object.
`fields`	Optional character vector of covariates to report. Defaults to every categorical covariate the dictionary normalises.
`by`	Optional column name to report within, typically `"file"` or `"short_course_id"`. `NULL`, the default, pools the whole register.

Details

- `n` — rows at this level, filled rows included.
- `n_filled` — how many of those `n` came from a participant-level fill rather than from an answer somebody wrote down. Read `n - n_filled` as the observed count.
- `n_records` — every row in the group, the denominator of `share_records`.
- `n_observed` — rows where the raw field was actually recorded, taken from the `.missing_` indicator, and the denominator of `share_observed`, which counts observed rows only.
- `n_missing`, `share_missing` — what is not there, in the same row as the shares it is missing from.

This is the honest half of the missingness policy. The imputing half lives in `sdt_course_composition()`, which fills and flags.

Value

A tibble with one row per level, carrying `n`, `n_filled`, `n_observed`, `n_records`, `share_observed`, `share_records`, `n_missing` and `share_missing`.

See Also

`sdt_read_participants()`, `sdt_course_composition()`.

Examples

```
## Not run:
p <- sdt_read_participants("TDSC_course_participant_relations.xlsx")
sdt_participant_summary(p)
sdt_participant_summary(p, fields = "seniority", by = "short_course_id")

## End(Not run)
```

`sdt_patient_card` *Create a patient card*

Description

Constructs a single validated `patient_card` describing one casualty in a mass-casualty scenario. Fields follow the TDSC tabletop patient-card model.

Usage

```
sdt_patient_card(
  id,
  age,
  sex,
  life_points,
  ventilated,
  injury_region,
  triage,
  therapy,
  arrival_order
)
```

Arguments

`id`	Character scalar identifier for the casualty.
`age`	Whole number, patient age in years.
`sex`	One of `"f"`, `"m"`, `"d"`.
`life_points`	Whole number in `1:6`; lower is more critical.
`ventilated`	Logical; whether the casualty requires ventilation.
`injury_region`	Leading injury region; one of `sdt_injury_regions()`.
`triage`	True sighting category; one of `sdt_triage_levels()`.
`therapy`	True therapy concept; one of `sdt_therapy_levels()`.
`arrival_order`	Whole number giving position in the arrival sequence.

Value

An S3 object of class `patient_card`.

Examples

```
sdt_patient_card(
  id = "P01", age = 34, sex = "m", life_points = 4L,
  ventilated = TRUE, injury_region = "thorax",
  triage = "red", therapy = "DCS", arrival_order = 1L
)
```

`sdt_plot_board` *Draw the board, with the casualties standing on it*

Description

The picture the exercise is played on: the nine areas of the published board laid out as in the source's Fig. 1, with every casualty drawn as a point in the area it had reached by a given moment. Scrub `at` across the game and you watch the casualty stream move over the board, which is the one view that makes a decision path legible without reading a table.

Usage

```
sdt_plot_board(run, at = NULL, replication = 1L, interactive = FALSE)
```

Arguments

`run`	A `game_run` from `sdt_simulate()`.
`at`	Time in minutes to draw. Defaults to the end of the game.
`replication`	Which replication to draw.
`interactive`	If `TRUE` and **ggiraph** is installed, draw the casualties as interactive points carrying a tooltip with the patient card. Wrap the result in `ggiraph::girafe()` to get the interactivity; it renders as an ordinary plot otherwise.

Details

A casualty's area at time `at` is the last area it had *entered* by then, which is the same rule `summary.game_run()` uses. That matters at the end of the game: someone still being operated on when the clock stops is in theatre, not back where they came from.

Value

A **ggplot2** object.

See Also

`sdt_plot_flow()` for the same event log as a Gantt chart.

Examples

```
run <- sdt_simulate(sdt_reference_scenario(), replications = 1,
                    seed = 1, verbose = FALSE)
sdt_plot_board(run)

# Halfway through the game.
sdt_plot_board(run, at = sdt_reference_horizon() / 2)
```

`sdt_plot_end_state` *Plot the end-of-game distribution against the published reference*

Description

Draws the observed proportion per reporting location as bars, with the published reference proportion overlaid as a point and the gap drawn as a segment between them. It is the honest version of a calibration claim: the gap is visible rather than asserted.

Usage

```
sdt_plot_end_state(run, version = c("2.0", "1.0"))
```

Arguments

`run`	A `game_run` from `sdt_simulate()`.
`version`	Which evaluated version of the exercise to compare against, `"2.0"` or `"1.0"`.

Value

A **ggplot2** object.

See Also

`summary.game_run()`, which produces the underlying tibble.

Examples

```
run <- sdt_simulate(sdt_reference_scenario(), replications = 4,
                    seed = 1, verbose = FALSE)
sdt_plot_end_state(run)
sdt_plot_end_state(run, version = "1.0")
```

`sdt_plot_fidelity` *Draw a fidelity table*

Description

A dot-and-range chart of the dispersion ratios: one row per statistic, the ratio on the x axis, the passing band shaded, and the bootstrap interval drawn as a range. Controls carry their own marker because they are read backwards from the rest – see the "Negative controls" section of `sdt_fidelity()`.

Usage

```
sdt_plot_fidelity(x)
```

Arguments

x	A fidelity table from `sdt_fidelity()`.

Details

A ratio of `Inf`, which is what a simulator that produces no between-group spread at all earns, is drawn as an arrow pinned to the right-hand edge and labelled, rather than being dropped from the plot or silently rescaling it.

Value

A **ggplot2** object.

See Also

`sdt_fidelity()`.

Examples

```
set.seed(1)
cards <- sprintf("c%02d", 1:6)
obs <- expand.grid(group_id = sprintf("g%02d", 1:10), card = cards,
                   stringsAsFactors = FALSE)
team <- stats::setNames(stats::runif(10, 0.1, 0.9), sprintf("g%02d", 1:10))
obs$surgery <- stats::runif(nrow(obs)) < team[obs$group_id]
sim <- expand.grid(sim_group = sprintf("s%02d", 1:10), card = cards,
                   stringsAsFactors = FALSE)
sim$surgery <- stats::runif(nrow(sim)) < mean(team)
sdt_plot_fidelity(sdt_fidelity(sim, obs, n_boot = 50, seed = 1))
```

sdt_plot_flow *Plot patient flow through the board*

Description

A Gantt chart of one replication: each casualty is a row, each board area a coloured bar spanning the time it was there. This is the picture of the event log, and the fastest way to see where a board is congested.

Usage

```
sdt_plot_flow(run, replication = 1L, max_casualties = 40L)
```

Arguments

run	A game_run from sdt_simulate().
replication	Which replication to draw.
max_casualties	Cap on the number of casualties shown, to keep the chart readable. The earliest arrivals are kept.

Value

A **ggplot2** object.

Examples

```
co <- sdt_synthetic_cohort(15, seed = 1)
run <- sdt_simulate(sdt_scenario(co), replications = 1, verbose = FALSE)
sdt_plot_flow(run)
```

sdt_plot_sequences *Plot decision-path sequences*

Description

Two standard sequence-analysis views, both drawn with **ggplot2** rather than **TraMineR**'s base-graphics plotting, so they compose and theme like every other plot in the package.

Usage

```
sdt_plot_sequences(
  run,
  type = c("index", "distribution"),
  replication = NULL,
  clusters = NULL
)
```

Arguments

run	A game_run from sdt_simulate().
type	"index" or "distribution".
replication	Optional replication to restrict to.
clusters	Optional cluster assignment from sdt_seq_cluster(). When supplied, the plot is facetted by cluster.

Details

"index" draws one row per casualty, coloured by the area occupied at each position: the shape of individual pathways. "distribution" draws the share of casualties in each area at each position: the shape of the cohort.

Value

A **ggplot2** object.

See Also

sdt_seq_cluster(), sdt_decision_paths().

Examples

```
co <- sdt_synthetic_cohort(n = 20, seed = 1)
run <- sdt_simulate(sdt_scenario(co), replications = 2, seed = 1,
                    verbose = FALSE)
sdt_plot_sequences(run)
sdt_plot_sequences(run, type = "distribution")
```

---

sdt_plot_triage_mix *Plot the triage mix of a scenario*

Description

Plot the triage mix of a scenario

Usage

```
sdt_plot_triage_mix(scenario)
```

Arguments

scenario	A scenario object.

Value

A **ggplot2** object.

Examples

```
sdt_plot_triage_mix(sdt_scenario(sdt_synthetic_cohort(20, seed = 1)))
```

`sdt_plot_utilisation` *Plot resource utilisation from a game run*

Description

Plot resource utilisation from a game run

Usage

```
sdt_plot_utilisation(run)
```

Arguments

`run`	A `game_run` from `sdt_simulate()`.

Value

A **ggplot2** object.

Examples

```
co <- sdt_synthetic_cohort(20, seed = 1)
sc <- sdt_scenario(co, resources = list(or = 2))
run <- sdt_simulate(sc, replications = 1, verbose = FALSE)
sdt_plot_utilisation(run)
```

`sdt_predictions` *Held-out predictions from a fitted model*

Description

The predictions `sdt_fit()` made on the test half of its own split, with the truth alongside. This is what the metrics were computed from.

Usage

```
sdt_predictions(model)
```

Arguments

`model`	An `sdt_model` from `sdt_fit()`.

Value

A `tibble::tibble()` with `.pred_class`, the per-class probability columns, and `truth`.

Examples

```
# See ?sdt_fit for a runnable example.
sdt_metric_set
```

`sdt_profile` *Dependence profiles for a fitted model*

Description

How the model's predicted probability moves as one predictor changes, with the others held at their observed values. Partial dependence averages over the data; accumulated local effects are more trustworthy when predictors are correlated, which on a casualty card they are.

Usage

```
sdt_profile(
  x,
  variables = NULL,
  type = c("partial", "accumulated"),
  data = NULL
)
```

Arguments

`x`	An `sdt_model` from `sdt_fit()`, or an explainer from `sdt_explain()`.
`variables`	Predictors to profile. Defaults to all of them.
`type`	`"partial"` for a PDP, `"accumulated"` for ALE.
`data`	Optional data to explain on, passed to `sdt_explain()`.

Value

A **ggplot2** object.

See Also

`sdt_importance()`, `sdt_shap()`.

Examples

```
# See ?sdt_importance for a runnable example.
sdt_engines()
```

`sdt_read_gold_standard` *Read the gold-standard triage codebook*

Description

Reads the exercise deck's answer key: one row per patient card, giving the sighting category the card was written to carry, plus the card's recorded sex, age, clinical findings and vital signs.

Usage

```
sdt_read_gold_standard(path, sheet = NULL, verbose = TRUE)
```

Arguments

`path`	Path to the codebook workbook (`.xlsx`). Read-only; nothing is written to it or beside it.
`sheet`	Sheet to read. `NULL`, the default, takes the sheet whose name begins with `"Verteilung"`, or the only sheet if the workbook has one.
`verbose`	Report the parsed deck, the guard drops and the cross-checks while reading.

Details

This is the ground truth the course exports do not contain. An export records the calls teams made, so any triage number computed from it alone is inter-team agreement; the field that looks like an answer key is a round-robin placeholder. Pair this reader with `sdt_apply_gold_standard()` and `sdt_triage_accuracy()` to score calls against the deck rather than against each other.

Value

An `sdt_gold_standard` object: a `tibble::tibble()` subclass with one row per card and the columns `card` (`"01"`–`"46"`), `triage`, `sex`, `age`, `findings`, `vitals` and `sk_raw`. The `meta` attribute carries the path, the sheet, the guard drops, the legend cross-check and the working sheet cross-check.

The sheet is not a clean rectangle

Codebooks acquire duplicate test rows and legend blocks below the data, and a legend written into shifted columns parses as data if it is not caught. Three guards remove them, all by rule rather than by row position, since row positions are the first thing an edited workbook changes:

- `blank` — every contract column is empty.
- `non_numeric_card` — the card id is not a number, which is what catches both duplicate test rows carrying a letter id and the shifted legend.
- `card_out_of_range` — the card id is a number outside the deck.

How many rows each guard removed is reported when `verbose = TRUE` and kept in the `meta` attribute, so a silent change in the workbook shows up as a changed count rather than as a changed result.

Cross-checks rather than a preference

Two independent statements of the same counts are read back and compared with the parse: the sheet's own SK I/SK II/SK III legend, whose totals include the duplicate rows the guards removed, and any earlier working sheet in the same workbook. Disagreements are reported, never resolved by quietly preferring one source.

See Also

sdt_apply_gold_standard() to write it into a cohort, sdt_triage_accuracy() to score calls against it.

Examples

```
## Not run:
gold <- sdt_read_gold_standard("Patienten Codebook.xlsx")
gold
attr(gold, "meta")$guards

## End(Not run)
```

---

sdt_read_participants *Read the TDSC course and group metadata register*

Description

Reads the participant workbook, binds its per-course sheets, applies sdt_participant_dictionary() and returns one row per participant record with the raw and the normalised columns side by side.

Usage

```
sdt_read_participants(
  path,
  dictionary = sdt_participant_dictionary(),
  sheets = NULL,
  verbose = TRUE
)
```

Arguments

path	Path to the participant workbook (.xlsx). Read-only; nothing is written to it or beside it.
dictionary	A sdt_participant_dictionary.
sheets	Optional character vector of per-course sheet names to restrict to. NULL, the default, reads every sheet named in the workbook's Course_Index.
verbose	Report row counts, neutralised leaks and per-covariate missingness while reading.

Value

A sdt_participants object: a tibble::tibble() subclass, one row per participant record, with attributes course_index (one row per course, including the empty ones), dictionary and meta.

What this data can and cannot do

The register links a person to a **course**. It carries no group, team or "Gruppe" column, so a participant cannot be placed in group A rather than group B. Every group inside one course therefore shares the same composition row. In the behaviour model y ~ card + (1 | block) + (1 | group) these covariates are block-level: they can compete with the **session** random effect, and can never explain the **team** random effect, which has no composition variance to spend. Treat any composition analysis as a between-course regression on as many points as there are courses.

Missingness

Courses with no linked participants are kept, not dropped: the course_index attribute carries every course in the workbook with its n_participants and has_participants, so a caller sees the empty courses rather than inferring them from an absence.

Rows the extractor wrote a diagnostic into instead of an answer are detected and neutralised to NA before anything is normalised, and a row with nothing but such a diagnostic is a course placeholder rather than a person: it is removed from the participant rows and counted in course_index$n_placeholder_rows.

Two logical column families record what happened to every raw field:

- .missing_<field> — the information is not available. For funktion this includes the declared non-answer "Nichts angegeben", which is missing information rather than a grade; for weiterbildungsjahre it includes unparseable free text such as ",".
- .imputed_<field> — the normalised value was filled rather than observed. Only titel has a participant-level fill rule by default (an empty title box becomes none, see sdt_participant_dictionary()); the rest are filled, if at all, at course level by sdt_course_composition(). An imputed value is never indistinguishable from an observed one.

See Also

sdt_course_composition() to aggregate to course level, sdt_participant_summary() for descriptives that state their denominators.

Examples

```
## Not run:
p <- sdt_read_participants("TDSC_course_participant_relations.xlsx")
p
attr(p, "course_index")
sdt_course_composition(p)

## End(Not run)
```

`sdt_reference` *Reference distributions for the mass-casualty domain*

Description

Returns the marginal distributions `sandtablR` calibrates against, read from the file generated by `data-raw/tdsc_reference.R` and cached for the session.

Usage

```
sdt_reference(
  which = c("all", "published", "design", "defaults", "meta"),
  version = c("2.0", "1.0")
)
```

Arguments

`which`	Which part to return: `"all"`, `"published"`, `"design"`, `"defaults"` or `"meta"`.
`version`	Which evaluated version of the exercise the published marginals should come from: `"2.0"` (the current exercise, the default) or `"1.0"` (its predecessor). Ignored for parts other than `"published"`, which are common to both.

Details

The list separates three things that are easy to conflate:

- `published` — marginals attributed to the TDSC tabletop evaluation (Achatz et al. 2023, doi:10.1007/s00068022022171). These are what the package reports and tests as "reference". Both evaluated versions of the exercise are carried; see `version`.
- `design` — the exercise as specified in that paper's Methods: duration, card count, team roles, and the nine-area board. Published, but structural rather than distributional.
- `defaults` — modelling choices made by `sandtablR` where the source does not pin a value (service times, routing probabilities, and so on). They are **not** attributed to the paper.

Calibration tests use `published` only. Mixing the two would test the package against its own assumptions.

Value

A named list.

See Also

sdt_vocabularies for the levels these marginals range over.

Examples

```
sdt_reference("published")$triage_mix
sdt_reference("published", version = "1.0")$triage_mix

# What the two versions did differently, side by side.
rbind(
  `2.0` = sdt_reference("published", "2.0")$end_state,
  `1.0` = sdt_reference("published", "1.0")$end_state
)

sdt_reference("design")$duration
sdt_reference("meta")$doi
```

---

sdt_reference_horizon *The horizon the engine is calibrated on*

Description

The horizon the engine is calibrated on

Usage

```
sdt_reference_horizon()
```

Value

A single number: the game length in minutes used for calibration.

Examples

```
sdt_reference_horizon()
```

---

sdt_reference_scenario *The scenario the engine is calibrated on*

Description

Builds the board configuration that `sdt_simulate()`'s defaults were tuned against: a 40-casualty cohort on the resource capacities recorded in `sdt_reference("defaults")$calibration`. Running it over the calibrated horizon reproduces the published end-of-game distribution.

Usage

```
sdt_reference_scenario(seed = NULL)
```

Arguments

seed	Optional integer seed for the cohort. Defaults to the calibration seed.

Details

Use it as a known-good starting point, and as the fixture the calibration test checks, so that "the model still reproduces the reference" is a claim someone can re-run rather than take on trust.

Value

A scenario object.

See Also

sdt_simulate(), summary.game_run().

Examples

```
sc <- sdt_reference_scenario()
sc
run <- sdt_simulate(sc, replications = 2, seed = 1, verbose = FALSE)
summary(run, reference = TRUE)
```

---

sdt_roc *ROC curve for a fitted model*

Description

Draws the receiver operating characteristic of an sdt_fit() model on its held-out split. With more than two classes, one curve per class is drawn one-versus-all, which is what roc_auc's Hand-Till estimator summarises.

Usage

```
sdt_roc(model)
```

Arguments

model	An sdt_model from sdt_fit().

Value

A **ggplot2** object.

Examples

```
# See ?sdt_fit for a runnable example.
sdt_metric_set
```

sdt_run_app *Launch the sandtablR workbench*

Description

Starts the interactive **shiny** workbench: build a scenario, run the board, annotate decisions, train and compare models, explain them, and analyse decision paths.

Usage

```
sdt_run_app(...)
```

Arguments

`...`	Passed to `shiny::runApp()`.

Details

The app is assembled by `sdt_app()` rather than loaded from a directory, so the modules are ordinary package functions: checked, linted, covered and testable like everything else.

Value

Invisible `NULL`. Launches a Shiny application.

See Also

`sdt_app_ui()`, `sdt_app_server()`.

Examples

```
if (interactive()) {
  sdt_run_app()
}
```

sdt_run_behaviour *Turn a discrete-event game run into a (group, card) behaviour frame*

Description

Reshapes the event log of a `sdt_simulate()` run into the same one-row-per (group, card) frame that `sdt_fidelity()` scores, so the discrete-event engine can be held to exactly the same statistics as a fitted behaviour model. Each replication is one group and each casualty `id` is one card, which is what makes the comparison legitimate: groups within a real course session play the same fixed deck, and so do replications.

Usage

```
sdt_run_behaviour(run, group = "group_id", card = "card")
```

Arguments

run	A game_run from sdt_simulate().
group	Name for the group column in the result.
card	Name for the card column in the result.

Value

A tibble with one row per (replication, casualty), the two key columns named by group and card, the cohort's triage and therapy, the casualty's end_area, and one logical column per behaviour described above.

What each column means, and what it cannot mean

- red, yellow, green, ETC, DCS, TASC and ventilated are read off the cohort. The engine *routes on* them, it does not choose them, so they are identical in every replication and their between-group SD is exactly zero. That is not a bug in this function; it is the engine's structure made visible, and sdt_fidelity() will report it as an infinite dispersion ratio.
- surgery, diagnostics, icu, ward, recovery, green_area record whether the casualty ever entered that board area. These *can* vary between replications, through queueing and the horizon.
- died is end_area == "morgue", and not_seen is a casualty the game ended before. The engine has no branch to the morgue, so died is always FALSE.

The observed counterparts, for a course export, are therapy == "TASC" and so on for the therapy concepts, called_triage == "red" for the sighting categories, therapy != "none" for surgery, and the reported end location for icu. There is deliberately no eFAST or CT column: the engine has a single radiology area and no modality, so inventing one would be inventing a comparison.

See Also

sdt_fidelity(), summary.game_run().

Examples

```
co <- sdt_synthetic_cohort(n = 12, seed = 1)
run <- sdt_simulate(sdt_scenario(co), replications = 3, seed = 1,
                    verbose = FALSE)
sdt_run_behaviour(run)
```

sdt_scenario *Build a mass-casualty scenario*

Description

Assembles an ordered casualty cohort and the board resource capacities into a validated `scenario` object, the central input to the simulation engine.

Usage

```
sdt_scenario(cohort, resources = list(), name = NULL, seed = NULL)
```

Arguments

`cohort`	A data frame / tibble of casualties, one row per patient, with at least the columns produced by `sdt_patient_card()` (`id`, `age`, `sex`, `life_points`, `ventilated`, `injury_region`, `triage`, `therapy`, `arrival_order`).
`resources`	Named list of integer capacities for board areas, e.g. `list(triage = 1, or = 2, icu_imc = 6, ed_red = 2, ed_yellow = 4, ward = 20)`. Missing areas default to 1.
`name`	Optional scenario name.
`seed`	Optional integer seed recorded in metadata for reproducibility.

Value

An S3 object of class `scenario`.

Examples

```
co <- sdt_synthetic_cohort(n = 20, seed = 1)
sdt_scenario(co, resources = list(or = 2, icu_imc = 6), name = "concert")
```

sdt_seq_cluster *Cluster decision paths*

Description

Groups casualties by how similar their board pathways are: an optimal matching dissimilarity between state sequences, then partitioning around medoids. What comes out is a small number of characteristic ways through the board, which is usually more useful than a per-casualty path listing.

Usage

```
sdt_seq_cluster(run, k = 3L, method = "OM", indel = 1, sm = "CONSTANT")
```

Arguments

run	A game_run from sdt_simulate().
k	Number of clusters.
method	Dissimilarity passed to TraMineR::seqdist(). "OM" is optimal matching.
indel	Insertion/deletion cost for optimal matching.
sm	Substitution-cost matrix or method. "CONSTANT" applies a flat cost, the right default when no substitution is more plausible than another.

Value

A tibble::tibble() with one row per sequence: replication, id, cluster, and medoid, which flags the sequence representing its cluster.

See Also

sdt_seqdef(), sdt_plot_sequences().

Examples

```
co <- sdt_synthetic_cohort(n = 30, seed = 1)
run <- sdt_simulate(sdt_scenario(co), replications = 2, seed = 1,
                    verbose = FALSE)
if (requireNamespace("TraMineR", quietly = TRUE) &&
    requireNamespace("WeightedCluster", quietly = TRUE)) {
  sdt_seq_cluster(run, k = 3)
}
```

---

sdt_seqdef *Build a TraMineR state-sequence object*

Description

Converts the decision paths of a game run into a TraMineR::seqdef() object over the board-area alphabet, ready for sequence analysis. The alphabet is fixed to sdt_board_areas(), so sequences from different runs stay comparable even when a run never visits some area.

Usage

```
sdt_seqdef(run, replication = NULL)
```

Arguments

run	A game_run from sdt_simulate().
replication	Optional replication to restrict to. NULL keeps all, which is what you want when clustering across replications.

Value

A `stslist` object from **TraMineR**, carrying `replication` and `casualty` attributes recording where each row came from.

See Also

`sdt_seq_cluster()`, `sdt_plot_sequences()`.

Examples

```
co <- sdt_synthetic_cohort(n = 12, seed = 1)
run <- sdt_simulate(sdt_scenario(co), replications = 2, seed = 1,
                    verbose = FALSE)
if (requireNamespace("TraMineR", quietly = TRUE)) {
  seqs <- sdt_seqdef(run)
  dim(seqs)
}
```

---

sdt_shap *Attribution for one casualty*

Description

Breaks the model's prediction for a single casualty into per-predictor contributions, using SHAP values. This answers "why this casualty?", as opposed to importance, which answers "what does this model use in general?".

Usage

```
sdt_shap(x, casualty, data = NULL, n_permutations = 10L)
```

Arguments

`x`	An `sdt_model` from `sdt_fit()`, or an explainer from `sdt_explain()`.
`casualty`	A one-row data frame, or the `id` of a casualty in `data`.
`data`	Optional data to explain on, passed to `sdt_explain()`. Also where `casualty` is looked up when given as an id.
`n_permutations`	Number of sampling rounds for the SHAP estimate.

Value

A **ggplot2** object.

See Also

`sdt_importance()`, `sdt_profile()`.

Examples

```
# See ?sdt_importance for a runnable example.
sdt_engines()
```

`sdt_simulate` *Run a mass-casualty patient-flow simulation*

Description

Executes a discrete-event simulation of a scenario with **simmer**, routing each casualty through board-area resources according to its sighting category and therapy concept, and returns per-area event records, per-casualty flow records and resource utilisation.

Usage

```
sdt_simulate(
  scenario,
  replications = 1L,
  service_times = list(),
  horizon = 270,
  parallel = FALSE,
  workers = NULL,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

`scenario`	A `scenario` object from `sdt_scenario()`.
`replications`	Number of independent replications (positive whole number). The replication index is carried in every returned table.
`service_times`	Named list of mean service times in minutes per board area, used as exponential rates. Missing areas fall back to the reference defaults.
`horizon`	Length of the game in minutes. Casualties still in the system when the clock stops are recorded where they stood, which is what makes the end-of-game distribution meaningful. `Inf` runs every casualty to completion.
`parallel`	If `TRUE`, run replications concurrently. Defaults to `FALSE`. The backend is set up for the duration of the call and the caller's plan is restored on exit.
`workers`	Number of parallel workers, or `NULL` to choose one: a core short of the machine, capped at 8.
`seed`	Optional integer seed, applied with `withr::local_seed()`.
`verbose`	If `TRUE`, report progress via **cli**.

Details

Theatre time scales with the therapy concept: abbreviated surgical care is quickest, damage-control surgery the reference, early total care the longest. The multipliers live in `sdt_reference("defaults")$or_multiplier`.

Value

An S3 object of class `game_run` with elements:

`events` One row per casualty per board area entered: `replication`, `id`, `step`, `area`, `start`, `end`, `duration`, `service_time`. A state runs from entering an area to entering the next, and the last runs to the horizon, so `duration` is time spent standing in that area. `service_time` is the treatment time simmer actually recorded, and is `NA` where the casualty was still being treated at the horizon or was only being held. The two differ whenever a casualty is finished but waiting for a bed downstream.

`arrivals` One row per casualty per replication, carrying the card fields and total time in system.

`resources` Resource utilisation over time.

See Also

`summary.game_run()` for the end-of-game distribution, and `sdt_decision_paths()` to turn the event log into state sequences.

Examples

```
co <- sdt_synthetic_cohort(n = 20, seed = 1)
sc <- sdt_scenario(co, resources = list(or = 2, icu_imc = 6))
run <- sdt_simulate(sc, replications = 2, seed = 1, verbose = FALSE)
run
head(run$events)
```

---

sdt_simulate_groups *Simulate new groups from a fitted behaviour model*

Description

Draws teams that were never in the data. For each simulated group and each modelled outcome the team's own random effect is redrawn from `N(0, sd_group)`, a session effect is redrawn from `N(0, sd_block)`, and the outcome is then drawn per card from the fitted card effects:

Usage

```
sdt_simulate_groups(
  model,
  n_groups = 20L,
  cards = NULL,
  seed = NULL,
  new_teams = TRUE,
```

```
  nsim = 1L,
  composition = NULL
)
```

Arguments

model	An sdt_behaviour_model from sdt_fit_behaviour().
n_groups	Number of simulated groups per replicate.
cards	Cards to deal to each simulated group. NULL uses the fitted frame's cards; anything passed must be a subset of them.
seed	Optional integer seed, applied with withr::local_seed().
new_teams	If TRUE, redraw the team and session effects. If FALSE, reuse the fitted ones.
nsim	Number of replicates. Replicates are stacked and carry a .replicate column; sim_group is unique across the whole result, so grouping by it never merges two different simulated teams.
composition	Optional block-level covariate values to condition on: a data frame with one row per course (a slice of sdt_course_composition() will do), or a named list or vector for a single course. NULL simulates marginally, exactly as a model fitted without covariates does. Requires a model fitted with block_covariates.

Details

```
p = plogis(card effect + team effect + session effect)
```

This is deliberately not stats::simulate() on the fitted model with the fitted grouping structure, which would put the training teams back on the table. Every simulated group here is a new team playing in a new session.

Value

An S3 object of class sdt_simulated_groups, a tibble::tibble() with .replicate, sim_group, card and one logical column per modelled outcome. Carries the fitted model, the call, and a draws attribute holding the random effects used for each simulated group.

Which random effects are redrawn

With new_teams = TRUE (the default) both the group and the block effect are drawn fresh and independently for every simulated group, so a simulated group is "a new team in a new session" and its total random effect has variance sd_group^2 + sd_block^2 – the same marginal spread a real, previously unseen group has.

With new_teams = FALSE the fitted conditional modes are reused: simulated groups are the observed teams, with their own session effects, playing the same cards again. On a model fitted with block covariates their own course's composition goes back into the linear predictor as well, because the fitted session modes are residuals with that part already removed. That is a posterior-predictive check of the fit, not a generator of new teams. It takes the first n_groups fitted teams in frame order rather than a random sample, so it is reproducible without a seed; pass n_groups = model$meta$n_groups to use them all, which is what a like-for-like check against the observed frame wants. The draws attribute records which observed team each simulated one came from.

Simulating a course of a given composition

`composition` takes one row – or several – of block-level covariate values on their natural scale, so a caller can ask for "a course whose participants look like this". Each row is a course: `n_groups` teams are drawn inside **each** of them, and they share one drawn session effect, because groups in one course played in one session. The result carries a `.composition` column naming the row each simulated team belongs to.

Conditioning changes two things. The covariates' fixed contribution is added to the linear predictor, and the session effect is drawn from the **residual** block SD – what is left after the covariates – rather than from the total. With `composition = NULL` neither happens: the covariates sit at their fitted mean and the session effect is drawn from the total block SD, which is the marginal spread of an unseen course and exactly what a model fitted without covariates produces from the same seed.

A composition value outside the range the fitted courses covered is warned about rather than refused. It is extrapolation from a between-course slope estimated on as many points as there were courses, and it does not travel.

`composition` cannot be combined with `new_teams = FALSE`: reusing the fitted teams reuses the courses they actually played in, so asking for a hypothetical composition at the same time asks for two contradictory things.

Each row is labelled by the block key when the caller passes one, so a slice of `sdt_course_composition()` keeps its file name and the reader can see which real course a hypothetical one was copied from. That label is a course identifier, and `sdt_export_simulation()` will refuse to write it into a pseudonymised bundle – rename the key column to something neutral before simulating if the result is going to be exported.

What is not reproduced

Each outcome was fitted separately, so a simulated team's tendencies are drawn independently across outcomes. Real teams that image more also operate more; that correlation is absent here. Card-to-card dependence within a casualty (an eFAST leading to a CT) is likewise absent.

Every simulated group is also dealt every card, while an observed group can have missing outcome values – a casualty its team never triaged. Those groups differ in *which* cards contribute to their rate, and that is spread the simulation has no way to produce. On the validation corpus `red` is missing in 105 of 3410 cells and its dispersion ratio reads 1.20 over all 110 groups but 0.97 over the 64 groups with a complete card set. `model$meta$sds$n_missing` counts those cells, so a fidelity comparison can restrict itself to the groups that have none.

See Also

`sdt_fit_behaviour()`.

Examples

```
set.seed(1)
d <- expand.grid(
  card = sprintf("c%02d", seq_len(8)),
  group_id = sprintf("g%02d", seq_len(12)),
  stringsAsFactors = FALSE
)
d$session_id <- ifelse(d$group_id < "g07", "s1", "s2")
```

```
b <- stats::rnorm(12, sd = 1.2)
p <- stats::plogis(-0.5 + b[match(d$group_id, unique(d$group_id))])
d$did_efast <- stats::runif(nrow(d)) < p
if (requireNamespace("lme4", quietly = TRUE)) {
  fit <- sdt_fit_behaviour(d, outcomes = "efast", verbose = FALSE)
  sdt_simulate_groups(fit, n_groups = 5, seed = 1)
}
```

`sdt_simulate_triage` *Simulate triage calls from a fitted error model*

Description

Draws teams that were never in the data and, for each simulated team and each card, a **called** sighting category given the card's **true** one. It is the generative counterpart of `sdt_fit_triage_error()`: where that fits the rows of the confusion matrix, this replays them.

Usage

```
sdt_simulate_triage(
  model,
  n_groups = 20L,
  cards = NULL,
  seed = NULL,
  new_teams = TRUE,
  nsim = 1L,
  team_effects = TRUE
)
```

Arguments

`model`	An `sdt_triage_error_model` from `sdt_fit_triage_error()`.
`n_groups`	Number of simulated teams per replicate.
`cards`	Cards to deal to each team. `NULL` uses every card in the fitted frame; anything passed must be a subset of them.
`seed`	Optional integer seed, applied with `withr::local_seed()`.
`new_teams`	If `TRUE` (default), redraw the team and session effects, so every simulated team is new. If `FALSE`, reuse the fitted conditional modes of the first `n_groups` fitted teams, which is a posterior-predictive check rather than a generator.
`nsim`	Number of replicates. Replicates are stacked and carry a `.replicate` column; `sim_group` is unique across the whole result.
`team_effects`	If `FALSE`, simulate from the null fits – the same cells with `(1 | group)` removed. This is the counterfactual that shows what the team term is doing, and it is a different model rather than the same one with the draw switched off.

Details

Each simulated team gets its own random effect per cell, drawn from `N(0, sd_group)`, plus a session effect from `N(0, sd_block)`. For a card of true category `k` the call is then a two-step draw: correct with probability `plogis(card effect + team effect + session effect)` from the `hit` cell, and otherwise one of the two wrong categories from the `split` cell. That is the confusion-matrix row, redrawn for a new team.

Value

An S3 object of class `sdt_simulated_triage`, a tibble with `.replicate`, `sim_group`, `card`, `triage`, `called_triage`, and the `hit`, `over` and `under` logicals. Carries the `model`, the `call`, a `draws` attribute holding the random effects used, and a `gaps` attribute counting the cells left `NA`.

What comes back as `NA`

A cell that was never fitted is never guessed. If a category's `hit` cell is missing, every card of that category returns `NA`; if only its `split` cell is missing, the correct calls stand and the errors return `NA`. A card with no coefficient in a cell – a card the split part never saw, because no team miscalled it – returns `NA` for the same reason. Every one of these is counted in the result's `gaps` attribute and warned about once.

What is not reproduced

The two parts are drawn independently, so a simulated team's accuracy and its error direction are uncorrelated. That is the one parameter the decomposition gives up, and `joint = TRUE` in `sdt_fit_triage_error()` measures it. Card-to-card dependence within a team – a team that has just over-triaged three greens in a row – is absent as well: cards are drawn independently given the team.

See Also

`sdt_fit_triage_error()`, `sdt_error_fidelity()`.

Examples

```
set.seed(1)
cards <- sprintf("c%02d", seq_len(12))
truth <- rep(c("red", "yellow", "green"), each = 4)
d <- expand.grid(card = cards, group_id = sprintf("g%02d", seq_len(20)),
                 stringsAsFactors = FALSE)
d$session_id <- ifelse(d$group_id < "g11", "s1", "s2")
d$triage <- truth[match(d$card, cards)]
b <- stats::rnorm(20, sd = 1)
p <- stats::plogis(1 + b[match(d$group_id, unique(d$group_id))])
hit <- stats::runif(nrow(d)) < p
wrong <- vapply(seq_len(nrow(d)), function(i) {
  sample(setdiff(sdt_triage_levels(), d$triage[[i]]), 1)
}, character(1))
d$called_triage <- ifelse(hit, d$triage, wrong)
if (requireNamespace("lme4", quietly = TRUE)) {
```

```
  fit <- sdt_fit_triage_error(d, verbose = FALSE)
  sdt_simulate_triage(fit, n_groups = 5, seed = 1)
}
```

`sdt_synthesis_utility` *Score a synthetic replica against its seed cohort*

Description

Reports how faithfully a synthetic cohort reproduces its seed, per variable, so a replica can be judged before it is released. Categorical columns are compared by total variation distance between their proportion tables; numeric columns by standardised difference in means and by the Kolmogorov-Smirnov statistic.

Usage

```
sdt_synthesis_utility(original, synthetic)
```

Arguments

`original`	The real seed cohort.
`synthetic`	A replica, e.g. from `sdt_synthesise()`.

Value

A `tibble::tibble()` with one row per shared column and statistic: `variable`, `type`, `statistic`, `value`, and `flag`, which is `TRUE` where the divergence exceeds a conventional 0.1 and the column deserves a look. Returns a 0-row tibble of the same shape when nothing is shared.

Examples

```
seed_cohort <- sdt_synthetic_cohort(n = 80, seed = 1)
replica <- sdt_synthetic_cohort(n = 80, seed = 2)
sdt_synthesis_utility(seed_cohort, replica)
```

`sdt_synthesise` *Synthesise a releasable replica of a real cohort*

Description

Fits sequential CART models to a real casualty cohort with `synthpop::syn()` and draws a synthetic replica from them. Use this when the real data cannot be shared but its multivariate structure should survive into something that can be.

Usage

```
sdt_synthesise(data, n = NULL, method = "cart", seed = NULL, verbose = FALSE)
```

Arguments

data	A real cohort, or any data frame carrying the cohort columns.
n	Number of synthetic rows to draw. Defaults to nrow(data).
method	**synthpop** method, passed to synthpop::syn(). "cart" is the default and handles mixed types without distributional assumptions.
seed	Optional integer seed, applied with withr::local_seed().
verbose	If TRUE, let **synthpop** report its progress.

Details

Unlike sdt_synthetic_cohort(), which generates from marginals alone, this reproduces the joint structure of the seed data, including relationships nobody thought to specify. That is the point, and also the risk: a synthetic record that lands too close to a real one is a disclosure. Score the replica with sdt_synthesis_utility() before releasing it.

Value

A cohort object of n rows. Identifier and ordering columns are regenerated rather than synthesised, because carrying them over would leak the seed data's row identities.

See Also

sdt_synthesis_utility() to score the replica against its seed.

Examples

```
seed_cohort <- sdt_synthetic_cohort(n = 60, seed = 1)
if (requireNamespace("synthpop", quietly = TRUE)) {
  replica <- sdt_synthesise(seed_cohort, n = 30, seed = 2)
  replica
}
```

---

sdt_synthetic_cohort *Generate a synthetic mass-casualty cohort*

Description

Produces a reproducible synthetic cohort, one row per casualty, with the fields expected by sdt_scenario(). Marginals default to the published reference distributions returned by sdt_reference(), and can be overridden.

Usage

```
sdt_synthetic_cohort(
  n = 20L,
  triage_prob = NULL,
  seed = NULL,
  backend = c("base", "simstudy")
)
```

Arguments

n	Number of casualties to generate (positive whole number).
triage_prob	Named numeric vector of sighting-category probabilities. Defaults to the published triage mix. Need not sum to 1; it is normalised.
seed	Optional integer seed. Applied with withr::local_seed(), so the caller's random stream is restored on exit.
backend	Either "base" (always available, frozen behaviour) or "simstudy", which draws correlated age, injury severity and life points through a **simstudy** definition table. If **simstudy** is not installed, "simstudy" warns and falls back to "base".

Value

A cohort object (a tibble::tibble() subclass) with columns id, age, sex, life_points, ventilated, injury_region, triage, therapy, arrival_order. The "simstudy" backend additionally carries iss, an injury-severity proxy.

See Also

sdt_synthesise() to synthesise from a real seed cohort instead.

Examples

```
sdt_synthetic_cohort(n = 10, seed = 42)

# Override the mix to stress the red pathway.
sdt_synthetic_cohort(
  n = 10, triage_prob = c(red = 0.8, yellow = 0.2), seed = 1
)
```

---

sdt_transformer *A tabular transformer specification*

Description

A small FT-Transformer-shaped model for the workbench: one self-attention block over per-feature embeddings, mean-pooled to a softmax. Registered as the "transformer" engine in sdt_engines() and graded "closed", because it decides in a representation it learned rather than in the columns it was given, so a per-column attribution describes something it no longer reasons in.

Usage

```
sdt_transformer(d_model = 16L, num_heads = 2L, epochs = 30L)
```

Arguments

d_model	Embedding width per feature.
num_heads	Attention heads.
epochs	Training epochs.

Details

On a corpus whose effective sample size is the number of distinct cards, this is expected to match rather than beat a logistic regression, and to say so. It becomes the right tool when the training set is expanded with simulated cohorts.

Value

A **parsnip** model specification.

Examples

```
## Not run:
sdt_transformer(d_model = 16, num_heads = 2)

## End(Not run)
```

---

sdt_triage_accuracy *Score triage calls against the truth*

Description

Hit rate per sighting category and overall, with the confusion matrix, and with the two directions of error kept apart. This is the function that only becomes answerable once a gold standard exists: without one, the same arithmetic measures inter-team agreement.

Usage

```
sdt_triage_accuracy(x, truth = "triage", estimate = "called_triage", by = NULL)
```

Arguments

x	A data frame carrying both labels.
truth	Column holding the true category. With a cohort that has been through sdt_apply_gold_standard(), this is "triage".
estimate	Column holding the called category.
by	Optional character vector of columns to split by, for example "group" or c("file", "group"). NULL, the default, pools everything.

Value

An sdt_triage_accuracy object: a tibble::tibble() subclass with the by columns, then category (each triage level plus "overall"), n, n_hit, hit_rate, n_over, over_rate, n_under, under_rate and kappa. The confusion attribute holds the matrix (truth in rows, estimate in columns), or a named list of them when by is given; meta records the columns used and how many rows were unusable.

Direction matters

A triage error is not symmetric. On the severity order `red > yellow > green`, calling a casualty *more* urgent than they are is an **over-triage**: it wastes a scarce resource. Calling them *less* urgent is an **under-triage**: it can cost a life. A single accuracy number hides which of the two a team is doing, so both rates are reported alongside the hit rate, per category and overall.

Cohen's kappa is reported on the overall row: it is a property of the whole table, and repeating it on every category row would state it four times without adding anything.

See Also

`sdt_read_gold_standard()`, `sdt_apply_gold_standard()`.

Examples

```
calls <- data.frame(
  triage = c("red", "red", "yellow", "green", "green"),
  called_triage = c("red", "yellow", "yellow", "yellow", "green")
)
acc <- sdt_triage_accuracy(calls)
acc
attr(acc, "confusion")
```

---

sdt_vocabularies *Controlled vocabularies for the mass-casualty domain*

Description

Character vectors giving the allowed levels for the core categorical fields used throughout the package. Exposed so downstream code and the Shiny app can validate against a single source of truth.

Usage

```
sdt_triage_levels()

sdt_therapy_levels()

sdt_diagnostics()

sdt_board_areas()

sdt_life_range()

sdt_injury_regions()

sdt_sex_levels()
```

Format

Each is a character vector:

`sdt_triage_levels()` Sighting categories `"red"`, `"yellow"`, `"green"` (SK I / II / III).

`sdt_therapy_levels()` Therapy concepts `"ETC"`, `"DCS"`, `"TASC"`, and `"none"` for the card's "no operation" option.

`sdt_diagnostics()` Diagnostic modalities printed on the card: `"efast"`, `"xray"`, `"ct"`.

`sdt_board_areas()` The twelve board states a casualty can occupy: the nine of the published board, `"morgue"` and `"external"` for the two terminal states real exercises produce but the published figure does not report (died on the board, and transferred out of the hospital), and `"not_seen"` for a casualty the game ended before. The engine has no branch to `"morgue"` or `"external"`; they exist so that observed data can be represented without being folded into a state it is not.

`sdt_life_range()` The inclusive range of life points, `0:6`. Zero is the death marker printed on the version 2.0 card: a casualty who ran out of life points during play. Generators draw from `1:6` only, since death is a state reached in play rather than a card attribute.

`sdt_injury_regions()` Leading injury region recorded on the card.

`sdt_sex_levels()` Recorded sex: `"f"`, `"m"`, `"d"` (diverse).

Value

A character vector of allowed levels.

See Also

`sdt_reference()` for the published marginals over these levels.

Examples

```
sdt_triage_levels()
sdt_therapy_levels()
sdt_diagnostics()
sdt_board_areas()
sdt_life_range()
sdt_injury_regions()
sdt_sex_levels()
```

---

sdt_workflowset *Screen every engine on one cohort*

Description

Fits each available engine to the same data with the same preprocessing and the same resamples, and returns a leaderboard. This is the whole bench: glass-box and black-box models scored side by side, so the price of interpretability is a number rather than an opinion.

Usage

```
sdt_workflowset(
  data,
  target = c("triage", "therapy", "called_triage", "called_therapy"),
  engines = NULL,
  predictors = NULL,
  tune = TRUE,
  grid = 5L,
  folds = 5L,
  prop = 0.75,
  parallel = FALSE,
  workers = NULL,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

data	A cohort or data frame of casualties.
target	Outcome to predict. "triage" and "therapy" are the true values on the card; "called_triage" and "called_therapy" are what a participant assigned, as produced by sdt_add_decisions(). The true values are close to a deterministic function of the card, so every engine scores near 1 on them and the comparison is uninformative. The participant's call is the target worth benchmarking.
engines	Which engines to screen. Defaults to all of them.
predictors	Character vector of predictor columns. Defaults to the card fields, excluding identifiers and every outcome column, so a model for the participant's call cannot simply read the true category.
tune	If TRUE, tune hyperparameters over a grid; if FALSE, fit the specification's defaults once.
grid	Number of candidate hyperparameter settings.
folds	Number of cross-validation folds, stratified on the target.
prop	Proportion of the data used for training in the initial split.
parallel	If TRUE, fit resamples and grid points in parallel. Defaults to FALSE: a package that claims every core without being asked is a bad guest on a shared machine, and a sequential run needs no thought about RNG streams. It pays off on a tuned fit or an engine screen and costs a little on a single untuned one. The backend is set up for the duration of the call and the caller's plan is restored on exit.
workers	Number of parallel workers, or NULL to choose one: a core short of the machine, capped at 8.
seed	Optional integer seed, applied with withr::local_seed().
verbose	If TRUE, report progress via **cli**.

Details

Engines whose package is not installed are skipped, and named in the `skipped` attribute rather than silently dropped. So is the transformer when **keras3** is installed but has no working backend. The transformer is fitted after every other engine, so Python and its backend are not loaded into the session before their parallel workers are started.

Value

A `tibble::tibble()` with one row per engine and metric: `engine`, `class_box`, `.metric`, `.estimator`, `.estimate`, ranked with the best `roc_auc` first. Carries a `skipped` attribute listing engines that could not be fitted here, and a `models` attribute holding the fitted `sdt_model` objects.

See Also

`sdt_fit()`, `sdt_metrics()`.

Examples

```
co <- sdt_synthetic_cohort(n = 150, seed = 4)
needed <- c("parsnip", "recipes", "workflows", "rsample", "tune",
            "yardstick", "rpart")
if (all(vapply(needed, requireNamespace, logical(1), quietly = TRUE))) {
  sdt_workflowset(co, target = "triage", engines = "tree",
                  tune = FALSE, seed = 1, verbose = FALSE)
}
```

---

summary.game_run *End-of-game location distribution*

Description

Reports where the casualties stood when the clock stopped, which is the quantity the published reference distribution describes. A casualty's end state is the last board area it had entered by the horizon; one that never cleared triage counts as `"not_seen"`.

Usage

```
## S3 method for class 'game_run'
summary(object, reference = FALSE, version = c("2.0", "1.0"), ...)
```

Arguments

`object`	A `game_run` from `sdt_simulate()`.
`reference`	If `TRUE`, fold the areas onto the published reporting locations and add the reference proportion beside the observed one, with their difference.
`version`	Which evaluated version of the exercise to compare against, `"2.0"` or `"1.0"`. Only used when `reference = TRUE`.
`...`	These dots are for future extensions and must be empty.

Details

Two granularities are available. By default the tibble has one row per board area. With `reference = TRUE` it is folded onto the eight locations the source reports, because those are the only ones a published proportion exists for: the two emergency-department areas become one, and the green treatment area joins the ward. Both pairings are recorded in `sdt_reference("defaults")$end_state_map` rather than being applied here, so they can be inspected and argued with.

Value

A `tibble::tibble()` with columns `area`, `label`, `n` and `proportion`; when `reference = TRUE`, also `reference` and `difference`. Every area is present even at a zero count, so the shape does not depend on the run.

See Also

`sdt_reference()` for the published values and their provenance.

Examples

```
co <- sdt_synthetic_cohort(n = 40, seed = 1)
run <- sdt_simulate(sdt_scenario(co), replications = 2, seed = 1,
                    verbose = FALSE)
summary(run)
summary(run, reference = TRUE)
summary(run, reference = TRUE, version = "1.0")
```

`validate_cohort` *Validate a casualty cohort*

Description

Checks a `cohort` against the domain invariants: controlled vocabularies for `sex`, `injury_region`, `triage` and `therapy`; `life_points` inside `1:6`; unique non-missing `id`; and a positive whole `arrival_order`. All violations across all columns are collected and reported together.

Usage

```
validate_cohort(x, call = rlang::caller_env())
```

Arguments

`x`	A `cohort` from `sdt_cohort()`.
`call`	The calling environment, used to attribute the error.

Value

`x`, invisibly, on success; otherwise an error of class `sandtablR_error_cohort`.

Examples

```
co <- sdt_cohort(sdt_synthetic_cohort(n = 5, seed = 1))
validate_cohort(co)
```

`validate_patient_card` *Validate a patient card*

Description

Checks that a `patient_card` satisfies the domain invariants: the controlled vocabularies for `sex`, `injury_region`, `triage` and `therapy`, and `life_points` inside `1:6`.

Usage

```
validate_patient_card(x, call = rlang::caller_env())
```

Arguments

`x`	A `patient_card` object.
`call`	The calling environment, used to attribute the error.

Value

The validated `x`, invisibly on success; otherwise an error of class `sandtablR_error_card`.

See Also

`validate_cohort()`, which applies the same rules to a whole cohort in one vectorised pass.

Examples

```
card <- sdt_patient_card(
  "P01", 34, "m", 4L, TRUE, "thorax", "red", "DCS", 1L
)
validate_patient_card(card)
```

Index

as_tibble.scenario, 3
autoplot.scenario, 4

format.patient_card, 5
format.sdt_describe *(*sdt_describe*)*, 17

ggiraph::girafe(), *44*
ggplot2::autoplot(), *4*

knitr::kable(), *21*

predict.sdt_model, 5
print(), *5*
print.sdt_describe *(*sdt_describe*)*, 17

sdt_add_decisions, 6
sdt_add_decisions(), *30*, *74*
sdt_app, 7
sdt_app(), *57*
sdt_app_server, 8
sdt_app_server(), *7*, *9*, *57*
sdt_app_ui, 8
sdt_app_ui(), *7, 8*, *57*
sdt_apply_gold_standard, 9
sdt_apply_gold_standard(), *35*, *51, 52*, *71, 72*
sdt_area_labels, 10
sdt_as_patient_cards, 11
sdt_as_patient_cards(), *12*
sdt_board_areas *(*sdt_vocabularies*)*, 72
sdt_board_areas(), *10, 11*, *60*
sdt_cohort, 12
sdt_cohort(), *6*, *9*, *11*, *76*
sdt_consistency_score, 13
sdt_consistency_score(), *6*
sdt_course_composition, 13
sdt_course_composition(), *19*, *33, 34*, *42*, *53*, *64, 65*
sdt_decision_consistency, 15
sdt_decision_consistency(), *17*
sdt_decision_paths, 16
sdt_decision_paths(), *16*, *48*, *63*
sdt_describe, 17
sdt_describe(), *20, 21*
sdt_describe_participants, 20
sdt_describe_participants(), *19*
sdt_describe_render, 21
sdt_describe_render(), *18*
sdt_diagnostics *(*sdt_vocabularies*)*, 72
sdt_engines, 21
sdt_engines(), *30*, *70*
sdt_error_fidelity, 22
sdt_error_fidelity(), *36*, *38*, *67*
sdt_explain, 24
sdt_explain(), *31*, *38*, *50*, *61*
sdt_export_simulation, 25
sdt_export_simulation(), *65*
sdt_fidelity, 27
sdt_fidelity(), *23, 24*, *26, 27*, *46*, *57, 58*
sdt_fit, 30
sdt_fit(), *5, 6*, *24, 25*, *38*, *40*, *49, 50*, *56*, *61*, *75*
sdt_fit_behaviour, 32
sdt_fit_behaviour(), *24*, *35, 36*, *64, 65*
sdt_fit_triage_error, 35
sdt_fit_triage_error(), *66, 67*
sdt_importance, 38
sdt_importance(), *25*, *50*, *61*
sdt_injury_regions *(*sdt_vocabularies*)*, 72
sdt_injury_regions(), *43*
sdt_life_range *(*sdt_vocabularies*)*, 72
sdt_metric_set, 39
sdt_metrics, 40
sdt_metrics(), *31*, *75*
sdt_participant_dictionary, 40
sdt_participant_dictionary(), *52, 53*
sdt_participant_summary, 42
sdt_participant_summary(), *15*, *19*, *53*
sdt_patient_card, 43

sdt_patient_card(), *12*, *59*
sdt_plot_board, 44
sdt_plot_end_state, 45
sdt_plot_fidelity, 46
sdt_plot_fidelity(), *23*, *29*
sdt_plot_flow, 47
sdt_plot_flow(), *44*
sdt_plot_sequences, 47
sdt_plot_sequences(), *17*, *60, 61*
sdt_plot_triage_mix, 48
sdt_plot_triage_mix(), *4*
sdt_plot_utilisation, 49
sdt_predictions, 49
sdt_predictions(), *6*
sdt_profile, 50
sdt_profile(), *25*, *39*, *61*
sdt_read_gold_standard, 51
sdt_read_gold_standard(), *9, 10*, *72*
sdt_read_participants, 52
sdt_read_participants(), *14, 15*, *19, 20*, *41, 42*
sdt_reference, 54
sdt_reference(), *69*, *73*, *76*
sdt_reference_horizon, 55
sdt_reference_scenario, 55
sdt_roc, 56
sdt_roc(), *31*
sdt_run_app, 57
sdt_run_app(), *7–9*
sdt_run_behaviour, 57
sdt_run_behaviour(), *28, 29*
sdt_scenario, 59
sdt_scenario(), *12*, *62*, *69*
sdt_seq_cluster, 59
sdt_seq_cluster(), *16*, *48*, *61*
sdt_seqdef, 60
sdt_seqdef(), *16, 17*, *60*
sdt_sex_levels *(*sdt_vocabularies*)*, 72
sdt_shap, 61
sdt_shap(), *25*, *39*, *50*
sdt_simulate, 62
sdt_simulate(), *15, 16*, *28*, *33*, *44, 45*, *47–49*, *55–58*, *60*, *75*
sdt_simulate_groups, 63
sdt_simulate_groups(), *26–28*, *33, 34*
sdt_simulate_triage, 66
sdt_simulate_triage(), *22*, *24*, *37, 38*
sdt_synthesis_utility, 68
sdt_synthesis_utility(), *69*
sdt_synthesise, 68
sdt_synthesise(), *68*, *70*
sdt_synthetic_cohort, 69
sdt_synthetic_cohort(), *6*, *69*
sdt_therapy_levels *(*sdt_vocabularies*)*, 72
sdt_therapy_levels(), *43*
sdt_transformer, 70
sdt_triage_accuracy, 71
sdt_triage_accuracy(), *10*, *24*, *38*, *51, 52*
sdt_triage_levels *(*sdt_vocabularies*)*, 72
sdt_triage_levels(), *36*, *43*
sdt_vocabularies, *54*, 72
sdt_workflowset, 73
sdt_workflowset(), *31*
shiny::reactiveValues(), *8*
shiny::runApp(), *57*
shiny::shinyApp(), *7*
stats::predict(), *6*
stats::simulate(), *64*
summary.game_run, 75
summary.game_run(), *11*, *44, 45*, *56*, *58*, *63*
synthpop::syn(), *68, 69*

tibble, *23*, *28*, *58*, *67*
tibble::as_tibble(), *18*
tibble::tibble(), *6*, *10*, *12, 13*, *16–18*, *21, 22*, *40*, *50, 51*, *53*, *60*, *64*, *68*, *70, 71*, *75, 76*
TraMineR::seqdef(), *60*
TraMineR::seqdist(), *60*

utils::modifyList(), *40*

validate_cohort, 76
validate_cohort(), *12*, *77*
validate_patient_card, 77

withr::local_seed(), *6*, *31*, *62*, *64*, *66*, *69, 70*, *74*