

Package ‘scorecraft’

October 6, 2026

Title Scorecard Development and Internal Ratings-Based Risk Parameters

Version 0.3.0

Description Builds points scorecards for binary targets (credit risk, fraud, propensity) on the optimal binning and weight of evidence engine of 'OptimalBinningWoE', and takes them to the risk parameters of the internal ratings-based (IRB) approach. Variables are selected through optimal binning, eight admission rules, hold-out revalidation with frozen bins and a consensus of 'glmnet', 'xgboost', 'lightgbm' and 'ranger' models weighted by out-of-sample performance; the audit funnel never drops a candidate from the report. The scorecard is fitted with an explicit, auditable scale alignment (a log-odds regression on the raw score composed with the points-to-double-the-odds map); cut-offs are swept with frozen cuts; reject inference is reported as a sensitivity band; the population and characteristic stability indices (PSI and CSI) are monitored with both the fixed and the sample-size-adjusted threshold; and production SQL is generated in fourteen dialects, with the agreement between R and SQL verified by test. The IRB layer builds the default flag; calibrates the scorecard to a long-run default rate with rating grades, margins of conservatism and floors to give the probability of default (PD); models workout loss given default (LGD) in two stages with downturn and in-default estimates; models credit conversion factors from facility snapshots to give the exposure at default (EAD); and computes expected loss, risk weights, regulatory capital and expected credit loss from parameter tables selected by framework preset. The heavy numeric kernels (rank correlation of wide weight of evidence tables, exact concordance counts for Somers' D, streamed expected credit loss paths) are compiled with 'RcppArmadillo'. The scorecard methodology follows Siddiqi (2017) <doi:10.1002/9781119282396> and Thomas et al. (2017) <doi:10.1137/1.9781611974560>.

License MIT + file LICENSE

URL <https://github.com/evandeilton/scorecraft>

BugReports <https://github.com/evandeilton/scorecraft/issues>

Encoding UTF-8

Language en-GB

Depends R (>= 4.1.0)

Imports data.table (>= 1.14.0), OptimalBinningWoE (>= 1.13.4),
xgboost, stats, utils, graphics, parallel, Rcpp (>= 1.0.10)

LinkingTo Rcpp, RcppArmadillo

Suggests glmnet, lightgbm, ranger, DBI, odbc, RSQLite, duckdb,
openxlsx, betareg, bit64, testthat (>= 3.0.0), knitr,
rmarkdown, withr

LazyData true

LazyDataCompression xz

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel false

Config/roxygen2/version 8.1.0

NeedsCompilation yes

Author Jose Evandeilton Lopes [aut, cre, cph]

Maintainer Jose Evandeilton Lopes <evandeilton@gmail.com>

Repository CRAN

Date/Publication 2026-10-06 16:10:02 UTC

Contents

predict.scr_align	4
predict.scr_grades	5
predict.scr_pd	6
scr_align	7
scr_apply	9
scr_bin	10
scr_bin_continuous	12
scr_calibrate	13
scr_capital	15
scr_classing_accept	18
scr_classing_apply	19
scr_classing_choose	20
scr_classing_propose	21
scr_classing_spec	22
scr_classing_view	23
scr_coarse_classing	24
scr_compare	26
scr_config	27
scr_config_keys	29
scr_connect	30
scr_core	31
scr_cutoff	31

scr_decisions	32
scr_default	33
scr_default_rate	35
scr_demo	36
scr_demo_ead	37
scr_demo_lgd	38
scr_demo_lgd_cashflows	39
scr_demo_panel	40
scr_demo_portfolio	41
scr_demo_rates	42
scr_ead	43
scr_ead_data	44
scr_ead_downturn	46
scr_ead_validate	47
scr_ecl	49
scr_el	51
scr_elbe	52
scr_export	53
scr_fetch	55
scr_funnel	56
scr_gains	57
scr_grades	58
scr_irb_params	60
scr_irb_rw	61
scr_iv	63
scr_leakage	64
scr_lgd	65
scr_lgd_downturn	66
scr_lgd_floor	67
scr_lgd_pools	68
scr_lgd_validate	69
scr_master_scale	71
scr_metrics	72
scr_migration	73
scr_moc	74
scr_model	76
scr_monitor	77
scr_monitoring_plan	78
scr_pd	79
scr_pd_pit_ttc	80
scr_pd_stress	81
scr_pd_validate	82
scr_presets	84
scr_psi	84
scr_reasons	86
scr_reject	87
scr_result	88
scr_run	89

scr_runset	90
scr_sa_rw	91
scr_scorecard	93
scr_score_gains	95
scr_score_metrics	95
scr_select	96
scr_selected	98
scr_split	99
scr_sql	100
scr_strategy	103
scr_triage	104
scr_verbose	105
scr_workout	106
Index	108

predict.scr_align	<i>Apply an alignment to raw scores</i>
-------------------	---

Description

Apply an alignment to raw scores

Usage

```
## S3 method for class 'scr_align'  
predict(object, raw, type = c("score", "prob"), ...)
```

Arguments

object	An object from scr_align() .
raw	Raw scores on the same scale used in the fit.
type	"score" (default) returns points; "prob" returns the calibrated event probability implied by the alignment.
...	Ignored.

Value

A numeric vector of the length of raw.

See Also

Other production: [scr_apply\(\)](#), [scr_export\(\)](#), [scr_monitor\(\)](#), [scr_monitoring_plan\(\)](#), [scr_reasons\(\)](#), [scr_sql\(\)](#)

Examples

```
set.seed(3)
y <- stats::rbinom(2000, 1, 0.15)
raw <- stats::qlogis(0.15) + 1.3 * y + stats::rnorm(2000)
al <- scr_align(raw, y)
head(predict(al, raw))
head(predict(al, raw, type = "prob"))
```

predict.scr_grades	<i>Grade a score vector with the cut points of an scr_grades object</i>
--------------------	---

Description

Grade a score vector with the cut points of an scr_grades object

Usage

```
## S3 method for class 'scr_grades'
predict(object, score, type = c("grade", "pd"), ...)
```

Arguments

object	An scr_grades() object.
score	Numeric production scores.
type	"grade" (integer grade) or "pd" (calibrated individual PD).
...	Ignored.

Value

A vector of the length of score.

See Also

Other irb-pd: [predict.scr_pd\(\)](#), [scr_calibrate\(\)](#), [scr_grades\(\)](#), [scr_master_scale\(\)](#), [scr_migration\(\)](#), [scr_moc\(\)](#), [scr_pd\(\)](#), [scr_pd_pit_ttc\(\)](#), [scr_pd_validate\(\)](#)

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  use_lightgbm = FALSE, xgb_rounds = 40, n_boot = 10)
d <- scr_demo[, c("default", "ref_date", "ds_region", "ds_band", "vl_score_01",
                  "vl_score_02", "vl_score_05", "vl_score_10", "vl_hist_01")]
res <- scr_select(d, "default", config = cfg, date_col = "ref_date")
sc <- scr_scorecard(res)
gr <- scr_grades(sc, scr_calibrate(sc, target = 0.06), n_grades = 7, min_defaults = 10)
predict(gr, score = c(480, 560, 640))
predict(gr, score = c(480, 560, 640), type = "pd")
```

predict.scr_pd	<i>Predict grade and PD from an scr_pd object</i>
----------------	---

Description

Predict grade and PD from an scr_pd object

Usage

```
## S3 method for class 'scr_pd'
predict(
  object,
  newdata = NULL,
  score = NULL,
  type = c("grade", "pd", "pd_final", "score"),
  ...
)
```

Arguments

object	An <code>scr_pd()</code> object.
newdata	A table with the source columns of the scorecard, scored with <code>scr_apply()</code> ; ignored when score is given.
score	Production scores, as an alternative to newdata.
type	"grade", "pd" (calibrated individual PD), "pd_final" (grade PD after MoC and floor) or "score".
...	Ignored.

Value

A vector of the length of the input.

See Also

Other irb-pd: `predict.scr_grades()`, `scr_calibrate()`, `scr_grades()`, `scr_master_scale()`, `scr_migration()`, `scr_moc()`, `scr_pd()`, `scr_pd_pit_ttc()`, `scr_pd_validate()`

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
  use_lightgbm = FALSE, xgb_rounds = 40, n_boot = 10)
d <- scr_demo[, c("default", "ref_date", "ds_region", "ds_band", "vl_score_01",
  "vl_score_02", "vl_score_05", "vl_score_10", "vl_hist_01")]
res <- scr_select(d, "default", config = cfg, date_col = "ref_date")
sc <- scr_scorecard(res)
pd <- scr_pd(scr_grades(sc, n_grades = 6, min_defaults = 10))
predict(pd, score = c(480, 560, 640), type = "pd_final")
head(predict(pd, newdata = scr_demo[1:10, ], type = "grade"))
```

scr_align

*Stage 5: align a raw score to the declared scale***Description**

Takes the raw score of **any** engine (the scorecard logit, the output of a tree challenger, a legacy score) to the scale defined by base_score, base_odds and pdo, recording odds_orientation on the object. This is what makes two scorecards directly comparable. It runs automatically inside [scr_scorecard\(\)](#); it is exposed to align other scores to the same scale.

Usage

```
scr_align(
  raw,
  y,
  base_score = 600,
  base_odds = 50,
  pdo = 20,
  direction = c("higher_is_safer", "higher_is_riskier"),
  method = c("regression", "direct"),
  n_bands = 10L,
  laplace = 0.5,
  weights = NULL
)
```

Arguments

raw	Raw score: an event logit (or any score on which a higher value means a higher probability of the event).
y	0/1 outcome vector (numeric or logical), same length as raw.
base_score	Score at which the odds are base_odds.
base_odds	Odds at base_score, positive, in the orientation of direction.
pdo	Points that double the odds, positive.
direction	"higher_is_safer" or "higher_is_riskier".
method	"regression" (default) or "direct".
n_bands	Bands of the calibration regression.
laplace	Smoothing of the counts per band.
weights	Optional non-negative weights per observation (sample reweighting), of the length of raw.

Value

An scr_align object with base_score, base_odds, pdo, direction, odds_orientation, factor, offset, sign, calibration (method, intercept, slope, r2, n_bands, bands) and the final coefficients a and b of $\text{score} = a + b * \text{raw}$. Use [predict.scr_align\(\)](#) to apply it.

Mechanism

With method = "regression" (default): the raw score is banded by quantiles on the reference data, the empirical log-odds of every band is computed with Laplace smoothing in the orientation direction implies, and a regression weighted by band size fits

$$\ln(\text{odds}) = I + S \cdot \text{raw}.$$

This absorbs sample reweighting, miscalibration of the WOE fit and prior shift. It then composes with the PDO map:

$$\text{factor} = \text{pdo} / \ln 2, \quad \text{offset} = \text{base_score} - \text{factor} \cdot \ln(\text{base_odds}),$$

$$\text{score} = \text{offset} + \text{factor} (I + S \cdot \text{raw}) = a + b \cdot \text{raw}.$$

With method = "direct", the model is assumed calibrated: $I = 0$ and S is the sign of the direction (-1 under higher_is_safer, +1 under higher_is_riskier), that is, $\ln(\text{odds})$ is raw itself in the right orientation.

Odds orientation

base_odds is always expressed in the orientation direction implies: non-event:event ("safe:event") under higher_is_safer, event:non-event ("event:safe") under higher_is_riskier. The same word, odds, changes meaning between the two, and that is the most common sign trap in the literature; hence the object records odds_orientation explicitly.

References

Siddiqi, N. (2006). *Credit Risk Scorecards*. Wiley, chapter 6.

See Also

Other stages: [scr_bin\(\)](#), [scr_cutoff\(\)](#), [scr_model\(\)](#), [scr_reject\(\)](#), [scr_scorecard\(\)](#), [scr_select\(\)](#), [scr_split\(\)](#), [scr_strategy\(\)](#), [scr_triage\(\)](#)

Examples

```
set.seed(3)
y <- stats::rbinom(4000, 1, 0.15)
raw <- stats::qlogis(0.15) + 1.3 * y + stats::rnorm(4000, sd = 1.2)
al <- scr_align(raw, y, base_score = 600, base_odds = 50, pdo = 20)
al
head(predict(al, raw))
head(predict(al, raw, type = "prob"))

# propensity: more points = more event, odds event:non-event
scr_align(raw, y, base_score = 500, base_odds = 1/9, pdo = 40,
          direction = "higher_is_riskier")
```

scr_apply

*Apply the WOE transformation or the scorecard to new data***Description**

Materialises in R exactly what the production SQL does: the frozen Stage 1 pre-processing (training median, special-population flags, "MISSING") followed by the frozen Stage 2 binning and, for a scorecard, by the points. Nothing is refitted. The two paths, R and SQL, produce the same numbers, and a test guarantees it.

Usage

```
scr_apply(x, newdata, ...)

## S3 method for class 'scr_result'
scr_apply(
  x,
  newdata,
  features = scr_selected(x),
  what = c("woe", "bin", "both"),
  ...
)

## S3 method for class 'scr_scorecard'
scr_apply(x, newdata, what = c("score", "points", "woe", "all"), ...)

## S3 method for class 'scr_ead'
scr_apply(x, newdata, what = c("all", "ead", "pool"), ...)

## S3 method for class 'scr_lgd'
scr_apply(x, newdata, what = c("pool", "lgd", "all"), ...)

## S3 method for class 'scr_pd'
scr_apply(x, newdata, ...)
```

Arguments

x	An object from <code>scr_select()</code> (returns WOE/bin of the approved variables) or from <code>scr_scorecard()</code> (returns score and points).
newdata	New table with the source columns of the requested variables. The target column is not needed.
...	Passed on to the methods.
features	For <code>scr_result</code> : which variables to transform. Defaults to the approved ones.
what	For <code>scr_result</code> : "woe", "bin" or "both". For <code>scr_scorecard</code> : "score", "points", "woe" or "all". For <code>scr_lgd</code> : "pool" (pool and pool LGDs), "lgd" (adds the predicted LGD) or "all" (adds the cure probability and the severity).

For `scr_ead`: "all" (default), "ead" (pool, measure, applied CCF, predicted EAD and the floor flag) or "pool" (pool and measure).

Value

A `data.table` with one row per row of newdata.

Output columns

For `scr_result`: `<f>_woe` and/or `<f>_bin` per variable. For `scr_scorecard`, "score" gives link (logit), prob (model probability), score (exact, $a + b * \text{logit}$) and score_points (base plus the whole points per bin); "points" gives score, score_points and `<f>_points`; "woe" gives link, score and `<f>_woe`; "all" gives everything.

IRB models

`scr_pd` returns score, score_points, grade, pd (calibrated individual PD), pd_be and pd_final of the grade. `scr_lgd` returns pool, lgd_lra, lgd_dt, lgd_final and, with what, p_cure, severity and lgd_pred. `scr_ead` returns pool, measure, utilisation, undrawn, ccf_applied, ead_model, ead_floor, ead_predicted and ead_floor_binding; the predicted EAD is never below the drawn amount. `scr_capital()` reads pd_final, lgd_final and ead_predicted from these outputs in its list form.

See Also

Other production: `predict.scr_align()`, `scr_export()`, `scr_monitor()`, `scr_monitoring_plan()`, `scr_reasons()`, `scr_sql()`

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
  date_col = "ref_date")
new <- head(scr_demo, 50)
str(scr_apply(res, new)[, 1:3])
sc <- scr_scorecard(res)
head(scr_apply(sc, new))
head(scr_apply(sc, new, what = "points"))
```

scr_bin

Stage 2: optimal binning, screening, hold-out revalidation and pruning

Description

Fits the bins **on the training rows only** (cut points and WOE use the target), in parallel by column, and applies in sequence:

Usage

```
scr_bin(triage, config = scr_config())
```

Arguments

triage	An object from scr_triage() .
config	An object from scr_config() .

Details

1. **Screening**, native to the engine: eight admission rules (IV_BELOW_MIN, IV_SUSPICIOUS, NOT_MONOTONIC, TOO_FEW_BINS, TOO_MANY_BINS, SMALL_BIN, DEGENERATE_BIN, BINNING_ERROR).
2. **Hold-out revalidation** with frozen bins: IV recomputed on the same labels, train/hold-out PSI (the fixed threshold decides; the n-adjusted one is reported) and the fraction of hold-out without a bin.
3. **Redundancy pruning** by rank correlation on the WOE space, ranked by hold-out IV. Under `allow_derived_final = FALSE` the derived flags leave before this step (`derived_excluded`), so that a flag that cannot be delivered never prunes a real column.

Value

An `scr_bins` object with `fit` (an `obwoe` object), `screen` (summary and full), `holdout`, `prune`, `pool` (eligible for the models), `derived_excluded`, the counts binned, `pos_screen` and `pos_holdout` (the survivors of each gate, in order), the WOE matrices `woe_train/woe_holdout`, the originating `triage` and the `config`.

Parallelism

Columns are split into `config$nthread` chunks, each chunk is binned by a worker and the fits are merged. The result is identical to the serial one (a test pins this): the engine is deterministic per column.

See Also

Other stages: [scr_align\(\)](#), [scr_cutoff\(\)](#), [scr_model\(\)](#), [scr_reject\(\)](#), [scr_scorecard\(\)](#), [scr_select\(\)](#), [scr_split\(\)](#), [scr_strategy\(\)](#), [scr_triage\(\)](#)

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1)
sp <- scr_split(scr_demo, "default", date_col = "ref_date", drop = "id")
bn <- scr_bin(scr_triage(sp, cfg), cfg)
bn
head(bn$holdout)
```

scr_bin_continuous *Bin drivers against a continuous target (LGD, CCF)*

Description

Supervised binning for a bounded continuous target, with the result in the shape of an obwoe object, so that the `scr_apply()` and `scr_sql()` machinery (`OptimalBinningWoE::obwoe_apply()` and `OptimalBinningWoE::obwoe_sql()`) reproduces the bin statistic unchanged. The woe slot of every bin carries the target mean of the bin (or its logit with `scale = "logit"`); `iv` carries the bin's share of the between-bin sum of squares, so `total_iv` is the eta-squared of the driver, in $[0, 1]$.

Usage

```
scr_bin_continuous(
  data,
  target,
  features,
  train_idx = NULL,
  holdout_idx = NULL,
  min_bins = 2L,
  max_bins = 6L,
  min_share = 0.05,
  min_n = 30L,
  monotone = c("auto", "increasing", "decreasing", "none"),
  scale = c("mean", "logit"),
  nthread = 1L,
  alpha = 0.05
)
```

Arguments

<code>data</code>	A <code>data.frame</code> or <code>data.table</code> .
<code>target</code>	Column name of the continuous target.
<code>features</code>	Column names of the drivers.
<code>train_idx, holdout_idx</code>	Row indices; <code>NULL</code> uses every row for training and skips the revalidation.
<code>min_bins, max_bins</code>	Target range of bins per driver.
<code>min_share</code>	Minimum share of training rows per bin.
<code>min_n</code>	Minimum number of training rows per bin.
<code>monotone</code>	"auto" (direction from the Spearman sign), "increasing", "decreasing" or "none".
<code>scale</code>	"mean" (bin mean in the woe slot) or "logit".
<code>nthread</code>	Parallel workers by driver, through the package backend.
<code>alpha</code>	Alpha of the PSI critical value in the revalidation.

Details

Numeric drivers must not contain missing values: run `scr_triage()` (or `impute`) first, exactly as the scorecard pipeline does. Categorical missing values become the level "NA", as in the engine. When a `holdout_idx` is given, the frozen bins are revalidated: the hold-out bin means are recomputed, the PSI of the bin shares is reported with the sample-size-adjusted critical value, a driver whose hold-out means break the training order is flagged `UNSTABLE_HOLDOUT`, one whose bin shares shift (fixed PSI flag "shift", PSI at or above 0.25) is flagged `PSI_ACTION`, and one with more than 1% of hold-out rows outside the bins `UNBINNED_HOLDOUT`.

Value

An object of class `scr_cbins`: `fit` (the `obwoe`-shaped object), `summary` (one row per driver: `feature`, `type`, `n_bins`, `eta2`, `direction`, `converged`, and after revalidation `eta2_holdout`, `psi`, `psi_flag`, `holdout_ok`, `holdout_reason`), `holdout` (bin table per driver with train and hold-out means), `scale` and `target`. `summary` keeps the engine columns (`algorithm`, `total_iv`, `iterations`, `error`) and, after revalidation, `psi_critical`, `psi_flag_adjusted` and `pct_unbinned`.

See Also

Other `irb-ead`: `scr_ead()`, `scr_ead_data()`, `scr_ead_downturn()`, `scr_ead_validate()`

Examples

```
set.seed(1)
d <- data.frame(x = runif(600), g = sample(c("a", "b", "c", "d"), 600, TRUE))
d$y <- pmin(1, pmax(0, 0.2 + 0.6 * d$x + (d$g == "d") * 0.2 + rnorm(600, 0, 0.1)))
cb <- scr_bin_continuous(d, "y", c("x", "g"), train_idx = 1:400, holdout_idx = 401:600)
cb
cb$fit$results$x$bin
cb$fit$results$x$woe      # bin means of y
```

scr_calibrate

Calibrate the alignment to a central tendency

Description

Re-anchors the probability of default of a scorecard to a long-run average default rate (the central tendency, CT) without touching the points: the result is a **new** alignment (I^* , S^*) such that `predict(alignment, raw, type = "prob")` is the calibrated PD, while the scorecard keeps its own alignment for the score. Four methods:

Usage

```
scr_calibrate(
  x,
  target,
  sample_rate = NULL,
  method = NULL,
```

```

    ar_target = NULL,
    segment = NULL,
    raw = NULL,
    y = NULL,
    sample = "holdout"
)

```

Arguments

x	An <code>scr_scorecard()</code> (uses the $\ln(\text{odds})$ and outcome of sample), an <code>scr_align()</code> (pass raw and, for the two-parameter methods, y) or a numeric vector of event $\ln(\text{odds})$ (aligned directly to the default 600/50/20 scale).
target	The central tendency: a number in (0, 1) or an <code>scr_dr</code> from <code>scr_default_rate()</code> (its <code>lra\$mean</code> is used). With segment, a named vector with one CT per segment.
sample_rate	Event rate of the calibration sample; NULL uses the mean of y.
method	"intercept", "logodds_ab", "scaling" or "qmm"; NULL uses <code>config\$pd_calibration</code> .
ar_target	Target accuracy ratio for "logodds_ab" and "qmm".
segment	Optional vector of segment labels, one per calibration row: one alignment per segment is fitted as well.
raw, y	Raw $\ln(\text{odds})$ and 0/1 outcome when x is not a scorecard.
sample	Sample of the scorecard used for the calibration.

Details

- "intercept" The prior-correction shift of King and Zeng (2001), $\delta = \ln[\tau(1 - \bar{y})/((1 - \tau)\bar{y})]$, added to the event $\ln(\text{odds})$; S unchanged, so the rank order and every discrimination statistic are untouched. The closed form is exact on the odds; when the calibration sample is available the shift is refined by a one-dimensional root so that the mean PD equals the CT exactly (the closed form is reported as `shift_prior`).
- "logodds_ab" Tasche (2013): $\ln(\text{odds}^*) = a + b \ln(\text{odds})$, with (a, b) solving $\text{mean}(\text{PD}^*) = \text{CT}$ and implied accuracy ratio equal to `ar_target` (default: the accuracy ratio observed on the sample). With the observed accuracy ratio this is Tasche's quasi-moment matching (QMM) proper. The implied AUC is the probability that a default has a higher PD than a non-default when the PDs are true: each score carries weight PD^* among the defaults and $1 - \text{PD}^*$ among the non-defaults, ties counted one half.
- "qmm" The outcome-free variant of the same two equations: the target accuracy ratio is the implied AR of the current PDs, so no outcome is needed and the implied discriminatory power of the uncalibrated curve is carried over to the new level.
- "scaling" $\text{PD}^* = \text{PD} * \text{CT} / \bar{y}$. The proportional rescaling is not a logit map, so the slope is the least-squares projection of $\text{logit}(\text{PD}^*)$ on the $\ln(\text{odds})$ and the intercept is solved to the CT.

Value

An object of class `scr_pd_calibration`: `alignment` (the new `scr_align`), `alignment_before`, `method`, `ct`, `target_source`, `sample_rate`, `shift` (change of the event intercept), `shift_prior`

(the closed-form King-Zeng shift), slope_ratio (S^* / S), mean_pd_before, mean_pd_after, ar_before, ar_after (observed), ar_implied_before, ar_implied_after, n, segments (table and alignments when segment is given), ledger. Also ar_target, note and sample.

References

- King, G. and Zeng, L. (2001). Logistic regression in rare events data. *Political Analysis*, 9(2), 137-163.
- Tasche, D. (2013). The art of probability-of-default curve calibration. *Journal of Credit Risk*, 9(4), 63-103.

See Also

Other irb-pd: [predict.scr_grades\(\)](#), [predict.scr_pd\(\)](#), [scr_grades\(\)](#), [scr_master_scale\(\)](#), [scr_migration\(\)](#), [scr_moc\(\)](#), [scr_pd\(\)](#), [scr_pd_pit_ttc\(\)](#), [scr_pd_validate\(\)](#)

Examples

```
set.seed(1)
l <- stats::qlogis(0.12) + stats::rnorm(2000)
y <- stats::rbinom(2000, 1, stats::plogis(l))
cal <- scr_calibrate(l, target = 0.04, y = y)
cal
mean(predict(cal$alignment, l, type = "prob"))
scr_calibrate(l, target = 0.04, y = y, method = "logodds_ab", ar_target = 0.55)
```

scr_capital

Expected loss, risk-weighted assets and capital of a portfolio

Description

Runs [scr_irb_rw\(\)](#) on every exposure, aggregates by segment, compares the IRB result with the standardised approach for the output floor, reconciles regulatory expected loss with the provision stock (shortfall deducted from capital; excess eligible as tier 2 up to 0.6 % of the IRB risk-weighted assets), measures the impact of each input floor, runs a fixed sensitivity grid and reports the name concentration of the book. The parameter tables come from params; the object records whether they were edited.

Usage

```
scr_capital(
  x,
  pd = "pd",
  lgd = "lgd",
  ead = "ead",
  segment = NULL,
  asset_class = config$asset_class,
  m = NULL,
```

```

defaulted = NULL,
elbe = NULL,
provisions = NULL,
ltv = NULL,
rating = NULL,
sales = NULL,
fi = NULL,
transactor = NULL,
grade = NULL,
id = NULL,
claim = NULL,
granular = TRUE,
params = scr_irb_params(config$framework),
config = scr_config(),
keep_rows = FALSE
)

```

Arguments

x	A table of exposures (data.frame or data.table) or the list form described above.
pd, lgd, ead	Column names of the probability of default, loss given default and exposure at default.
segment	Optional column name of the reporting segment.
asset_class	A column name or a single asset class (see scr_irb_rw()).
m, defaulted, elbe, provisions, ltv, rating, sales, fi, transactor, grade, id	Optional column names: effective maturity, default flag, best estimate of expected loss, provision stock, loan-to-value, external rating, annual sales, financial-institution flag, transactor flag, PD grade (defines the SQL pools together with segment) and exposure identifier.
claim	Optional column name: the claim type of each exposure under the foundation approach (a row of params\$lgd_firb); the supervisory LGD then replaces lgd.
granular	TRUE, FALSE or a column name: whether the retail exposures belong to a granular regulatory retail pool (the standardised comparison uses the non-granular weight otherwise).
params	An scr_irb_params() object; defaults to the preset of config\$framework.
config	An scr_config() object (capital_approach, capital_target_ratio, capital_output_floor, capital_sensitivity, nthread, verbose).
keep_rows	Keep the per-exposure table in the object.

Value

An object of class `scr_capital`: a list with exposures (per-exposure table, only with `keep_rows = TRUE`), segments (the reconciliation table: `segment`, `n`, `ead`, `pd_mean`, `lgd_mean`, `m`, `r_mean`, `k_mean`, `rw`, `rwa_irb`, `rwa_sa`, `irb_sa_ratio`, `el`, `provisions`, `shortfall_excess`), pools (one

row per segment and grade with the constants the SQL emits), totals (n, ead, el, rwa_irb, rwa_sa, irb_sa_ratio, output_floor, rwa_floor, rwa_reported, floor_binding, headroom, density, target_ratio, capital, provisions, shortfall, excess, tier2_addback, tier2_cap, hhi, n_eff, max_share, granular), floors (floor, n_hit, ead_hit, delta_rwa), sensitivity (shock, rwa, delta, delta_pct), concentration (share of EAD and RWA by segment), framework, approach, params, config, ledger, model_card and, after `scr_export()`, files. segments and totals also carry n_defaulted; totals also el_rate and rwa_irb_no_floors; concentration has segment, n, ead, rwa, ead_share, rwa_share and hhi_contribution; columns records the column names the SQL reads.

Inputs

x is either a table of exposures, the remaining arguments naming its columns, or a list `list(pd = , lgd = , ead = , data = )` whose elements are fitted models with an `scr_apply()` method (the PD, LGD and EAD objects of the IRB modules) and data the table to apply them to. In the list form each model present fills the corresponding vector from the columns `pd_final`, `lgd_final` and `ead_predicted` of its `scr_apply()` output, and the provenance is written to the ledger; elements that are NULL fall back to the named columns of data.

asset_class is a column name of x or a single class applied to every row. Segment means are weighted by EAD. The sensitivity grid shocks the PD (x1.10, x1.25, x1.50), the LGD (+5 percentage points), the EAD (+10 %), removes the input floors, scales the correlation (x1.25) and stresses the PD with the one-factor model at $q = 0.95$ and 0.99 (`scr_pd_stress()`, the stressed PD then re-entering the function so that the correlation follows it).

References

Basel Committee on Banking Supervision (2023). *The Basel Framework*, CRE31, CRE35 (treatment of expected losses and provisions), RBC20 (output floor).

See Also

Other irb-capital: `scr_ecl()`, `scr_el()`, `scr_irb_rw()`, `scr_pd_stress()`, `scr_sa_rw()`

Examples

```
cfg <- scr_config(verbose = FALSE)
cap <- scr_capital(scr_demo_portfolio, segment = "segment", asset_class = "asset_class",
  m = "m", defaulted = "defaulted", elbe = "elbe", provisions = "provision",
  ltv = "ltv", rating = "rating", sales = "sales", transactor = "transactor",
  grade = "grade", id = "id", config = cfg)

cap
cap$segments[, c("segment", "n", "rw", "irb_sa_ratio")]
cap$floors
```

scr_classing_accept	<i>Accept or discard a proposal</i>
---------------------	-------------------------------------

Description

reason is mandatory. Accepting replaces the variable's current bins; the previous accepted proposal is marked superseded. A BLOCKED proposal needs override = TRUE, and the override is itself a ledger row.

Usage

```
scr_classing_accept(lab, proposal, reason, override = FALSE)
```

```
scr_classing_discard(lab, proposal, reason)
```

Arguments

lab	An object from scr_coarse_classing() .
proposal	An object from scr_classing_propose() .
reason	Free text, at least 5 characters.
override	Accept a BLOCKED proposal.

Value

The updated lab, invisibly.

See Also

[scr_coarse_classing\(\)](#) for a complete session, from lab to scorecard.

Other classing: [scr_classing_apply\(\)](#), [scr_classing_choose\(\)](#), [scr_classing_propose\(\)](#), [scr_classing_spec\(\)](#), [scr_classing_view\(\)](#), [scr_coarse_classing\(\)](#), [scr_decisions\(\)](#)

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                 use_lightgbm = FALSE, xgb_rounds = 40, n_boot = 10)
d <- scr_demo[, c("default", "ref_date", "ds_region", "ds_band", "vl_score_01",
                 "vl_score_02", "vl_score_05", "vl_score_10", "vl_hist_01")]
res <- scr_select(d, "default", config = cfg, date_col = "ref_date")
lab <- scr_coarse_classing(res)
p <- scr_classing_propose(lab, "ds_region",
                        groups = list(edge = c("NORTH", "SOUTH"),
                                     core = c("EAST", "WEST", "CENTRE")))
lab <- scr_classing_accept(lab, p, reason = "edge/core is what pricing uses")
scr_classing_view(lab, "ds_region")
# a second proposal on the same variable, rejected with its reason
p2 <- scr_classing_propose(lab, "ds_region",
```

```

      groups = list(c("NORTH", "SOUTH", "EAST"), c("WEST", "CENTRE")))
lab <- scr_classing_discard(lab, p2, reason = "no business rationale for this grouping")
scr_decisions(lab)

```

scr_classing_apply *Commit the lab into a new selection result*

Description

Returns a new `scr_result` in which the accepted manual entries replace the optimal ones inside `fit` (the automatic fit is frozen as `fit_auto`), the screening and hold-out rows of those variables are recomputed with the very same pipeline functions, the final shortlist is the one implied by `scr_classing_choose()`, and the funnel, gains, SQL and summary are rebuilt with a provenance column. `scr_selected()` on the result returns the final list (which = "consensus" still gives the automatic one). The ledger travels with the result and into `scr_scorecard()` and `scr_export()`. The input result is not modified.

Usage

```
scr_classing_apply(lab)
```

Arguments

`lab` An object from `scr_coarse_classing()`.

Value

An `scr_result` with a `lab` component (ledger, spec, shortlist, source).

See Also

`scr_coarse_classing()` for a complete session, from lab to scorecard.

Other classing: `scr_classing_accept()`, `scr_classing_choose()`, `scr_classing_propose()`, `scr_classing_spec()`, `scr_classing_view()`, `scr_coarse_classing()`, `scr_decisions()`

Examples

```

cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
  use_lightgbm = FALSE, xgb_rounds = 40, n_boot = 10)
d <- scr_demo[, c("default", "ref_date", "ds_region", "ds_band", "vl_score_01",
  "vl_score_02", "vl_score_05", "vl_score_10", "vl_hist_01")]
res <- scr_select(d, "default", config = cfg, date_col = "ref_date")
lab <- scr_coarse_classing(res)
p <- scr_classing_propose(lab, "ds_region",
  groups = list(edge = c("NORTH", "SOUTH"),
    core = c("EAST", "WEST", "CENTRE")))
lab <- scr_classing_accept(lab, p, reason = "edge/core is what pricing uses")
res2 <- scr_classing_apply(lab)
scr_selected(res2)
scr_decisions(res2)

```

scr_classing_choose *Choose the final variable list manually*

Description

The final list is (consensus shortlist + force) - drop, then intersected with keep when given. force is allowed only for variables that reached binning; a variable failed for IV_SUSPICIOUS (the leakage ceiling) and a derived __sp flag under allow_derived_final = FALSE are refused unless override = TRUE. reason is one string for every variable named, or a character vector named by variable.

Usage

```
scr_classing_choose(
  lab,
  keep = NULL,
  drop = NULL,
  force = NULL,
  reason = NULL,
  override = FALSE
)
```

Arguments

lab	An object from scr_coarse_classing() .
keep	Variables to keep (restricts the final list).
drop	Variables to remove from the final list.
force	Variables to add to the final list.
reason	Mandatory when drop or force is given.
override	Allow a refused force.

Value

The updated lab, invisibly.

See Also

[scr_coarse_classing\(\)](#) for a complete session, from lab to scorecard.

Other classing: [scr_classing_accept\(\)](#), [scr_classing_apply\(\)](#), [scr_classing_propose\(\)](#), [scr_classing_spec\(\)](#), [scr_classing_view\(\)](#), [scr_coarse_classing\(\)](#), [scr_decisions\(\)](#)

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  use_lightgbm = FALSE, xgb_rounds = 40, n_boot = 10)
d <- scr_demo[, c("default", "ref_date", "ds_region", "ds_band", "vl_score_01",
                  "vl_score_02", "vl_score_05", "vl_score_10", "vl_hist_01")]
res <- scr_select(d, "default", config = cfg, date_col = "ref_date")
lab <- scr_coarse_classing(res)
lab <- scr_classing_choose(lab, drop = "vl_score_10",
                           reason = "not available at decision time")

lab
```

scr_classing_propose *Propose manual bins for a variable*

Description

Exactly one of breaks, groups, merge, split or reset per call; missing_to and other_to compose with a categorical instruction. The instruction is resolved against the current bins into an absolute specification, the WOE is refitted on the training rows, the bins are applied frozen to the hold-out, and the comparison against the optimal bins is printed. The proposal is a value: nothing changes in the lab until [scr_classing_accept\(\)](#).

Usage

```
scr_classing_propose(
  lab,
  variable,
  breaks = NULL,
  groups = NULL,
  merge = NULL,
  split = NULL,
  missing_to = NULL,
  other_to = NULL,
  reset = FALSE
)
```

Arguments

lab	An object from scr_coarse_classing() .
variable	A variable of the lab.
breaks	Numeric: interior cut points, $(-\text{Inf}, b_1]$, $(b_1, b_2]$,
groups	Categorical: a list of character vectors, one per bin; names are display labels. Every training category must be assigned, or other_to must name the catch-all bin.
merge	Bin ids to merge (adjacent for numerics).
split	<code>c(id, at)</code> : split numeric bin id at at.

missing_to	Categorical: fold the "MISSING" category into this bin.
other_to	Categorical: the bin that receives every training category not listed in groups.
reset	TRUE proposes a return to the optimal bins.

Value

An `scr_classing_proposal` with `id`, `variable`, `instruction` (the resolved instruction as text), `entry` (the hand-built bins), `checks`, `optimal` (the checks of the optimal bins), `compare`, `verdict` (ACCEPTABLE, REVIEW or BLOCKED), `warnings` and `blocking`.

See Also

[scr_coarse_classing\(\)](#) for a complete session, from lab to scorecard.
Other classing: [scr_classing_accept\(\)](#), [scr_classing_apply\(\)](#), [scr_classing_choose\(\)](#), [scr_classing_spec\(\)](#), [scr_classing_view\(\)](#), [scr_coarse_classing\(\)](#), [scr_decisions\(\)](#)

scr_classing_spec	<i>Classing specification as a long table, with its file round trip</i>
-------------------	---

Description

One row per bin of every variable in the lab, optimal and manual, with the authoritative columns a reviewer may edit (`lower/upper` for numerics, `categories/is_other` for categoricals, `reason`) and context columns that are regenerated on read. Open ends are written as NA. [scr_classing_read\(\)](#) validates a file back into a spec and [scr_classing_import\(\)](#) turns every variable whose bins differ from the lab's current ones into a proposal, so a spreadsheet edit never enters silently.

Usage

```
scr_classing_spec(lab, file = NULL)

scr_classing_read(file, sep = "%;%")

scr_classing_import(lab, file)
```

Arguments

lab	An object from scr_coarse_classing() , or an <code>scr_result</code> returned by scr_classing_apply() .
file	For scr_classing_spec() , an optional <code>.csv</code> or <code>.xlsx</code> path to write the table to. For scr_classing_read() , the path to read. For scr_classing_import() , a path or an <code>scr_classing_spec</code> object.
sep	Bin separator of the <code>categories</code> column (the configuration's <code>bin_separator</code>). It is validated (no empty category, no category in two bins) and recorded on the spec, so that scr_classing_import() refuses a spec read with a different separator.

Value

A data.frame of class `scr_classing_spec`.

`scr_classing_import()` returns a named list of proposals (one per variable whose bins differ from the lab's current ones), each to be accepted or discarded.

See Also

[scr_coarse_classing\(\)](#) for a complete session, from lab to scorecard.

Other classing: [scr_classing_accept\(\)](#), [scr_classing_apply\(\)](#), [scr_classing_choose\(\)](#), [scr_classing_propose\(\)](#), [scr_classing_view\(\)](#), [scr_coarse_classing\(\)](#), [scr_decisions\(\)](#)

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                 use_lightgbm = FALSE, xgb_rounds = 40, n_boot = 10)
d <- scr_demo[, c("default", "ref_date", "ds_region", "ds_band", "vl_score_01",
                 "vl_score_02", "vl_score_05", "vl_score_10", "vl_hist_01")]
res <- scr_select(d, "default", config = cfg, date_col = "ref_date")
lab <- scr_coarse_classing(res)
p <- scr_classing_propose(lab, "ds_region",
                        groups = list(edge = c("NORTH", "SOUTH"),
                                      core = c("EAST", "WEST", "CENTRE")))
lab <- scr_classing_accept(lab, p, reason = "edge/core is what pricing uses")
sp <- scr_classing_spec(lab)
sp
# round trip through a file: a fresh lab receives the manual bins as proposals
f <- tempfile(fileext = ".csv")
scr_classing_spec(lab, file = f)
props <- scr_classing_import(scr_coarse_classing(res), scr_classing_read(f))
names(props)
unlink(f)
```

`scr_classing_view`
Inspect the current bins of a variable in the lab

Description

Prints the current bins (optimal, or the accepted manual ones) with train and hold-out side by side and a text bar chart of the event rate, or, without variable, one line per variable of the lab.

Usage

```
scr_classing_view(lab, variable = NULL)
```

Arguments

`lab` An object from [scr_coarse_classing\(\)](#).

`variable` A variable name, or NULL for the overview.

Value

Invisibly, the bins table (variable given) or the overview table.

See Also

[scr_coarse_classing\(\)](#) for a complete session, from lab to scorecard.

Other classing: [scr_classing_accept\(\)](#), [scr_classing_apply\(\)](#), [scr_classing_choose\(\)](#), [scr_classing_propose\(\)](#), [scr_classing_spec\(\)](#), [scr_coarse_classing\(\)](#), [scr_decisions\(\)](#)

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                 use_lightgbm = FALSE, xgb_rounds = 40, n_boot = 10)
d <- scr_demo[, c("default", "ref_date", "ds_region", "ds_band", "vl_score_01",
                 "vl_score_02", "vl_score_05", "vl_score_10", "vl_hist_01")]
res <- scr_select(d, "default", config = cfg, date_col = "ref_date")
lab <- scr_coarse_classing(res)
scr_classing_view(lab)
scr_classing_view(lab, "ds_region")
```

<code>scr_coarse_classing</code>	<i>Coarse classing lab: manual binning and manual variable choice</i>
----------------------------------	---

Description

Opens a lab on an [scr_select\(\)](#) result. Inside it the analyst inspects the optimal bins of any binned variable ([scr_classing_view\(\)](#)), proposes new breaks or groupings ([scr_classing_propose\(\)](#)), reads the comparison against the optimal bins, accepts or discards each proposal with a mandatory reason ([scr_classing_accept\(\)](#), [scr_classing_discard\(\)](#)), chooses the final variable list ([scr_classing_choose\(\)](#)) and commits everything to a new `scr_result` ([scr_classing_apply\(\)](#)) that the rest of the pipeline consumes unchanged: [scr_scorecard\(\)](#), [scr_apply\(\)](#), [scr_sql\(\)](#), [scr_export\(\)](#).

Usage

```
scr_coarse_classing(
  x,
  features = NULL,
  laplace = 0,
  max_iv_loss = NULL,
  author = Sys.info()[["user"]]
)
```

Arguments

<code>x</code>	An object from <code>scr_select()</code> .
<code>features</code>	Variables the lab covers. Default: every variable that reached binning (<code>names(x\$fit\$results)</code>), so a variable failed by screening can be rebinned and forced in with a reason.
<code>laplace</code>	Smoothing added to the bin counts when recomputing WOE. 0 (default) is exactly the engine's formula.
<code>max_iv_loss</code>	Advisory threshold: a manual bin whose hold-out IV falls more than this fraction below the optimal one raises <code>IV_LOSS_VS_OPTIMAL</code> . NULL uses <code>config\$lab_max_iv_loss</code> .
<code>author</code>	Free text recorded in the ledger.

Value

An `scr_classing` object (the lab), with a print method that summarises the session: variables touched, before/after IV, verdicts, reasons, pending proposals and the final choice.

Contract of a manual bin

A manual bin is recomputed **on the training rows only** (hold-out rows can never define a bin), with the engine's own WOE formula ($\ln(\%event / \%non-event)$), event-oriented, so glm coefficients stay positive), then revalidated on the hold-out with the bins frozen (IV, PSI with both thresholds, unbinned share) and screened with the eight engine rules, so the lab and the pipeline can never disagree. Numeric intervals are right-closed, $(a, b]$, exactly as the engine and its SQL. Re-declaring the optimal cut points of a numeric reproduces the engine's WOE exactly; for a categorical the engine applies a small internal smoothing of its own, so the raw log-ratio of the lab differs from it in the third decimal.

What is never allowed silently

An empty bin, a degenerate bin (no events or no non-events, unless `laplace > 0`), a bin below `lab_min_bin_pct_hard`, a manual IV crossing `iv_max` (the lab must not manufacture leakage), a category left unassigned, a missing reason. Those block the proposal (BLOCKED); accepting one needs `override = TRUE`, and the override is itself a ledger row.

See Also

Other classing: `scr_classing_accept()`, `scr_classing_apply()`, `scr_classing_choose()`, `scr_classing_propose()`, `scr_classing_spec()`, `scr_classing_view()`, `scr_decisions()`

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
lab <- scr_coarse_classing(res)
lab
scr_classing_view(lab, "ds_region")
p <- scr_classing_propose(lab, "ds_region",
                          groups = list(edge = c("NORTH", "SOUTH"),
```

```

                                core = c("EAST", "WEST", "CENTRE"))
p
lab <- scr_classing_accept(lab, p, reason = "edge/core is what pricing uses")
lab <- scr_classing_choose(lab, drop = "v1_score_10",
                           reason = "not available at decision time")

lab
res2 <- scr_classing_apply(lab)
scr_selected(res2)
scr_selected(res2, which = "consensus")
sc <- scr_scorecard(res2)
sc$model_card$binning_algorithm

```

scr_compare

*Compare runs across targets***Description**

One row per target, with the funnel, the hold-out performance of the best model (with CI) and the warning signs.

Usage

```
scr_compare(x)
```

Arguments

x An object from [scr_run\(\)](#), or a named list of `scr_result`.

Value

A data.table with one row per successful target.

See Also

Other portfolio: [scr_core\(\)](#), [scr_run\(\)](#), [scr_runset](#)

Examples

```

cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
r1 <- scr_select(scr_demo, "default", config = cfg, drop = c("id", "churn"),
                 date_col = "ref_date")
r2 <- scr_select(scr_demo, "churn", config = cfg, drop = c("id", "default"),
                 date_col = "ref_date")
scr_compare(list(default = r1, churn = r2))
scr_core(list(default = r1, churn = r2), min_targets = 2)

```

scr_config	<i>Pipeline configuration</i>
------------	-------------------------------

Description

Builds the configuration object that crosses every stage, from `scr_split()` to `scr_export()`. A preset sets the tightness of the selection funnel; any individual key can be overridden through

Usage

```
scr_config(preset = c("moderate", "aggressive", "lazy"), ...)
```

Arguments

preset	One of "moderate" (default), "aggressive" or "lazy".
...	Overrides of any configuration key, by name. NULL means "keep the preset value", not "delete the key". An unknown name is an error, on purpose: a silent override leaves dead configuration in the file.

Value

An object of class `scr_config`: a named list with every key resolved.

Presets

A preset touches four keys and nothing else: `target_max`, `min_votes`, `corr_cutoff` and `iv_min`.

preset	variables at the end	min_votes	corr_cutoff	iv_min
"aggressive"	10 to 15	3	0.60	0.03
"moderate"	10 to 25	2	0.70	0.02
"lazy"	10 to 40	1	0.80	0.02

Use `scr_presets()` to see the resolved table and `scr_config_keys()` for the full key dictionary.

Risk or propensity (objective)

The two literatures use the same mathematics with opposite conventions. In credit and fraud, target = 1 is the **bad** case and the scorecard is built so that more points mean less risk. In propensity, target = 1 is the **good** case and the campaign list needs more points to mean a higher chance of engaging.

	"risk" (default)	"propensity"
target = 1 means	undesirable event	desirable event
Points scale	more points = safer	more points = more likely
Derived direction	"higher_is_safer"	"higher_is_riskier"
odds_orientation	safe:event	event:safe

objective **does not touch the selection**. It does not change the modelled target, the cut points, the IV or the shortlist; it acts on the direction of the points scale, on the odds orientation of the alignment and on the vocabulary of the reports. To model the other class as the event, the argument is `event_level` in `scr_split()` and `scr_select()`, and that one, unlike this, rewrites everything.

Binning algorithm (algorithm)

"jedi" is the default and stays exposed side by side with the alternatives, never hidden behind an "auto". Choices with distinct properties: "ivb", "dp" and "sblp" are provably optimal for categorical; "cm" (ChiMerge) and "fetb" have a principled stopping rule; "ir" (isotonic) guarantees monotone WOE; "fast_md1p" is the faithful Fayyad-Irani. The full list is in `OptimalBinningWoE::obwoe_algorithms()`. A numeric-only or categorical-only algorithm applies where it is valid and the other type falls back to "jedi".

The Information Value gate

`iv_min` Admission floor. Fails with IV_BELOW_MIN.

`iv_max` Admission ceiling. Fails with IV_SUSPICIOUS. 1.00 (default) tolerates a legitimately strong predictor and cuts the absurd; 0.50 is the engine default, calibrated for credit default.

`iv_suspect` Only the threshold of the report warning. Fails nothing.

The real leakage detector is `allow_degenerate = FALSE`: a bin with no events or no non-events is the symptom that has no innocent explanation.

Scorecard scale

`base_score`, `base_odds` and `pdo` are one statement: *at base_score points the odds are base_odds, and every pdo points they double*. `base_odds` is always expressed in the orientation direction implies (non-event:event under higher_is_safer; event:non-event under higher_is_riskier), and the alignment object records `odds_orientation` so this is never implicit. The classic 600/50/20 is Siddiqi's (2006) textbook example, not a parameter published by any bureau. `align_method = "regression"` (default) takes the raw score to that scale by regressing empirical log-odds on score bands, which absorbs reweighting, miscalibration and prior shift; "direct" assumes the model is calibrated and uses the logit as is.

References

Siddiqi, N. (2006). *Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring*. Wiley.

See Also

`scr_select()` to use the configuration, `scr_presets()` to compare presets, `scr_config_keys()` for the key dictionary.

Other configuration: `scr_config_keys()`, `scr_presets()`, `scr_verbose()`

Examples

```

cfg <- scr_config()
cfg

# propensity: more points = more likely to have the event
scr_config(objective = "propensity")$objective

# NULL keeps the preset value (here, iv_min = 0.03 from aggressive)
scr_config("aggressive", iv_min = NULL)$iv_min

# a wrong name fails loudly instead of becoming dead configuration
try(scr_config(iv_maximum = 1))

```

scr_config_keys	<i>Dictionary of configuration keys</i>
-----------------	---

Description

One row per [scr_config\(\)](#) key, with the stage it acts on, the default value and what it controls.

Usage

```
scr_config_keys(stage = NULL)
```

Arguments

stage	Optional filter by stage: 0 to 7 for the scorecard pipeline, 8 to 12 for the IRB models. NULL returns everything.
-------	---

Value

A data.frame with key, stage, default and description.

See Also

Other configuration: [scr_config\(\)](#), [scr_presets\(\)](#), [scr_verbose\(\)](#)

Examples

```

head(scr_config_keys(), 8)
scr_config_keys(stage = 5)

```

scr_connect

*Connect to a database (ODBC DSN or any DBI driver)***Description**

With dsn, a thin wrapper around `DBI::dbConnect()` over `odbc::odbc()` with one deliberate choice: `bigint = "numeric"`. Under the **odbc** default a BIGINT column arrives as integer64, and `is.numeric()` of an integer64 is FALSE: typing would treat the column as a categorical of very high cardinality. With driver, any DBI driver object is accepted (e.g. `RSQLite::SQLite()`, `duckdb::duckdb()`), which is how the database path is tested without a DSN.

Usage

```
scr_connect(dsn = NULL, driver = NULL, timeout = 20, ...)
```

Arguments

dsn	Name of the DSN configured on the system. Ignored when driver is given.
driver	Optional DBI driver object, used instead of ODBC.
timeout	Connection timeout in seconds (ODBC only).
...	Extra arguments passed on to <code>DBI::dbConnect()</code> (for example <code>dbname = ":memory:"</code> for SQLite).

Value

A DBI connection. Close it with `DBI::dbDisconnect()`.

See Also

Other database: `scr_fetch()`

Examples

```
con <- scr_connect(driver = RSQLite::SQLite(), dbname = ":memory:")
d <- scr_demo; d$ref_date <- as.character(d$ref_date) # SQLite has no Date type
DBI::dbWriteTable(con, "dtm", d)
dt <- scr_fetch(con, "dtm", sample_frac = 0.5, seed = 42)
nrow(dt)
DBI::dbDisconnect(con)
```

scr_core

*Variables that cross several targets***Description**

Which variables were approved on how many targets. A stable core across targets is the best argument in favour of a variable.

Usage

```
scr_core(x, min_targets = 2L)
```

Arguments

x An object from [scr_run\(\)](#), or a named list of `scr_result`.
min_targets Minimum number of targets to enter the result.

Value

A `data.table` with `feature`, `n_targets`, `targets` and `mean_rank`.

See Also

Other portfolio: [scr_compare\(\)](#), [scr_run\(\)](#), [scr_runset](#)

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
r1 <- scr_select(scr_demo, "default", config = cfg, drop = c("id", "churn"),
                 date_col = "ref_date")
r2 <- scr_select(scr_demo, "churn", config = cfg, drop = c("id", "default"),
                 date_col = "ref_date")
scr_core(list(default = r1, churn = r2), min_targets = 2)
```

scr_cutoff

*Stage 6: cut-off sweep with frozen cuts***Description**

For each candidate cut, what happens in each sample: the fraction of the population on the safe side (approval), the event rate on both sides, the events avoided (share of events falling on the risky side), the non-events lost and the KS at the cut. The candidate cuts are quantiles of the score **on train**, applied frozen to the hold-out: both samples answer on the same numbers, and the comparison between them measures the stability of the decision, not a sample difference.

Usage

```
scr_cutoff(x, n_cuts = NULL, cuts = NULL)
```

Arguments

x	An object from scr_scorecard() .
n_cuts	Number of candidate cuts. NULL uses config\$cutoff_n.
cuts	Explicit vector of cuts; overrides n_cuts.

Details

The "safe side" is the high-score side under higher_is_safer (credit) and the low-score side under higher_is_riskier (fraud, propensity).

Value

An scr_cutoff object with table (one row per sample and cut) and direction.

See Also

Other stages: [scr_align\(\)](#), [scr_bin\(\)](#), [scr_model\(\)](#), [scr_reject\(\)](#), [scr_scorecard\(\)](#), [scr_select\(\)](#), [scr_split\(\)](#), [scr_strategy\(\)](#), [scr_triage\(\)](#)

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
sc <- scr_scorecard(res)
ct <- scr_cutoff(sc, n_cuts = 10)
ct
st <- scr_strategy(sc, revenue_good = 1080, loss_bad = 4500)
st
rj <- scr_reject(sc)
rj
```

scr_decisions

Decision ledger of a lab, a result or a scorecard

Description

Returns the append-only ledger of manual decisions: every proposal accepted, discarded or superseded, every forced or dropped variable and every override, each with its reason.

Usage

```
scr_decisions(x)
```

Arguments

x An `scr_classing` lab, an `scr_result` from `scr_classing_apply()` or an `scr_scorecard` fitted on one.

Value

A `data.table`, one row per decision (append-only), or an empty one when no manual decision exists.

See Also

`scr_coarse_classing()` for a complete session, from lab to scorecard.

Other classing: `scr_classing_accept()`, `scr_classing_apply()`, `scr_classing_choose()`, `scr_classing_propose()`, `scr_classing_spec()`, `scr_classing_view()`, `scr_coarse_classing()`

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                 use_lightgbm = FALSE, xgb_rounds = 40, n_boot = 10)
d <- scr_demo[, c("default", "ref_date", "ds_region", "ds_band", "vl_score_01",
                 "vl_score_02", "vl_score_05", "vl_score_10", "vl_hist_01")]
res <- scr_select(d, "default", config = cfg, date_col = "ref_date")
lab <- scr_coarse_classing(res)
p <- scr_classing_propose(lab, "ds_region",
                        groups = list(edge = c("NORTH", "SOUTH"),
                                       core = c("EAST", "WEST", "CENTRE")))
lab <- scr_classing_accept(lab, p, reason = "edge/core is what pricing uses")
scr_decisions(lab)
scr_decisions(scr_classing_apply(lab))
scr_decisions(res) # no manual decision: an empty ledger
```

scr_default

Build the default flag from a monthly panel

Description

Applies the standard definition of default to a panel with one row per unit (id) and month (date): a unit enters default when `dpd` reaches `default_days` and the arrears are material (both `default_abs` in currency units and `default_rel` as a share of exposure, when arrears and exposure are supplied), or when `utp` (unlikeliness to pay) is `TRUE`. It leaves default after `default_probation` consecutive months without a trigger (`default_probation_restructured` when restructured was `TRUE` at any point of the event). With obligor supplied and `default_level = "obligor"`, a unit whose obligor has more than `default_pulling` of its exposure in default is pulled into default too, and a defaulted obligor defaults all its units.

Usage

```
scr_default(
  data,
  id,
  date,
  dpd = NULL,
  arrears = NULL,
  exposure = NULL,
  utp = NULL,
  restructured = NULL,
  obligor = NULL,
  config = scr_config()
)
```

Arguments

<code>data</code>	A <code>data.frame</code> or <code>data.table</code> , one row per <code>id</code> and <code>date</code> .
<code>id, date</code>	Column names of the unit identifier and the month.
<code>dpd</code>	Column name of days past due (integer). Optional when <code>utp</code> is given.
<code>arrears, exposure</code>	Column names of the overdue amount and the total exposure, both optional; when given, the materiality test applies.
<code>utp</code>	Column name of a logical unlikeliness-to-pay flag, optional.
<code>restructured</code>	Column name of a logical distressed-restructuring flag, optional.
<code>obligor</code>	Column name of the obligor when <code>id</code> is a facility, optional; enables the pulling effect.
<code>config</code>	A <code>scr_config()</code> ; keys <code>default_*</code> .

Details

Rows must be monthly; gaps are tolerated (the probation counts observed months). The result keeps the row-level flags: they are the product.

Value

An object of class `scr_default`: `flags` (`id`, `date`, `default` 0/1, `event_id`, `trigger`, `months_in_default`, `cured`), `events` (one row per event: `event_id`, `id`, `start`, `end`, `trigger`, `cured`, `months`), `summary`, `ledger` and `config`.

See Also

Other `irb`-parameters: `scr_default_rate()`, `scr_irb_params()`

Examples

```
d <- scr_default(scr_demo_panel, id = "id", date = "ref_date", dpd = "dpd",
  arrears = "arrears", exposure = "exposure",
  restructured = "restructured",
  config = scr_config(verbose = FALSE))

d
head(d$events)
```

scr_default_rate	<i>One-year default rates by cohort and the long-run average</i>
------------------	--

Description

From a flagged panel (an `scr_default()` object or any table with a 0/1 default column by unit and month), computes for every cohort start the population of non-defaulted units, the share that defaults within horizon months, optionally by grade or segment (as observed at the cohort start) and exposure-weighted when exposure is given. The long-run average is the arithmetic mean of the cohort rates. When the analyst proposes an adjusted value (`lra_adjusted`, for instance after judging that the period lacks bad years), it is benchmarked against the larger of the last five years' mean and the whole period's mean, and a flag records when it sits below that benchmark.

Usage

```
scr_default_rate(
  x,
  id = "id",
  date = "date",
  default = "default",
  horizon = 12L,
  by = NULL,
  grade = NULL,
  segment = NULL,
  exposure = NULL,
  lra_adjusted = NULL,
  config = scr_config()
)
```

Arguments

<code>x</code>	An <code>scr_default</code> or a <code>data.frame/data.table</code> .
<code>id, date, default</code>	Column names (ignored for an <code>scr_default</code>).
<code>horizon</code>	Months of the default window after the cohort start.
<code>by</code>	Cohort frequency: "month", "quarter" or "year".
<code>grade, segment</code>	Optional column names observed at the cohort start.
<code>exposure</code>	Optional column name; adds exposure-weighted rates.

lra_adjusted	Optional adjusted long-run average proposed by the analyst, in [0, 1]; benchmarked and flagged, never applied.
config	A scr_config() ; only pd_dr_by is read (the default of by).

Value

An object of class `scr_dr`: table (cohort rates, by grade or segment when given), portfolio (one row per cohort: n, defaults, dr), lra (mean, weighted_mean, recent5_mean, benchmark, adjusted, flag_below_benchmark, min, max, sd, n_cohorts, years), horizon and by.

See Also

Other irb-parameters: [scr_default\(\)](#), [scr_irb_params\(\)](#)

Examples

```
d <- scr_default(scr_demo_panel, id = "id", date = "ref_date", dpd = "dpd",
                config = scr_config(verbose = FALSE))
dr <- scr_default_rate(d, by = "quarter")
dr
dr$table
```

scr_demo	<i>Synthetic example data</i>
----------	-------------------------------

Description

A fabricated table so that every example and every vignette runs without a database. On purpose, it carries the defects real data has: sentinel -999 in several masses (`vl_hist_*`), missing values (`vl_partial_*`), a column that only degrades in the last period (`vl_late`), constant, near-constant, exact duplicate, high cardinality, a redundant pair and pure noise. Without them, the audit funnel would have nothing to show.

Usage

```
scr_demo
```

Format

- A data.frame with 4,200 rows and 41 columns:
- `id` Identifier (never a candidate; goes in drop).
 - `ref_date` Monthly reference date, six periods; the key of the out-of-time split.
 - `vl_score_01 to vl_score_12` Numerics with decreasing signal.
 - `vl_hist_01 to vl_hist_05` Numerics with an increasing mass of sentinel -999; the signal is in the absence.
 - `vl_partial_01 to vl_partial_03` Numerics with genuine NA.

vl_late Numeric with NA/sentinel in the last period only.
 vl_noise_01 to vl_noise_06 Pure noise.
 vl_constant, vl_near_const, ds_constant, vl_duplicate, vl_redundant, ds_high_card Structural pathologies.
 ds_region, ds_band, ds_channel, ds_optin Categoricals with signal; ds_optin has NA.
 default Risk target (0/1, about 14% events).
 churn Propensity target (0/1, about 29% events), for the portfolio examples.

Source

Synthetic. Generated by data-raw/scr_demo.R, seed 20260903, in the package source repository <https://github.com/evandeilton/scorecraft>.

See Also

Other data: [scr_demo_ead](#), [scr_demo_lgd](#), [scr_demo_lgd_cashflows](#), [scr_demo_panel](#), [scr_demo_portfolio](#), [scr_demo_rates](#)

Examples

```
str(scr_demo[, 1:6])
mean(scr_demo$default)
```

scr_demo_ead

Synthetic monthly facility snapshots for the EAD/CCF module

Description

1,200 revolving facilities (cards, overdrafts and revolving lines) of 950 obligors, observed monthly (snapshots dated the first day of the month) over 30 months. Utilisation follows a facility-specific level with a mild drift; about 10% of the facilities default, with the utilisation ramping up over the twelve months before the default month at an intensity that depends on the drivers a CCF model is expected to find (utilisation, product, months on book, days past due). Some defaulters are fully drawn or over the limit at the reference date, some repay before default (negative realised CCF), some have their limit cut before default, a share of all facilities gets a limit change and a few facilities originate inside the window (fast defaults). Built for [scr_ead_data\(\)](#), [scr_ead\(\)](#) and [scr_ead_validate\(\)](#).

Usage

```
scr_demo_ead
```

Format

A data.frame with 34,811 rows and 9 columns:

facility_id Facility identifier.
 obligor_id Obligor identifier (a few obligors hold two facilities).
 ref_date Month (first day), 30 periods from 2023-01.
 limit Limit of the facility at the month end.
 drawn Drawn amount at the month end, gross; may exceed the limit.
 product "card", "overdraft" or "line".
 months_on_book Age of the facility in months.
 dpd Days past due at the month end (multiples of 30).
 defaulted 0/1 flag, 1 from the default month onwards.

Source

Synthetic. Generated by data-raw/scr_demo_ead.R, seed 20260904, in the package source repository <https://github.com/evandeilton/scorecraft>.

See Also

Other data: [scr_demo](#), [scr_demo_lgd](#), [scr_demo_lgd_cashflows](#), [scr_demo_panel](#), [scr_demo_portfolio](#), [scr_demo_rates](#)

Examples

```
head(scr_demo_ead)
length(unique(scr_demo_ead$facility_id[scr_demo_ead$defaulted == 1]))
```

scr_demo_lgd

Synthetic default events for the workout LGD examples

Description

900 default events on three products (unsecured, mortgage, auto) observed until 2026-06-30, with the drivers a workout LGD model reads: collateral, loan-to-value, time on book, the worst delinquency before default and the region. Cures return to performing within six months; non-cures recover according to product-specific profiles until a close date; events still running at the observation date are open, a few of them older than the maximum recovery period. Thirty facilities default twice after a cure, half of them within nine months, so that [scr_workout\(\)](#) merges the two spells into one event. Built for [scr_workout\(\)](#), [scr_lgd\(\)](#) and the in-default examples.

Usage

```
scr_demo_lgd
```

Format

A data.frame with 900 rows and 12 columns:

default_id Default event identifier.

facility_id Facility identifier; repeated for the second defaults.

default_date Date of default.

ead Exposure at default.

product "unsecured", "mortgage" or "auto".

collateral_value Collateral value at default; 0 when unsecured.

ltv Loan-to-value at default; 0 when unsecured.

months_on_book Months since origination at default.

prior_dpd_max Worst days past due in the year before default (multiples of 30).

region Region of the facility.

status "closed", "cured" or "open" at the observation date.

close_date Date the workout closed or the cure was confirmed; NA when open.

Source

Synthetic. Generated by data-raw/scr_demo_lgd.R, seed 20260904, in the package source repository <https://github.com/evandeilton/scorecraft>.

See Also

Other data: [scr_demo](#), [scr_demo_ead](#), [scr_demo_lgd_cashflows](#), [scr_demo_panel](#), [scr_demo_portfolio](#), [scr_demo_rates](#)

Examples

```
table(scr_demo_lgd$product, scr_demo_lgd$status)
```

```
scr_demo_lgd_cashflows
```

Synthetic post-default cash flows of scr_demo_lgd

Description

Long table of the cash flows of every default event of [scr_demo_lgd](#): recoveries spread over up to four years for unsecured facilities, a collateral sale between one and three years in for mortgages, a repossession sale within nine months for cars, direct costs at the start of the workout and at the sale, and a few drawings after default. Cures carry only the payments that brought the facility back to performing.

Usage

```
scr_demo_lgd_cashflows
```

Format

A data.frame with 5,737 rows and 4 columns:

default_id Default event identifier, matching [scr_demo_lgd](#).

date Date of the cash flow.

amount Amount, always positive.

type "recovery", "direct_cost" or "drawing".

Source

Synthetic. Generated by data-raw/scr_demo_lgd.R, seed 20260904, in the package source repository <https://github.com/evandeilton/scorecraft>.

See Also

Other data: [scr_demo](#), [scr_demo_ead](#), [scr_demo_lgd](#), [scr_demo_panel](#), [scr_demo_portfolio](#), [scr_demo_rates](#)

Examples

```
head(scr_demo_lgd_cashflows)
table(scr_demo_lgd_cashflows$type)
```

scr_demo_panel

Synthetic monthly panel for the default engine and PD calibration

Description

600 obligors observed over 36 months. Days past due evolve as a chain whose slip probability depends on a latent risk; arrears are proportional to the exposure, with a few obligors whose arrears stay below the absolute materiality threshold on purpose; 25 obligors are restructured from month 18; score is a behavioural score (higher = safer) with genuine rank-ordering power. Built for [scr_default\(\)](#), [scr_default_rate\(\)](#) and the PD grade examples.

Usage

```
scr_demo_panel
```

Format

A data.frame with 21,600 rows and 7 columns:

id Obligor identifier.

ref_date Month (first day), 36 periods from 2023-01.

dpd Days past due at the snapshot (multiples of 30).

arrears Overdue amount.

exposure Total exposure of the obligor (constant).
 restructured Logical distressed-restructuring flag.
 score Behavioural score, higher is safer.

Source

Synthetic. Generated by data-raw/scr_demo_panel.R, seed 20260904, in the package source repository <https://github.com/evandeilton/scorecraft>.

See Also

Other data: [scr_demo](#), [scr_demo_ead](#), [scr_demo_lgd](#), [scr_demo_lgd_cashflows](#), [scr_demo_portfolio](#), [scr_demo_rates](#)

Examples

```
head(scr_demo_panel)
mean(scr_demo_panel$dpd >= 90)
```

scr_demo_portfolio	<i>Synthetic exposure snapshot for expected loss, capital and ECL</i>
--------------------	---

Description

5,000 exposures in six segments that map one-to-one to the asset classes of the IRB risk-weight function. The PD is a grade PD (constant within asset class and grade; the ten geometric grades start below the regulatory floors on purpose) and the LGD a pool value constant within the segment, so that segment by grade is a homogeneous pool and the production SQL of [scr_capital\(\)](#) reproduces R exactly. About 3 % of the exposures are in default with a best estimate of expected loss and a provision close to it; the other columns feed the standardised comparison and the accounting stage rule of [scr_ecl\(\)](#).

Usage

```
scr_demo_portfolio
```

Format

A data.frame with 5,000 rows and 21 columns:

id Exposure identifier.
 segment Reporting segment (retail_loans, mortgages, cards_revolver, cards_transactor, corporate_large, corporate_sme).
 asset_class Asset class of the risk-weight function.
 grade PD grade, G01 (safest) to G10.
 pd One-year PD of the grade (decimal).

lgd Downturn LGD of the pool (decimal).
ead, drawn, undrawn Exposure at default and its split.
m Effective maturity in years (corporates only).
sales Annual sales in millions (corporates only).
ltv Loan-to-value at origination (mortgages only).
rating External rating (large corporates; NA unrated).
transactor Logical: revolving facility repaid in full monthly.
defaulted 0/1 default flag (about 3 %).
elbe Best estimate of expected loss of the defaulted rows.
provision Provision stock.
stage Accounting stage 1, 2 or 3.
dpd Days past due.
pd_orig One-year PD at origination.
eir Annual effective interest rate.

Source

Synthetic. Generated by data-raw/scr_demo_portfolio.R, seed 20260904, in the package source repository <https://github.com/evandeilton/scorecraft>.

See Also

Other data: [scr_demo](#), [scr_demo_ead](#), [scr_demo_lgd](#), [scr_demo_lgd_cashflows](#), [scr_demo_panel](#), [scr_demo_rates](#)

Examples

```
head(scr_demo_portfolio)
table(scr_demo_portfolio$asset_class)
```

scr_demo_rates	<i>Synthetic monthly reference rate series</i>
----------------	--

Description

A smooth annualised reference rate by month from 2019-01 to 2026-06, falling to two per cent in 2020-2021 and rising above thirteen per cent in 2022-2023, used by [scr_workout\(\)](#) as the rate at the default date.

Usage

```
scr_demo_rates
```

Format

A data.frame with 90 rows and 2 columns:

date First day of the month.

rate Annual reference rate as a decimal.

Source

Synthetic. Generated by data-raw/scr_demo_lgd.R, seed 20260904, in the package source repository <https://github.com/evandeilton/scorecraft>.

See Also

Other data: [scr_demo](#), [scr_demo_ead](#), [scr_demo_lgd](#), [scr_demo_lgd_cashflows](#), [scr_demo_panel](#), [scr_demo_portfolio](#)

Examples

```
range(scr_demo_rates$rate)
```

scr_ead

Estimate CCF pools from the reference data set

Description

Splits the reference data set by reference date (the most recent cohorts form the hold-out), bins every driver against the realised CCF with the continuous binner ([scr_bin_continuous\(\)](#)) on the training rows, revalidates the frozen bins on the hold-out and admits a driver when it passes the named rules TOO_FEW_DEFAULTS, NO_SEPARATION, NOT_MONOTONIC and UNSTABLE_HOLDOUT. The cells of the cross of the admitted drivers are ordered by their predicted CCF and merged, adjacent cells first, down to config\$ccf_n_pools pools with at least config\$ccf_min_defaults defaults each. Rows in the limit-factor measure form their own pool LF.

Usage

```
scr_ead(x, drivers, config = scr_config(), holdout = 0.3, params = NULL)
```

Arguments

x	An scr_ead_data() object.
drivers	Column names of the candidate drivers (columns of x\$rds).
config	A scr_config() .
holdout	Hold-out share, by whole reference dates.
params	An scr_irb_params() object; NULL uses the preset of config\$framework.

Details

Per pool the estimate is the long-run (default-weighted) average of the realised values on the training rows, `lra`; `moc_est` is the one-sided normal estimation-error margin at `config$ccf_moc_alpha`; `ccf_dt` is the downturn value (equal to `lra` until `scr_ead_downturn()` is run); `ccf_final` = `max(lra, ccf_dt) + moc_est`; `ccf_floor` = `params$ccf_floor_fraction * config$ccf_sa_ccf`; and `ccf_applied` = `max(ccf_final, ccf_floor)`. For the LF pool the floor depends on the utilisation and is applied per row by `scr_apply()`.

Value

An object of class `scr_ead`: `pools` (the pool table), `cells` (every cell of the cross with its pool), `bins` (the obwoe-shaped fit of the admitted drivers), `bins_all` (the fit of every driver), `drivers` (admission table), `holdout` (frozen bins on the hold-out), `rds` (the reference rows with sample and pool), `metrics` (per sample: `rmse`, `mae`, `gauc` with a bootstrap interval, `spearman`, `ead_rmse`, `ead_mae`, `adequacy`, `cear`), `split`, `funnel`, `data_summary`, `downturn`, `ledger`, `model_card`, `params`, `config`, `meta`. Also `survivors` (the admitted drivers) and `lra` (the long-run averages of the data set); `metrics` also carries `n`, `n_main`, `gauc_se`, `somers_d` and `share_floor_binding`.

See Also

Other `irb-ead`: `scr_bin_continuous()`, `scr_ead_data()`, `scr_ead_downturn()`, `scr_ead_validate()`

Examples

```
cfg <- scr_config(verbose = FALSE, n_boot = 20, nthread = 1)
ed <- scr_ead_data(scr_demo_ead, facility_id = "facility_id", date_col = "ref_date",
                  limit = "limit", drawn = "drawn", defaulted = "defaulted",
                  drivers = c("product", "months_on_book", "dpd"), config = cfg)
m <- scr_ead(ed, drivers = c("utilisation_ref", "product", "months_on_book"), config = cfg)
m
m$pools
m$drivers
```

scr_ead_data

Build the realised-CCF reference data set from facility snapshots

Description

One row per default event (or per event and reference date under the variable-horizon comparison), with the facility as it stood at the reference date and the realised exposure at default (EAD) at the default date, from which the realised credit conversion factor (CCF) follows. The reference date follows `config$ccf_horizon`: "fixed" takes the snapshot `ccf_horizon_months` before the default month (the nearest earlier snapshot when that month is missing; the first snapshot for a facility younger than the horizon, flagged `FAST_DEFAULT`); "cohort" takes the start of the calendar cohort window in which the default falls; "variable" takes every snapshot in the horizon before the default, for comparison only.

Usage

```
scr_ead_data(
  snapshots,
  facility_id,
  obligor_id = NULL,
  date_col,
  limit,
  drawn,
  default_date = NULL,
  defaulted = NULL,
  drivers = NULL,
  config = scr_config(),
  keep_rows = FALSE
)
```

Arguments

snapshots	A data.frame or data.table with one row per facility and month.
facility_id, date_col, limit, drawn	Column names of the facility identifier, the snapshot month, the limit and the drawn amount.
obligor_id	Column name of the obligor, optional. With config\$default_level = "obligor" a default of any facility of the obligor is a default of all its facilities observed at that date.
default_date	Either the name of a column of snapshots holding the default date of the facility (NA when it never defaults) or a data.frame with the facility identifier column (same name as facility_id) and a default_date column, one row per event.
defaulted	Column name of a 0/1 default flag per snapshot, alternative to default_date: every run of ones opens an event at its first month.
drivers	Column names measured at the reference date and carried into the data set (candidate drivers of the pools). utilisation_ref, limit_ref, drawn_ref and horizon_months are always available.
config	A scr_config() ; keys ccf_*, post_default_drawings_in, default_level.
keep_rows	If TRUE, keeps every candidate event with its exclusion rule in rows.

Details

The realised measure per row follows config\$ccf_measure: under "auto" the undrawn-limit factor (CCF) when the utilisation at the reference date is below ccf_u_star and the limit factor (LF) at or above it; rows with nothing undrawn or over the limit at the reference date are always routed to the limit factor (ZERO_UNDRAWN, OVER_LIMIT_AT_REF), never dropped. The raw realised value is kept in ccf_raw; ccf carries the value after the optional floor (ccf_floor_realised) and cap (ccf_cap_realised), both logged in the funnel (NEGATIVE_CCF_FLOORED, CCF_ABOVE_ONE). The realised EAD is the drawn amount at the default date, uncapped; with post_default_drawings_in = "ccf" it is the maximum drawn amount over the default event when defaulted is given.

Value

An object of class `scr_ead_data`: `rds` (one row per event: `event_id`, `facility_id`, `obligor_id`, `ref_date`, `default_date`, `cohort`, `horizon_months`, `fast_default`, `limit_ref`, `drawn_ref`, `undrawn_ref`, `utilisation_ref`, `limit_default`, `limit_change`, `ead_realised`, `measure`, `ccf_raw`, `ccf`, `rule`, `drivers`), `funnel` (`rule`, `action`, `n`, `share`), `summary` (simple and exposure-weighted averages by cohort and measure, with a total row), `lra` (long-run averages and shares), `meta`, `ledger`, `config` and `rows` (with `keep_rows = TRUE`).

References

Basel Committee on Banking Supervision (2023). *The Basel Framework*, CRE32 and CRE36. Moral, G. (2006). EAD estimates for facilities with explicit limits. In Engelmann, B. and Rauhmeier, R. (eds), *The Basel II Risk Parameters*. Springer.

See Also

Other irb-ead: `scr_bin_continuous()`, `scr_ead()`, `scr_ead_downturn()`, `scr_ead_validate()`

Examples

```
cfg <- scr_config(verbose = FALSE)
ed <- scr_ead_data(scr_demo_ead, facility_id = "facility_id", obligor_id = "obligor_id",
                  date_col = "ref_date", limit = "limit", drawn = "drawn",
                  defaulted = "defaulted",
                  drivers = c("product", "months_on_book", "dpd"), config = cfg)

ed
ed$funnel
head(ed$rds[, c("facility_id", "ref_date", "default_date", "utilisation_ref", "measure", "ccf")])
```

scr_ead_downturn	<i>Downturn CCF per pool</i>
------------------	------------------------------

Description

Quantifies the downturn component of the CCF from user-supplied downturn periods. "type1" (observed impact) takes, per pool, the default-weighted average of the realised values of the training events whose default date falls in the periods (the hold-out stays independent) and sets `ccf_dt = max(lra, observed)`; "type3" (long-run average plus add-on) sets `ccf_dt = lra + add_on`; "none" resets `ccf_dt = lra`. The pool table is recomputed (`ccf_final`, `ccf_applied`) and the ledger records the periods, the method and the reason.

Usage

```
scr_ead_downturn(
  x,
  periods = NULL,
  method = NULL,
  add_on = 0.15,
```

```
    reason = NULL
  )
```

Arguments

x	An scr_ead() object.
periods	A data.frame with start and end dates of the downturn periods (needed for "type1").
method	"type1", "type3" or "none"; NULL uses config\$ccf_downturn.
add_on	Add-on of the "type3" method, in CCF units.
reason	Text justifying the periods and the method; mandatory.

Value

The scr_ead object with downturn (a list with method, periods, add_on and the per-pool table: pool, lra, n_downturn, dt_observed, dt_type3, ccf_dt), the updated pools and a new ledger row. The table also carries ccf_final and ccf_applied; the object n_rows_in_periods and reason.

See Also

Other irb-ead: [scr_bin_continuous\(\)](#), [scr_ead\(\)](#), [scr_ead_data\(\)](#), [scr_ead_validate\(\)](#)

Examples

```
cfg <- scr_config(verbose = FALSE, n_boot = 20, nthread = 1)
ed <- scr_ead_data(scr_demo_ead, facility_id = "facility_id", date_col = "ref_date",
  limit = "limit", drawn = "drawn", defaulted = "defaulted",
  drivers = c("product", "months_on_book"), config = cfg)
m <- scr_ead(ed, drivers = c("utilisation_ref", "product"), config = cfg)
m2 <- scr_ead_downturn(m, periods = data.frame(start = as.Date("2024-01-01"),
  end = as.Date("2024-12-01")),
  reason = "2024 chosen as the stress year of the demo panel")
m2$downturn$table
```

scr_ead_validate	<i>Validate CCF pools: calibration, discrimination, back-testing and stability</i>
------------------	--

Description

Per pool and in total, compares realised and predicted values on the validation rows (the hold-out of the model by default): simple and exposure-weighted averages, the one-sided t-test of realised above predicted (under-estimation) with its p-value, the EAD adequacy ratio (sum of realised EAD over sum of predicted EAD) and traffic lights (red at or below lights[1], amber at or below lights[2], green above; adequacy green at or below adequacy_lights[1], amber up to adequacy_lights[2], red above). Adds the discrimination block (gAUC with a bootstrap interval

against the development value, Spearman correlation, cumulative EAD accuracy ratio), the back-test by cohort and the stability of the pool distribution and of the driver bins ([scr_psi\(\)](#), fixed and sample-size-adjusted thresholds). The numeric limits of the lights are a convention of the package, stated as such in the output.

Usage

```
scr_ead_validate(
  x,
  newdata = NULL,
  lights = c(0.01, 0.05),
  adequacy_lights = c(1, 1.05)
)
```

Arguments

<code>x</code>	An scr_ead() object.
<code>newdata</code>	NULL (the hold-out rows of <code>x</code>), an scr_ead_data() object or its rds table.
<code>lights</code>	Two increasing p-value thresholds: red at or below the first, amber at or below the second.
<code>adequacy_lights</code>	Two increasing adequacy-ratio thresholds.

Value

An object of class `scr_ead_validation`: calibration, discrimination, backtest, stability, summary (test, statistic, p, light), n, source.

See Also

Other irb-ead: [scr_bin_continuous\(\)](#), [scr_ead\(\)](#), [scr_ead_data\(\)](#), [scr_ead_downturn\(\)](#)

Examples

```
cfg <- scr_config(verbose = FALSE, n_boot = 20, nthread = 1)
ed <- scr_ead_data(scr_demo_ead, facility_id = "facility_id", date_col = "ref_date",
  limit = "limit", drawn = "drawn", defaulted = "defaulted",
  drivers = c("product", "months_on_book"), config = cfg)
m <- scr_ead(ed, drivers = c("utilisation_ref", "product"), config = cfg)
v <- scr_ead_validate(m)
v
v$calibration
```

scr_ecl

*Expected credit loss with stage allocation***Description**

Discrete-time expected credit loss of every exposure:

Usage

```
scr_ecl(
  pd_term,
  lgd,
  ead,
  eir = 0,
  stage = NULL,
  dpd = NULL,
  pd_orig = NULL,
  scenarios = NULL,
  weights = NULL,
  prepay = NULL,
  rho = 0.15,
  t_max = NULL,
  segment = NULL,
  id = NULL,
  config = scr_config(),
  keep_rows = FALSE
)
```

Arguments

pd_term	Marginal monthly PDs: a matrix $n \times T$, or a vector of length n (a flat hazard recycled over t_max months), or a single number.
lgd, ead	Vectors of length n (or one) or matrices $n \times T$.
eir	Annual effective interest rate, vector of length n or one.
stage	Optional stage vector (1, 2, 3); NULL applies the rule.
dpd	Days past due (the rule); optional.
pd_orig	12-month PD at origination (the rule); optional.
scenarios	Named list of scenario shocks (see Details); NULL runs the base case only.
weights	Scenario weights; equal when NULL.
prepay	Monthly prepayment hazard: NULL, a vector or an $n \times T$ matrix.
rho	Asset correlation used by scenario shocks with z .
t_max	Term in months when <code>pd_term</code> is a vector (default <code>config\$ecl_horizon_months</code>).
segment	Optional character vector of length n for a segment summary.

id	Optional identifier vector of length n.
config	An scr_config() object (ecl_* keys, verbose).
keep_rows	Keep the per-exposure table.

Details

$$ECL_H = \sum_{t=1}^H S(t-1) h_t LGD_t EAD_t (1+r)^{-t/12}, \quad S(t) = \prod_{s \leq t} (1 - h_s - p_s)$$

with h_t the marginal monthly default hazard, p_t an optional prepayment hazard and r the annual effective interest rate (`config$ecl_discount = "none"` switches the discounting off). The 12-month figure uses $H = \text{config\$ecl_horizon_months}$, the lifetime figure the full term T . Stage 1 exposures carry the 12-month loss, stages 2 and 3 the lifetime loss; stage 3 exposures are credit-impaired and carry $LGD_1 * EAD_1$. When stage is NULL the rule is: stage 3 if $d_{pd} \geq \text{config\$ecl_stage_d}_{pd}[2]$, stage 2 if $d_{pd} \geq \text{config\$ecl_stage_d}_{pd}[1]$ or the 12-month PD now over the one at origination (pd_{orig}) is at least $\text{config\$ecl_s}_{icr_ratio}$, else stage 1.

Scenarios are a named list of shocks applied to the base inputs, each a list with any of pd_mult (non-negative multiplier of the hazards, capped at one), z (systematic factor of the one-factor model applied to the hazards with correlation ρ , negative in a bad year), lgd_add (added to the LGD, the result floored at zero) and ead_mult (non-negative); weights (normalised to one) give the probability-weighted result. When a shocked hazard plus the prepayment hazard exceeds one, the exit probability of that month is capped at one. The z shock is applied to each monthly hazard, not to the annual PD; because the Vasicek map is non-linear, the implied 12-month stressed PD is higher than the one obtained by stressing the annual PD with the same z and ρ (convert the annual PD yourself when that is wanted).

Value

An object of class `scr_ecl`: a list with exposures (only with `keep_rows = TRUE`: `id`, `segment`, `stage`, `ead`, `pd_12m`, `pd_life`, `ecl_12m`, `ecl_life`, `ecl`), `stages` (`stage`, `n`, `ead`, `ecl_12m`, `ecl_life`, `ecl`, `coverage`), `segments` (when a segment is given), `scenarios` (`scenario`, `weight`, `ecl_12m`, `ecl_life`, `ecl`), `totals` (`n`, `ead`, `ecl_12m`, `ecl_life`, `ecl`, `coverage`, `share_stage2`, `share_stage3`), `horizon`, `t_max`, `discount`, `stage_rule`, `ledger` and `config`.

References

International Accounting Standards Board (2014). *IFRS 9 Financial Instruments*, section 5.5 and paragraphs B5.5.1-B5.5.55.

See Also

Other irb-capital: [scr_capital\(\)](#), [scr_el\(\)](#), [scr_irb_rw\(\)](#), [scr_pd_stress\(\)](#), [scr_sa_rw\(\)](#)

Examples

```
cfg <- scr_config(verbose = FALSE)
d <- scr_demo_portfolio
h <- 1 - (1 - d$pd)^(1 / 12)      # flat monthly hazard from the annual PD
```

```
e <- scr_ecl(h, d$lgd, d$ead, eir = d$eir, dpd = d$dpd, pd_orig = d$pd_orig,
            t_max = 36L, segment = d$segment, config = cfg)
e
e$stages
```

scr_el

Expected loss per exposure

Description

The primitive every other function of the module uses: $pd * lgd * ead$ for performing exposures and $elbe * ead$ for defaulted ones, *elbe* being the best estimate of expected loss. A defaulted exposure without *elbe* uses *lgd* (PD equal to one). Arguments are recycled to a common length.

Usage

```
scr_el(pd, lgd, ead, defaulted = NULL, elbe = NULL)
```

Arguments

<i>pd</i> , <i>lgd</i> , <i>ead</i>	Numeric vectors: probability of default, loss given default (decimals) and exposure at default (currency).
<i>defaulted</i>	Optional 0/1 or logical vector.
<i>elbe</i>	Optional vector with the best estimate of expected loss of the defaulted rows (decimal of <i>ead</i>); ignored on performing rows.

Value

A numeric vector with the expected loss in currency.

See Also

Other irb-capital: [scr_capital\(\)](#), [scr_ecl\(\)](#), [scr_irb_rw\(\)](#), [scr_pd_stress\(\)](#), [scr_sa_rw\(\)](#)

Examples

```
scr_el(c(0.01, 0.02), 0.45, c(1000, 2000))
scr_el(0.02, 0.45, 1000, defaulted = TRUE, elbe = 0.6)
```

scr_elbe

*ELBE and in-default LGD on a grid of months since default***Description**

For every pool and every reference age τ of the grid, the expected loss best estimate is the mean realised LGD of the training defaults of the pool that were still in workout at τ (so that at $\tau = 0$ it equals the pool's long-run average), and the in-default LGD adds the unexpected-loss increment

$$\Delta^{UL}(\tau) = \max(0, \text{LGD}^{DT} - \text{LRA}) \frac{\rho(T_{\max}) - \rho(\tau - 1)}{\rho(T_{\max})}$$

read from the recovery profile of the pool's product mix, where $\rho(\tau - 1)$ is the cumulative discounted recovery rate of the months before age τ (zero at $\tau = 0$): the downturn uplift shrinks as the recoveries come in. The consistency table checks that `lgd_in_default` at $\tau = 0$ reproduces the pool's `lgd_dt`.

Usage

```
scr_elbe(x, grid = NULL)
```

Arguments

`x` An `scr_lgd()` object.
`grid` Months since default; NULL uses `lgd_elbe_grid`.

Value

An object of class `scr_elbe`: table (months_since_default, pool, n_open, share_open, recovered_share, elbe, delta_ul, lgd_in_default), consistency (per pool at $\tau = 0$), grid, t_max.

See Also

Other irb-lgd: `scr_lgd()`, `scr_lgd_downturn()`, `scr_lgd_floor()`, `scr_lgd_pools()`, `scr_lgd_validate()`, `scr_workout()`

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, n_boot = 20)
wo <- scr_workout(scr_demo_lgd, scr_demo_lgd_cashflows, rates = scr_demo_rates, config = cfg)
m <- scr_lgd(wo, drivers = c("product", "ltv", "prior_dpd_max"), config = cfg)
e <- scr_elbe(m)
e
```

scr_export

*Write the deliverables***Description**

For an `scr_result`: the selection workbook (`selection_<target>.xlsx`: funnel, gains, screening, hold-out, models, votes, consensus, ledger, redundancy), the WOE SQL and the executive summary in Markdown. For an `scr_scorecard`: three workbooks, as separate files, plus the score SQL:

Usage

```
scr_export(x, dir, stamp = TRUE, ...)

## S3 method for class 'scr_capital'
scr_export(x, dir, stamp = TRUE, ...)

## S3 method for class 'scr_classing'
scr_export(x, dir, stamp = TRUE, ...)

## S3 method for class 'scr_ead'
scr_export(x, dir, stamp = TRUE, validation = NULL, tag = "ccf", ...)

## S3 method for class 'scr_result'
scr_export(x, dir, stamp = TRUE, ...)

## S3 method for class 'scr_scorecard'
scr_export(x, dir, stamp = TRUE, ...)

## S3 method for class 'scr_lgd'
scr_export(
  x,
  dir,
  stamp = TRUE,
  validation = NULL,
  elbe = NULL,
  tag = "model",
  ...
)

## S3 method for class 'scr_pd'
scr_export(x, dir, stamp = TRUE, validation = NULL, ...)
```

Arguments

`x` An object from `scr_select()`, `scr_scorecard()`, `scr_coarse_classing()`, `scr_pd()`, `scr_lgd()`, `scr_ead()` or `scr_capital()`.

dir	Output directory. Created if it does not exist.
stamp	If TRUE (default), writes to a timestamped subdirectory, preserving earlier runs.
...	For scr_scorecard: precomputed cutoff, strategy, reject and monitor objects, and revenue_good/loss_bad for the default strategy table.
validation	For the IRB models (scr_pd, scr_lgd, scr_ead): the matching validation object (scr_pd_validate() , scr_lgd_validate() , scr_ead_validate()); NULL runs it on the hold-out where possible.
tag	For scr_lgd and scr_ead: the file tag (lgd_<tag>.xlsx, ead_<tag>.xlsx). scr_pd names its files after the target (pd_<target>.xlsx) and scr_capital after the framework (capital_<framework>.xlsx).
elbe	For scr_lgd: an scr_elbe() object; NULL computes it.

Details

scorecard_<target>.xlsx Score_Summary (with odds_orientation), Final_Scorecard, Coefficients, Sign_Check, Alignment, Alignment_Bands, Model_Card, Challenger and Swap_Set (when a challenger exists), Coarse_Classing and Decision_Ledger (after a lab commit).

validation_<target>.xlsx Score_Gains_Frozen, Variable_Gains_IV, Discrimination_CI, Stability_PSI_Timeline, Stability_CSI_Timeline, Stability_Variables, Calibration, Calibration_Bands, Performance_By_Vintage, Rank_Order_Diagnostics.

strategy_<target>.xlsx Population_Scope, Band_Coverage, Cutoff_Sweep, Strategy_Bands, Reject_Sensitivity, Monitoring_Plan.

For an scr_classing lab: one workbook (classing_<target>.xlsx) with the specification, the bins, the checks and the decision ledger. The IRB models write one workbook and one SQL file each (pd_<target>.xlsx, lgd_<tag>.xlsx, ead_<tag>.xlsx, capital_<framework>.xlsx), with the validation, the ledger and the model card as sheets.

The timeline and vintage sheets need the date column of the split; when it is absent they carry an availability row instead of a fabricated number.

Value

The object x, with \$files filled, invisibly.

See Also

Other production: [predict.scr_align\(\)](#), [scr_apply\(\)](#), [scr_monitor\(\)](#), [scr_monitoring_plan\(\)](#), [scr_reasons\(\)](#), [scr_sql\(\)](#)

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
  date_col = "ref_date")
out <- file.path(tempdir(), "scorecraft-example")
res <- scr_export(res, out, stamp = FALSE)
basename(unlist(res$files))
```

```
sc <- scr_export(scr_scorecard(res), out, stamp = FALSE)
basename(unlist(sc$files))
```

scr_fetch

Fetch a table with reproducible server-side sampling

Description

Sampling happens on the server, not in R: pulling a million rows to discard ninety per cent of them pays the network cost twice. `max_rows` is a memory guard: when it binds, the requested fraction is reduced on the server and the reduction is reported. The random expression follows the connection class (`rand(seed)` on Spark/Databricks/MySQL, `random()` on PostgreSQL/DuckDB, an integer-modulo expression on SQLite, which has no seedable `random()`); pass `sample_expr` to override it.

Usage

```
scr_fetch(
  con,
  table,
  sample_frac = 1,
  seed = NULL,
  max_rows = NULL,
  sample_expr = NULL,
  verbose = NULL
)
```

Arguments

<code>con</code>	A DBI connection, from scr_connect() .
<code>table</code>	Qualified table name.
<code>sample_frac</code>	Fraction of rows to fetch, in (0, 1].
<code>seed</code>	Seed of the server-side random function, where supported.
<code>max_rows</code>	Row cap. NULL switches it off.
<code>sample_expr</code>	Optional SQL expression yielding a uniform number in $[0, 1)$, used as <code>WHERE <sample_expr> <= sample_frac</code> .
<code>verbose</code>	TRUE/FALSE to echo (or not) the query for this call; NULL follows scr_verbose() .

Value

A data.table with the fetched table.

See Also

Other database: [scr_connect\(\)](#)

Examples

```
con <- scr_connect(driver = RSQLite::SQLite(), dbname = ":memory:")
d <- scr_demo; d$ref_date <- as.character(d$ref_date)
DBI::dbWriteTable(con, "dtm", d)
nrow(scr_fetch(con, "dtm", sample_frac = 0.5, seed = 42))
nrow(scr_fetch(con, "dtm", max_rows = 1000))
DBI::dbDisconnect(con)
```

scr_funnel

Audit funnel: every input variable and its fate

Description

The central deliverable. One row per input column, plus one per derived variable, with the descriptive profile, the verdict of every gate, the votes of every model and `exit_stage`, the exact stage at which the variable failed. No candidate disappears from the report.

Usage

```
scr_funnel(x, only_selected = FALSE, cols = "essentials")
```

Arguments

<code>x</code>	An object from <code>scr_select()</code> .
<code>only_selected</code>	If TRUE, returns only the approved ones.
<code>cols</code>	"essentials" (default) gives a lean view; "all" gives everything; or pass a vector of names.

Value

A data.table ordered with the approved first.

Values of `exit_stage`

- 00.config Never competed: it was in drop.
- 01.triage Constant, near-constant, high cardinality, exact duplicate, missing share above the ceiling, or no signal in the coarse IV.
- 02.binning The binning algorithm failed on this column.
- 03.screening Failed one of the eight admission rules.
- 04.holdout IV dropped out of sample, unstable PSI, or part of the hold-out falls in no bin.
- 05.correlation Redundant with a better-ranked variable.
- 05b.derived_excluded Passed everything, but is a column the pipeline created and `allow_derived_final = FALSE`.
- 06.consensus Not enough votes, or outside the top-N.

- 07. approved Entered the shortlist.
- 08. manual_drop In the automatic consensus, but removed by the analyst in the coarse classing lab ([scr_classing_choose\(\)](#)).

See Also

Other accessors: [scr_gains\(\)](#), [scr_leakage\(\)](#), [scr_result](#), [scr_score_gains\(\)](#), [scr_score_metrics\(\)](#), [scr_selected\(\)](#)

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
head(scr_funnel(res, only_selected = TRUE))
table(scr_funnel(res, cols = "all")$exit_stage)
```

scr_gains	<i>Gains table, at bin level</i>
-----------	----------------------------------

Description

One row per variable and bin, with counts, event rate, WOE, IV, lift, cumulative KS, precision and recall, plus the hold-out IV and the PSI of the variable.

Usage

```
scr_gains(x, only_selected = TRUE)
```

Arguments

- x An object from [scr_select\(\)](#).
- only_selected If TRUE (default), only the approved variables.

Value

A data. table at bin level.

See Also

Other accessors: [scr_funnel\(\)](#), [scr_leakage\(\)](#), [scr_result](#), [scr_score_gains\(\)](#), [scr_score_metrics\(\)](#), [scr_selected\(\)](#)

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
g <- scr_gains(res)
g[feature == scr_selected(res)[1], .(bin, count, pos_rate, woe, iv)]
```

scr_grades

Rating grades on the score

Description

Cuts the production score into grades whose PD is monotone. The grade boundaries are score cut points, direction-aware: grade 1 is the safest (the highest scores under higher_is_safer). Three constructions: "geometric" builds a [scr_master_scale\(\)](#) between percentiles 1 and 99 of the calibrated PD and converts its PD bounds into scores through the calibrated alignment; "quantile" cuts equal-count score bands (cut points moved half-way between neighbouring scores, so a boundary never sits on an observed value); "supplied" grades by the PD bands of a given master scale.

Usage

```
scr_grades(
  x,
  calibration = NULL,
  master_scale = NULL,
  n_grades = NULL,
  method = NULL,
  min_obligors = NULL,
  min_defaults = NULL,
  monotone = TRUE,
  pd_source = NULL,
  sample = "holdout",
  dr = NULL
)
```

Arguments

x	An scr_scorecard() .
calibration	An scr_calibrate() object (or its alignment); NULL uses the scorecard's own alignment.
master_scale	An scr_master_scale() for method = "supplied" (optional for "geometric").
n_grades, method, min_obligors, min_defaults, pd_source	NULL reads pd_n_grades, pd_grade_method, pd_min_obligors, pd_min_defaults and pd_source from the scorecard configuration.
monotone	Repair non-monotone grade PDs by pooling.

sample	Sample of the scorecard used to build the grades.
dr	Optional scr_dr with a grade column keyed by the final grades (see the section above).

Details

Grades below min_obligors obligors or min_defaults defaults are merged with the neighbour of closer default rate; the sequence of grade PDs is then repaired by pool-adjacent-violators when monotone = TRUE, and every merge is recorded in repairs. The grade PD (pd_be) is the long-run average of the grade default rates when a default-rate series by grade is given in dr (pd_source = "lra"), the sample default rate of the grade otherwise, or the mean of the calibrated individual PDs (pd_source = "mean_pd"). Concentration is reported as the Herfindahl index, the coefficient of variation of the grade shares and the Herfindahl-based hi index.

Value

An object of class scr_grades: table (grade, label, score_lo, score_hi, pd_lo, pd_hi, n, share, defaults, dr, pd_mean, pd_be, merged_from, and n_series, t_series when a series is given), breaks (ascending score cut points), band_grade (grade of every score band, ascending), direction, method, pd_source, master_scale, alignment (calibrated), alignment_score (the scorecard's), concentration (hhi, cv, hi, k), repairs, ledger, moc (empty, filled by [scr_moc\(\)](#)), dr (the pooled series), rows (score, outcome and grade of the sample), scorecard, sample, ct, sample_rate. Also calibration (the scr_pd_calibration when one was given), n_grades_requested, min_obligors, min_defaults, target and config.

Two-pass workflow with a default-rate series

The series must be keyed by the final grades of this same call. Run [scr_grades\(\)](#) once, grade the cohort panel with [predict.scr_grades\(\)](#), build the series with [scr_default_rate\(\)](#) (grade =) and pass it as dr in a second call with identical arguments (or in [scr_moc\(\)](#) and [scr_pd_validate\(\)](#), which read it the same way).

See Also

Other irb-pd: [predict.scr_grades\(\)](#), [predict.scr_pd\(\)](#), [scr_calibrate\(\)](#), [scr_master_scale\(\)](#), [scr_migration\(\)](#), [scr_moc\(\)](#), [scr_pd\(\)](#), [scr_pd_pit_ttc\(\)](#), [scr_pd_validate\(\)](#)

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  use_lightgbm = FALSE, xgb_rounds = 40, n_boot = 10)
res <- scr_select(scr_demo, "default", config = cfg, drop = c("id", "churn"),
                  date_col = "ref_date")
sc <- scr_scorecard(res)
cal <- scr_calibrate(sc, target = 0.06)
gr <- scr_grades(sc, cal, n_grades = 7, min_defaults = 10)
gr
gr$table[, c("grade", "score_lo", "score_hi", "n", "dr", "pd_be")]
# grade a cohort panel with the score cut points (the panel score is a
# different scale; the demo only shows the mechanics)
head(predict(gr, score = scr_demo_panel$score))
```

scr_irb_params

*IRB parameter tables by framework preset***Description**

Returns the numeric tables the internal ratings-based (IRB) functions read: probability of default (PD) floors, loss given default (LGD) input floors for own estimates, supervisory LGD values of the foundation approach, standardised credit conversion factors (CCF), asset-correlation parameters of the risk-weight function, maturity rules, the output floor and the standardised risk weights used for the floor comparison. Three presets ship: "bcb" (Brazil, BCB Resolutions 303/2023 and 229/2022), "basel3_final" (the consolidated Basel Framework in force from 2023) and "crr3" (the EU text applicable from 2025). The presets differ in a handful of cells, all visible with `print()`; users who need another jurisdiction edit the tables and pass the object to the functions that take params.

Usage

```
scr_irb_params(framework = c("bcb", "basel3_final", "crr3"))
```

Arguments

framework "bcb", "basel3_final" or "crr3".

Value

An object of class `scr_irb_params`: a list with `framework`, `source` (one line), `pd_floor`, `lgd_floor`, `lgd_firb`, `ccf_sa`, `ccf_floor_fraction`, `correlation`, `scaling_factor`, `confidence`, `m_default`, `m_range`, `output_floor`, `sa_rw` and `modified` (logical, set by the functions that receive the object when its tables were edited).

Regulatory texts

The presets encode a reading of the texts below at the time of the release. Regulation changes and is interpreted by each supervisor: before any regulatory use, check every table against the texts in force for the jurisdiction and the portfolio, and edit the tables where they differ. The package implements the calculations; it does not give regulatory advice.

References

Basel Committee on Banking Supervision. *The Basel Framework*, chapters CRE31 to CRE36 (IRB approach) and RBC20 (output floor).

Regulation (EU) 2024/1623 (CRR3), amending Regulation (EU) No 575/2013.

European Banking Authority (2017). *Guidelines on PD estimation, LGD estimation and the treatment of defaulted exposures*, EBA/GL/2017/16.

European Banking Authority (2019). *Guidelines for the estimation of LGD appropriate for an economic downturn*, EBA/GL/2019/03.

Banco Central do Brasil. BCB Resolution 229/2022 and BCB Resolution 303/2023.

International Accounting Standards Board (2014). *IFRS 9 Financial Instruments*.

See Also

Other irb-parameters: [scr_default\(\)](#), [scr_default_rate\(\)](#)

Examples

```
p <- scr_irb_params("bcb")
p
p$pd_floor
p2 <- p; p2$pd_floor$floor[p2$pd_floor$asset_class == "retail_other"] <- 0.001
```

scr_irb_rw

*IRB risk weight of one or many exposures***Description**

The asymptotic single risk factor function, vectorised over exposures: PD floors by asset class, LGD input floors for own estimates (approach = "airb"; the unsecured column of `params$lgd_floor` unless collateral names another column, blended with `secured_share`), the asset correlation of the class (with the firm-size adjustment of `corporate_sme` from sales and the multiplier for large or unregulated financial institutions when `fi` is TRUE), the maturity adjustment for wholesale classes only (`m` clipped to `params$m_range`, `params$m_default` when missing or under the foundation approach) and

Usage

```
scr_irb_rw(
  pd,
  lgd,
  ead = 1,
  m = NULL,
  asset_class,
  sales = NULL,
  fi = FALSE,
  defaulted = NULL,
  elbe = NULL,
  params = scr_irb_params("bcb"),
  approach = c("airb", "firb"),
  apply_floors = TRUE,
  collateral = NULL,
  secured_share = NULL,
  claim = NULL
)
```

Arguments

`pd`, `lgd`, `ead` Numeric vectors: probability of default, loss given default (decimals) and exposure at default (currency).

m	Effective maturity in years, non-negative (wholesale classes only, ignored and reported as NA on retail rows; NULL or NA uses params\$m_default).
asset_class	One of "corporate", "corporate_sme", "bank", "sovereign", "hvcre", "retail_mortgage", "qrre_revolver", "qrre_transactor", "retail_other"; a scalar or a vector.
sales	Annual sales of corporate_sme obligors, in the unit of params\$correlation\$sme, clipped to its bounds. A missing value takes the upper bound, i.e. no firm-size adjustment: the adjustment requires reported sales (Basel Framework CRE31.9; CRR Article 153(4)).
fi	Logical: regulated financial institution above the size threshold, or unregulated one (correlation multiplier).
defaulted	Optional 0/1 or logical vector.
elbe	Optional vector with the best estimate of expected loss of the defaulted rows (decimal of ead); ignored on performing rows.
params	An <code>scr_irb_params()</code> object.
approach	"airb" (own LGD, floored) or "firb" (supervisory LGD supplied by the caller, no LGD floor, maturity fixed).
apply_floors	TRUE (all input floors), FALSE (none) or a subset of c("pd", "lgd", "m").
collateral	Optional column of params\$lgd_floor naming the collateral type of each exposure ("financial", "receivables", "real_estate", "other_physical"); NULL means unsecured.
secured_share	Optional secured share in [0, 1] blending the unsecured and the collateral floors.
claim	Under "firb", an optional claim type per exposure naming a row of params\$lgd_firb (for example "senior_unsecured" or "subordinated"); the supervisory LGD of that row replaces lgd. NULL keeps the caller's lgd. The row names differ by preset ("senior_unsecured" under "bcb", "senior_unsecured_corporate" and "senior_unsecured_fi" otherwise); see params\$lgd_firb.

Details

$$K = \left[LGD \cdot N \left(\frac{G(PD) + \sqrt{R} G(0.999)}{\sqrt{1-R}} \right) - PD \cdot LGD \right] \cdot MA \cdot s$$

with $s = \text{params}\$scaling_factor$ and, for wholesale classes, $MA = (1 + (M - 2.5)b) / (1 - 1.5b)$, $b = (0.11852 - 0.05478 \ln PD)^2$ ($MA = 1$ for retail). Below $PD = 1e-5$, reachable only without a PD floor (sovereigns), b is held at its value at $1e-5$: the regulatory b makes $1 - 1.5b$ vanish near $PD = 2.9e-6$, where the adjustment would explode and change sign. Defaulted rows carry $K = \max(0, LGD - ELBE)$ under "airb" and zero under "firb"; a missing elbe is taken equal to lgd. $RW = 12.5 K$ and $RWA = RW * ead$.

Value

A data.table with one row per exposure: pd_used, lgd_used (after floors; PD one on defaulted rows), m (after clipping; params\$m_default under "firb"; NA on retail rows), r, b, ma, k, rw, rwa; attribute floors_hit counts the rows where each floor was binding.

References

Basel Committee on Banking Supervision (2023). *The Basel Framework*, CRE31 (IRB approach: risk-weight functions) and CRE32 (risk components). BCBS (2005). *An explanatory note on the Basel II IRB risk weight functions*.

See Also

Other irb-capital: [scr_capital\(\)](#), [scr_ecl\(\)](#), [scr_el\(\)](#), [scr_pd_stress\(\)](#), [scr_sa_rw\(\)](#)

Examples

```
scr_irb_rw(0.01, 0.45, m = 2.5, asset_class = "corporate")
scr_irb_rw(c(0.01, 0.02), c(0.20, 0.80), asset_class = c("retail_mortgage", "qrre_revolver"))
r <- scr_irb_rw(1e-4, 0.5, asset_class = "retail_other")
attr(r, "floors_hit")
# foundation approach: the supervisory LGD of the claim type
scr_irb_rw(0.01, lgd = 0, m = 2.5, asset_class = "corporate", approach = "firb",
          claim = "senior_unsecured")
```

scr_iv	<i>Information Value of any grouping</i>
--------	--

Description

Laplace smoothing by default. It is not cosmetic: without it, a single-class group (a normal situation in a small sentinel population) yields Inf and contaminates any ordering that depends on the IV.

Usage

```
scr_iv(g, y, laplace = 0.5)
```

Arguments

- g Group vector (any coercible type). Rows where g or y is NA are ignored, whatever the type of g.
- y 0/1 outcome vector.
- laplace Smoothing constant added to each count. 0 switches it off.

Details

Implemented with [tabulate\(\)](#) on integer codes rather than data.table aggregation: this function is called once per candidate variable, hundreds of times per run, and the fixed cost dominated the triage.

Value

Total IV, a scalar. Zero when fewer than two groups are populated.

See Also

Other metrics: [scr_metrics\(\)](#), [scr_psi\(\)](#)

Examples

```
set.seed(1)
y <- stats::rbinom(1000, 1, 0.3)
g <- ifelse(stats::runif(1000) < 0.5 + 0.3 * y, "A", "B")
scr_iv(g, y)
```

scr_leakage

Leakage and suspicious-strength audit

Description

Separates what the pipeline failed for excessive strength (IV_SUSPICIOUS, DEGENERATE_BIN) from what it admitted but deserves a second look (IV above config\$iv_suspect). A bin with no events or no non-events is the symptom with no innocent explanation: the variable determines the outcome on part of the population.

Usage

```
scr_leakage(x, threshold = NULL)
```

Arguments

x An object from [scr_select\(\)](#).
threshold Warning threshold. NULL uses config\$iv_suspect.

Value

An scr_leakage object (a list with barred, degenerate, approved_suspect), with a print method.

See Also

Other accessors: [scr_funnel\(\)](#), [scr_gains\(\)](#), [scr_result](#), [scr_score_gains\(\)](#), [scr_score_metrics\(\)](#), [scr_selected\(\)](#)

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
scr_leakage(res)
```

scr_lgd

Two-stage LGD model and pools on the reference data set

Description

Fits the standard two-stage structure on the RDS of [scr_workout\(\)](#):

$$\text{LGD} = P(\text{cure} \mid x) \text{LGD}^{\text{cure}} + (1 - P(\text{cure} \mid x)) E[\text{LGD} \mid \text{no cure}, x]$$

The **cure stage** is a binary model on `is_cure` with the scorecard machinery: optimal binning of the drivers on the training cohorts, WOE, hold-out revalidation with frozen bins and a logistic regression on the WOE columns with the sign check (every coefficient positive). The **severity stage** bins the same drivers against the realised LGD of the non-cures with [scr_bin_continuous\(\)](#) (bin means, monotone, at least `lgd_min_defaults_bin` defaults per bin, hold-out revalidated) and fits a fractional logit (glm with a quasi-binomial family on the bin means) or, with `lgd_severity = "beta"`, a beta regression through the `betareg` package. LGD^{cure} is the mean realised LGD of the cures on train (costs and the discount effect, never zero by decree).

Usage

```
scr_lgd(
  x,
  drivers,
  config = scr_config(),
  holdout = 0.3,
  date_col = "default_date"
)
```

Arguments

<code>x</code>	An scr_workout() object.
<code>drivers</code>	Column names of the RDS to use as drivers.
<code>config</code>	A scr_config() ; keys <code>lgd_*</code> , the binning and hold-out keys of stage 2, <code>max_abs_coef</code> , <code>n_boot</code> , <code>ci_level</code> , <code>seed</code> , <code>nthread</code> .
<code>holdout</code>	Share of the cohorts held out (by default date).
<code>date_col</code>	Column of the RDS with the default date.

Details

The split is by cohort of default: the last holdout share of the default dates is the hold-out. Metrics on both samples: RMSE, MAE, R-squared, Spearman rho, Somers' D of the prediction with respect to the realised LGD (generalised AUC $(D + 1) / 2$) with a bootstrap confidence interval, and the loss capture ratio. Pools come from [scr_lgd_pools\(\)](#). The object carries a provisional downturn (type 3 add-on, or none, by configuration) and no floor until [scr_lgd_downturn\(\)](#) and [scr_lgd_floor\(\)](#) run.

Value

An object of class `scr_lgd`: `split`, `drivers`, `cure` (`fit`, `features`, `coef`, `sign_check`, `bins`, `holdout`), `severity` (`fit`, `features`, `coef`, `engine`, `sign_check`, `bins`), `lgd_cure`, `has_cures`, `scored` (one row per default: `sample`, `p_cure`, `severity`, `lgd_pred`, `pool`, `lgd_real`), `bins_idx`, `samples` (predicted vs realised by decile of the prediction), `metrics`, `pools`, `downturn`, `floors`, `workout` (the profile and summary of the RDS), `model_card`, `ledger`, `config`.

See Also

Other irb-`lgd`: `scr_elbe()`, `scr_lgd_downturn()`, `scr_lgd_floor()`, `scr_lgd_pools()`, `scr_lgd_validate()`, `scr_workout()`

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, nboot = 20)
wo <- scr_workout(scr_demo_lgd, scr_demo_lgd_cashflows, rates = scr_demo_rates, config = cfg)
m <- scr_lgd(wo, drivers = c("product", "ltv", "prior_dpd_max", "months_on_book", "region"),
             config = cfg)

m
m$pools[, c("pool", "n", "lra", "lra_ew", "moc_c", "lgd_dt")]
m$metrics
```

<code>scr_lgd_downturn</code>	<i>Downturn LGD per pool</i>
-------------------------------	------------------------------

Description

Quantifies the downturn per pool from user-supplied downturn periods. `method = "type1"` (observed impact): the default-weighted realised LGD of the training defaults whose default date falls inside the periods; a pool with fewer than ten such defaults falls back to type 3. `method = "type3"`: the long-run average plus `add_on`. `method = "none"`: the long-run average. The reference value (a challenger, not a bound) is the mean of the two worst calendar years of the pool. Both use the training rows only, so the hold-out stays independent evidence. The downturn LGD used for capital is

$$\text{LGD}^{DT} = \min(1, \max(\text{LRA} + \text{MoC}, \text{DT} + \text{MoC}))$$

and the impact $\text{LGD}^{DT} - \min(1, \text{LRA} + \text{MoC})$ is reported per pool.

Usage

```
scr_lgd_downturn(
  x,
  periods = NULL,
  method = NULL,
  add_on = NULL,
  reason = NULL
)
```

Arguments

x	An <code>scr_lgd()</code> object.
periods	A table with start and end dates of the downturn periods. Required for "type1".
method	"type1", "type3" or "none"; NULL uses <code>lgd_downturn</code> .
add_on	Type-3 add-on; NULL uses <code>lgd_downturn_add_on</code> .
reason	Free text recorded in the ledger, mandatory: the choice of periods and method is an analyst decision.

Value

The `scr_lgd` object with downturn (table per pool: `lra`, `moc_c`, `dt_observed`, `n_downturn`, `dt_type3`, `reference_value`, `method_used`, `dt`, `lgd_dt`, `impact`, `below_reference`; periods, method, add_on, status, reason) and the pool columns `lgd_dt` and `lgd_final` updated.

See Also

Other irb-`lgd`: [scr_elbe\(\)](#), [scr_lgd\(\)](#), [scr_lgd_floor\(\)](#), [scr_lgd_pools\(\)](#), [scr_lgd_validate\(\)](#), [scr_workout\(\)](#)

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, n_boot = 20)
wo <- scr_workout(scr_demo_lgd, scr_demo_lgd_cashflows, rates = scr_demo_rates, config = cfg)
m <- scr_lgd(wo, drivers = c("product", "ltv", "prior_dpd_max"), config = cfg)
m <- scr_lgd_downturn(m, periods = data.frame(start = as.Date("2022-01-01"),
                                             end = as.Date("2023-12-31")),
                  reason = "reference rate above 13% in 2022-2023")
m$downturn$table
```

<code>scr_lgd_floor</code>	<i>Input floor on the downturn LGD per pool</i>
----------------------------	---

Description

Applies the LGD input floor of the framework's parameter table, blended between the unsecured and the collateralised floor with the secured share of the exposure:

$$\text{floor} = \text{floor}_U (1 - s) + \text{floor}_S s, \quad \text{LGD}^{\text{final}} = \max(\text{LGD}^{DT}, \text{floor})$$

A missing unsecured floor (residential mortgages, whose floor applies to the whole exposure) uses the collateral floor throughout; an asset class with no floor at all yields a floor of zero.

Usage

```
scr_lgd_floor(
  x,
  params = NULL,
  asset_class = NULL,
  secured_share = NULL,
  collateral = c("real_estate", "financial", "receivables", "other_physical")
)
```

Arguments

<code>x</code>	An scr_lgd() object.
<code>params</code>	An scr_irb_params() object; NULL uses the configured framework. Edits are detected and recorded in the ledger.
<code>asset_class</code>	Row of <code>params\$lgd_floor</code> ; NULL uses <code>asset_class</code> of the configuration.
<code>secured_share</code>	Secured share of the exposure in $[0, 1]$: one value or one per pool. NULL means unsecured.
<code>collateral</code>	Column of <code>params\$lgd_floor</code> for the secured part: "real_estate", "financial", "receivables" or "other_physical".

Value

The `scr_lgd` object with floors (table per pool with `lgd_dt`, `floor_unsecured`, `floor_secured`, `secured_share`, `floor`, `lgd_final`, `binding`; `asset_class`, `collateral`, `framework`, `params_modified`, `binding_share`) and the pool columns `floor` and `lgd_final` updated.

See Also

Other `irb`-lgd: [scr_elbe\(\)](#), [scr_lgd\(\)](#), [scr_lgd_downturn\(\)](#), [scr_lgd_pools\(\)](#), [scr_lgd_validate\(\)](#), [scr_workout\(\)](#)

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, nboot = 20)
wo <- scr_workout(scr_demo_lgd, scr_demo_lgd_cashflows, rates = scr_demo_rates, config = cfg)
m <- scr_lgd(wo, drivers = c("product", "ltv", "prior_dpd_max"), config = cfg)
m <- scr_lgd_floor(m, asset_class = "retail_other", secured_share = 0.4)
m$floors$table
```

Description

Cuts the training predictions into `n_pools` quantile bands, merges the bands with fewer than `min_defaults` defaults into the neighbour with the closer long-run average, then merges adjacent bands whose long-run averages break the increasing order (pool-adjacent violators), so that the pools are ordered both in predicted and in realised LGD. Per pool: the default-weighted long-run average (the regulatory estimate), the exposure-weighted one, the standard error, the category-C margin of conservatism (one-sided 95% t interval on the mean) and their sum.

Usage

```
scr_lgd_pools(x, n_pools = NULL, min_defaults = NULL)
```

Arguments

`x` An `scr_lgd()` object.
`n_pools` Target number of pools; NULL uses `lgd_n_pools`.
`min_defaults` Minimum defaults per pool; NULL uses `lgd_min_defaults_bin`.

Value

A data.table with one row per pool: `pool`, `pred_lo`, `pred_hi`, `pred_mean`, `n`, `share`, `ead`, `lra`, `lra_ew`, `sd`, `se`, `moc_c`, `lra_moc`, `merged_from`.

See Also

Other irb-`lgd`: `scr_elbe()`, `scr_lgd()`, `scr_lgd_downturn()`, `scr_lgd_floor()`, `scr_lgd_validate()`, `scr_workout()`

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, n_boot = 20)
wo <- scr_workout(scr_demo_lgd, scr_demo_lgd_cashflows, rates = scr_demo_rates, config = cfg)
m <- scr_lgd(wo, drivers = c("product", "ltv", "prior_dpd_max"), config = cfg)
scr_lgd_pools(m, n_pools = 4)
```

scr_lgd_validate

Validation battery of an LGD model

Description

Runs the three blocks of the usual LGD validation on the hold-out sample (or on newdata) against the training reference:

Usage

```
scr_lgd_validate(x, newdata = NULL)
```

Arguments

<code>x</code>	An <code>scr_lgd()</code> object.
<code>newdata</code>	NULL (the hold-out), an <code>scr_workout()</code> object or a table with the drivers, <code>lgd_real</code> , <code>ead</code> and the default date.

Details

- **Calibration.** Per pool and for the portfolio, the one-sided t-test of realised against estimated LGD (the pool long-run average), where under-estimation is the failure: $p = 1 - \Phi(t)$; the loss shortfall $1 - \sum(\text{LGD_real} \leq E) / \sum(\text{LGD_pred} \leq E)$; the coverage of the realised mean by the downturn LGD; the regression of realised on predicted.
- **Discrimination.** Somers' D / generalised AUC of the prediction with its bootstrap interval, compared with the training value through $S = (\text{gAUC_init} - \text{gAUC_curr}) / \text{sigma_curr}$; Spearman rho; the loss capture ratio; R-squared.
- **Stability.** PSI of the pool distribution and of the bins of every driver of both stages, with the fixed and the n-adjusted threshold.
- **Pools.** Homogeneity within a pool (Welch test between the halves of the pool split at its median prediction; a small p-value means a pool that still discriminates) and heterogeneity between adjacent pools (Welch test; a large p-value means pools that do not differ).

Traffic lights use the p-value thresholds of `config$pd_lights` (shared with the PD validation) (red below the first, amber below the second) and the fixed PSI thresholds.

Value

An object of class `scr_lgd_validation`: calibration (per pool), portfolio, discrimination, stability (pools, drivers), homogeneity, heterogeneity, summary (test, statistic, p, light), sample, n.

See Also

Other irb-lgd: `scr_elbe()`, `scr_lgd()`, `scr_lgd_downturn()`, `scr_lgd_floor()`, `scr_lgd_pools()`, `scr_workout()`

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, n_boot = 20)
wo <- scr_workout(scr_demo_lgd, scr_demo_lgd_cashflows, rates = scr_demo_rates, config = cfg)
m <- scr_lgd(wo, drivers = c("product", "ltv", "prior_dpd_max"), config = cfg)
v <- scr_lgd_validate(m)
v
v$calibration
```

scr_master_scale	<i>Master scale of PD grades</i>
------------------	----------------------------------

Description

A grade structure with geometric midpoints and geometric-mean boundaries:

$$PD_k = PD_1 r^{k-1}, \quad r = (PD_K / PD_1)^{1/(K-1)}, \quad \text{bound}_k = \sqrt{PD_k PD_{k+1}},$$

so that every grade doubles (or multiplies by r) the PD of the one before. With method = "supplied" the table comes from the user: a numeric vector of midpoints (boundaries derived as the geometric means) or a data.frame with pd_lo and pd_hi (and optionally pd_mid, label). Grade 1 is always the safest.

Usage

```
scr_master_scale(
  pd_min = 3e-04,
  pd_max = 0.3,
  n_grades = 10L,
  method = c("geometric", "supplied"),
  grades = NULL,
  labels = NULL
)
```

Arguments

pd_min, pd_max	PD midpoints of the first and the last grade.
n_grades	Number of grades.
method	"geometric" (default) or "supplied".
grades	For "supplied": a numeric vector of midpoints or a data.frame with pd_lo and pd_hi.
labels	Optional grade labels (default "1", "2", ...).

Value

A data.table of class scr_master_scale with grade, label, pd_lo, pd_mid, pd_hi, and the attributes ratio (the geometric ratio between consecutive midpoints) and method.

See Also

Other irb-pd: [predict.scr_grades\(\)](#), [predict.scr_pd\(\)](#), [scr_calibrate\(\)](#), [scr_grades\(\)](#), [scr_migration\(\)](#), [scr_moc\(\)](#), [scr_pd\(\)](#), [scr_pd_pit_ttc\(\)](#), [scr_pd_validate\(\)](#)

Examples

```
ms <- scr_master_scale(0.0005, 0.25, n_grades = 8)
ms
scr_master_scale(method = "supplied", grades = c(0.001, 0.01, 0.05, 0.20))
```

scr_metrics

AUC, KS and Gini of a score, with a bootstrap confidence interval

Description

AUC through the Mann-Whitney U statistic with tie correction, computed on the table of counts per unique score: a WOE score is constant within the bin, so ties are the rule. Everything in double on purpose: with integer counts, $n1 * n0$ overflows 2^{31} from about 46 thousand observations per class and returns NA.

Usage

```
scr_metrics(
  score,
  y,
  higher_is_event = TRUE,
  ci = TRUE,
  n_boot = 200L,
  level = 0.95,
  seed = NULL,
  nthread = 1L
)
```

Arguments

score	Numeric vector with the score.
y	0/1 outcome vector (numeric or logical), same length as score. NA rows are dropped; any other value is an error.
higher_is_event	If TRUE (default), a higher score means a higher probability of the event (logit, probability, propensity score). Pass FALSE for a credit points score (higher_is_safer), and the AUC is reported above 0.5 when the score ranks correctly.
ci	Compute the confidence interval. FALSE returns point estimates only.
n_boot	Number of bootstrap resamples.
level	Confidence level.
seed	Bootstrap seed, local to the call (the user's random stream is restored on exit); NULL draws from the user's stream.
nthread	Parallel workers for the resamples.

Details

The confidence interval is **always** computed by default: a bootstrap stratified by outcome, percentile method, with `n_boot` resamples. Gini is derived from AUC ($2 * AUC - 1$) inside each resample, never bootstrapped separately. The cost is absorbed by `nthread` (parallelism by resample).

DeLong's analytic variance is not used: the interval is a stratified percentile bootstrap, which also covers KS.

The AUC is computed from the counts per unique score after one sort, so its cost is $O(n \log n)$, never the $O(n_1 n_0)$ of the pairwise definition.

Value

A list of class `scr_metrics` with `auc`, `ks`, `gini`, the bounds `auc_lo/auc_hi`, `ks_lo/ks_hi`, `gini_lo/gini_hi` (NA when `ci = FALSE`), `n`, `events`, `n_boot` and `level`. Everything is `NA_real_` when only one class is present or no valid case exists.

References

DeLong, E. R., DeLong, D. M. and Clarke-Pearson, D. L. (1988). Comparing the areas under two or more correlated receiver operating characteristic curves. *Biometrics*, 44(3), 837-845.

See Also

Other metrics: [scr_iv\(\)](#), [scr_psi\(\)](#)

Examples

```
set.seed(1)
y <- rep(0:1, each = 500)
s <- stats::rnorm(1000) + 0.8 * y
m <- scr_metrics(s, y, n_boot = 50, seed = 1)
m
as.data.frame(m)
```

scr_migration	<i>Migration matrix between two rating dates</i>
---------------	--

Description

Counts N_{ij} of obligors in grade i at the first date and grade j at the second, the row probabilities p_{ij} , the upper and lower matrix weighted bandwidths

$$MWB_{up} = \frac{\sum_{i < j} |i - j| N_i p_{ij}}{\sum_i \max(|i - K|, |i - 1|) N_i \sum_{j > i} p_{ij}},$$

(and the mirror image for downgrades), the z statistic of every off-diagonal cell against its neighbour closer to the diagonal (a significantly positive value means the probability does not decay away from the diagonal) and the mobility summary. Values of `grade_t1` outside $1..K$ count as `default`, NA as `closed`; both stay out of the bandwidths.

Usage

```
scr_migration(grade_t0, grade_t1, K = NULL)
```

Arguments

grade_t0, grade_t1
Integer grades at the two dates, same length.

K
Number of grades; NULL uses the largest grade observed.

Value

An object of class `scr_migration`: matrix (counts, K rows, K + 2 columns), p (row probabilities), n (row totals), mwb_upper, mwb_lower, z (K x K), n_significant (cells with $z > 1.645$), mobility (share_stable, share_up, share_down, mean_distance, share_default, share_closed). Also K, the number of grades.

See Also

Other irb-pd: `predict.scr_grades()`, `predict.scr_pd()`, `scr_calibrate()`, `scr_grades()`, `scr_master_scale()`, `scr_moc()`, `scr_pd()`, `scr_pd_pit_ttc()`, `scr_pd_validate()`

Examples

```
set.seed(2)
g0 <- sample(1:5, 500, TRUE)
g1 <- pmin(5, pmax(1, g0 + sample(c(-1, 0, 0, 0, 1), 500, TRUE)))
g1[sample(500, 10)] <- NA
scr_migration(g0, g1, K = 5)
```

scr_moc

*Margin of conservatism, by category***Description**

Appends entries to the MoC ledger of an `scr_grades()` object. Category "C" (general estimation error) is quantified: "ci_timeseries" takes the upper bound of a one-sided level interval of the long-run average from the cohort series, $t_{q,T-1} sd(DR_t) / \sqrt{T}$ per grade; "ci_binomial" uses $z_q \sqrt{PD(1 - PD)/n}$ on the obligors (or obligor-years when a series exists); "bootstrap" resamples the outcomes of the sample within each grade (drawn as the resampled default rate, $\text{Binomial}(n, DR) / n$, its exact distribution) and takes the level quantile of the default rate above the estimate. Categories "A" (data and methodological deficiencies) and "B" (changes in standards or environment) are expert quantities: value (one number or one per grade, in PD units) and a non-empty reason are mandatory. The ledger is append-only: A and B entries accumulate, a new C supersedes the previous one (kept with `active = FALSE`).

Usage

```
scr_moc(
  x,
  category = c("A", "B", "C"),
  method = NULL,
```

```

    level = NULL,
    value = NULL,
    reason = NULL,
    dr = NULL,
    n_boot = 200L,
    seed = NULL
  )

```

Arguments

x	An scr_grades() object.
category	"A", "B" or "C".
method	For "C": "ci_timeseries", "ci_binomial" or "bootstrap"; NULL reads config\$pd_moc_method.
level	One-sided confidence level; NULL reads config\$pd_moc_level.
value	For "A"/"B": the add-on in PD units, length 1 or one per grade.
reason	Justification (mandatory for "A"/"B").
dr	Optional scr_dr by grade for "ci_timeseries", keyed by the final grades of x; NULL uses the series already stored in x\$dr by scr_grades() .
n_boot, seed	Bootstrap resamples and seed for "bootstrap".

Value

The scr_grades object with the entries appended to moc.

See Also

Other irb-pd: [predict.scr_grades\(\)](#), [predict.scr_pd\(\)](#), [scr_calibrate\(\)](#), [scr_grades\(\)](#), [scr_master_scale\(\)](#), [scr_migration\(\)](#), [scr_pd\(\)](#), [scr_pd_pit_ttc\(\)](#), [scr_pd_validate\(\)](#)

Examples

```

cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  use_lightgbm = FALSE, xgb_rounds = 40, n_boot = 10)
res <- scr_select(scr_demo, "default", config = cfg, drop = c("id", "churn"),
                  date_col = "ref_date")
sc <- scr_scorecard(res)
gr <- scr_grades(sc, n_grades = 6, min_defaults = 10)
gr <- scr_moc(gr, "C", method = "ci_binomial")
gr <- scr_moc(gr, "A", value = 0.002, reason = "missing unlikeliness-to-pay trigger before 2024")
gr$ moc

```

scr_model

*Stages 3 and 4: multi-strategy selection and consensus***Description**

Trains the classifiers enabled in the configuration on the WOE columns of the eligible pool, measures each on the hold-out (AUC/KS/Gini with a bootstrap CI) and combines the votes:

Usage

```
scr_model(bins, config = scr_config())
```

Arguments

bins An object from [scr_bin\(\)](#).
 config An object from [scr_config\(\)](#).

Details

consensus_score = mean of the importance rank percentiles, weighted by the hold-out Gini of each model
 votes = how many models elected the feature (top-K, or non-zero coefficient in the elastic net)

The final cut respects [target_min, target_max]. If the strict consensus does not reach target_min, relaxation happens in **named**, recorded steps (min_votes reduced; completed by score), never resurrecting a feature failed by an earlier gate.

Value

An scr_models object with votes (one row per model and feature), metrics (one row per model, with CI), consensus (table, selected, meta) and the originating bins.

See Also

Other stages: [scr_align\(\)](#), [scr_bin\(\)](#), [scr_cutoff\(\)](#), [scr_reject\(\)](#), [scr_scorecard\(\)](#), [scr_select\(\)](#), [scr_split\(\)](#), [scr_strategy\(\)](#), [scr_triage\(\)](#)

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
sp <- scr_split(scr_demo, "default", date_col = "ref_date", drop = "id")
md <- scr_model(scr_bin(scr_triage(sp, cfg), cfg), cfg)
md
md$consensus$selected
```

scr_monitor

*Monitor the scorecard on new data***Description**

Recomputes, per period of `date_col` (or for the whole data), the score PSI against train with frozen bands, the CSI of every variable with frozen bins plus the signed points shift and, when the target is present, the performance by vintage (event rate, AUC/KS/Gini with CI). Always reports both thresholds (fixed and n-adjusted). Schedules nothing: the analyst calls it when needed.

Usage

```
scr_monitor(
  x,
  newdata,
  date_col = NULL,
  target = NULL,
  alpha = NULL,
  n_boot = NULL,
  plan = NULL
)
```

Arguments

<code>x</code>	An object from scr_scorecard() .
<code>newdata</code>	New table with the source columns.
<code>date_col</code>	Period column. NULL treats newdata as a single period. Rows with a missing date form a period of their own (NA, last).
<code>target</code>	Target column in newdata, for the performance by vintage. NULL skips it.
<code>alpha</code>	Level of the adjusted threshold. NULL (default) takes it from the plan.
<code>n_boot</code>	CI resamples per vintage. NULL uses the configuration.
<code>plan</code>	The monitoring contract: NULL (default) uses the plan stored in the scorecard (scr_monitoring_plan()); otherwise an item/value table, or the path of a strategy workbook written by scr_export() , whose Monitoring_Plan sheet is read. The fixed thresholds of the PSI and CSI flags, the alpha of the adjusted threshold and <code>min_events_per_period</code> come from it.

Value

An `scr_monitor` object with `psi` (score, per period), `csi` (per variable and period), `vintage` (or NULL; status says "insufficient" when a period has fewer events than the plan requires) and `plan` (the contract actually used).

See Also

Other production: [predict.scr_align\(\)](#), [scr_apply\(\)](#), [scr_export\(\)](#), [scr_monitoring_plan\(\)](#), [scr_reasons\(\)](#), [scr_sql\(\)](#)

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
sc <- scr_scorecard(res)
mo <- scr_monitor(sc, scr_demo, date_col = "ref_date", target = "default")
mo
mo$psi
head(mo$csi)
```

scr_monitoring_plan	<i>Monitoring plan read by scr_monitor()</i>
---------------------	--

Description

A small item/value table with the thresholds and the frozen score bands of a scorecard. It is created by [scr_scorecard\(\)](#) from the configuration, written to the Monitoring_Plan sheet of the strategy workbook by [scr_export\(\)](#), and **read back** by [scr_monitor\(\)](#): change a threshold in the sheet, pass the file (or the edited table) as plan, and the flags follow the plan, not the configuration.

Usage

```
scr_monitoring_plan(x, breaks = NULL)
```

Arguments

x	An object from scr_scorecard() , or a configuration from scr_config() plus breaks.
breaks	Frozen score bands, when x is a configuration.

Value

A data.frame of class `scr_monitoring_plan` with the items `psi_score_fixed_moderate`, `psi_score_fixed_action`, `psi_adjusted_alpha`, `csi_variable_fixed_moderate`, `csi_variable_fixed_action`, `score_bands`, `min_events_per_period` and `threshold_source`.

See Also

Other production: [predict.scr_align\(\)](#), [scr_apply\(\)](#), [scr_export\(\)](#), [scr_monitor\(\)](#), [scr_reasons\(\)](#), [scr_sql\(\)](#)

Examples

```
plan <- scr_monitoring_plan(scr_config(), breaks = c(-Inf, 500, 550, 600, Inf))
plan
```

scr_pd

*The PD model: grades, margin of conservatism and the floor***Description**

Assembles the final grade table: `pd_be` from [scr_grades\(\)](#), the active entries of the MoC ledger by category (A and B summed over their entries, the latest C), `pd_moc` = `pd_be` + A + B + C, the PD floor of the asset class under the framework of `params`, and `pd_final` = `max(pd_moc, floor)`. With `philosophy` = "pit" the through-the-cycle `pd_moc` is converted with the one-factor bridge of [scr_pd_pit_ttc\(\)](#) before the floor (`rho` and `z` required).

Usage

```
scr_pd(
  grades,
  moc = NULL,
  params = NULL,
  asset_class = NULL,
  philosophy = c("ttc", "pit"),
  rho = NULL,
  z = NULL
)
```

Arguments

<code>grades</code>	An scr_grades() object, after scr_moc() .
<code>moc</code>	NULL uses <code>grades\$moc</code> ; otherwise a ledger in the same format.
<code>params</code>	An scr_irb_params() ; NULL uses <code>config\$framework</code> .
<code>asset_class</code>	Asset class of the floor; NULL uses <code>config\$asset_class</code> .
<code>philosophy</code>	"ttc" (default) or "pit".
<code>rho, z</code>	Asset correlation and systematic factor for "pit".

Value

An object of class `scr_pd`: table (`grade`, `label`, `score_lo`, `score_hi`, `n`, `share`, `defaults`, `dr`, `pd_be`, `moc_a`, `moc_b`, `moc_c`, `pd_moc`, `pd_ttc`, `pd_pit`, `floor`, `pd_final`, `floor_applied`), `breaks`, `band_grade`, `direction`, `alignment`, `alignment_score`, `master_scale`, `asset_class`, `framework`, `floor`, `philosophy`, `rho`, `z`, `moc_ledger`, `calibration`, `concentration`, `portfolio` (weighted `pd_be`, `pd_moc`, `pd_final`, `moc_bp`, `share_at_floor`), `scorecard`, `ledger`, `model_card`. Also `params_modified`, `repairs`, `dr`, `rows`, `pd_source`, `grade_method`, `sample`, `target`, `config` and, after [scr_export\(\)](#), files.

See Also

Other `irb-pd`: [predict.scr_grades\(\)](#), [predict.scr_pd\(\)](#), [scr_calibrate\(\)](#), [scr_grades\(\)](#), [scr_master_scale\(\)](#), [scr_migration\(\)](#), [scr_moc\(\)](#), [scr_pd_pit_ttc\(\)](#), [scr_pd_validate\(\)](#)

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                 use_lightgbm = FALSE, xgb_rounds = 40, n_boot = 10)
res <- scr_select(scr_demo, "default", config = cfg, drop = c("id", "churn"),
                 date_col = "ref_date")
sc <- scr_scorecard(res)
gr <- scr_moc(scr_grades(sc, n_grades = 6, min_defaults = 10), "C", method = "ci_binomial")
pd <- scr_pd(gr)
pd
pd$table[, c("grade", "pd_be", "moc_c", "pd_final", "floor_applied")]
head(predict(pd, score = c(480, 560, 640), type = "pd_final"))
head(scr_apply(pd, head(scr_demo, 5)))
cat(tail(scr_sql(pd), 8), sep = "\n")
```

scr_pd_pit_ttc

One-factor bridge between point-in-time and through-the-cycle PD

Description

Vasicek's conditional default probability:

$$PD_{PIT} = \Phi \left(\frac{\Phi^{-1}(PD_{TTC}) - \sqrt{\rho} z}{\sqrt{1 - \rho}} \right),$$

and its inverse for to = "ttc". A positive z is a benign state (lower PIT PD), a negative one a stressed state.

Usage

```
scr_pd_pit_ttc(pd, z, rho, to = c("pit", "ttc"))
```

Arguments

pd	Numeric PDs in (0, 1).
z	Systematic factor (standard normal scale).
rho	Asset correlation in (0, 1).
to	"pit" (input is TTC) or "ttc" (input is PIT).

Value

A numeric vector of the length of pd.

References

Vasicek, O. (2002). The distribution of loan portfolio value. *Risk*, 15(12), 160-162.

See Also

[scr_pd_stress\(\)](#), the same bridge with the systematic factor given as a quantile q rather than a value of z .

Other irb-pd: [predict.scr_grades\(\)](#), [predict.scr_pd\(\)](#), [scr_calibrate\(\)](#), [scr_grades\(\)](#), [scr_master_scale\(\)](#), [scr_migration\(\)](#), [scr_moc\(\)](#), [scr_pd\(\)](#), [scr_pd_validate\(\)](#)

Examples

```
scr_pd_pit_ttc(c(0.01, 0.05), z = -2, rho = 0.15)
scr_pd_pit_ttc(scr_pd_pit_ttc(0.02, z = -1, rho = 0.1), z = -1, rho = 0.1, to = "ttc")
```

scr_pd_stress	<i>Stressed PD of the one-factor model</i>
---------------	--

Description

The conditional PD at confidence q : $N((G(pd) + \sqrt{\rho}) G(q)) / \sqrt{1 - \rho})$. With $q = 0.999$ and the regulatory correlation it is the PD inside the risk-weight function, so that the capital requirement of a retail exposure is $lgd * (scr_pd_stress(pd, r, 0.999) - pd)$. Used by the sensitivity grid of [scr_capital\(\)](#) and by the scenario engine of [scr_ecl\(\)](#). Arguments are recycled.

Usage

```
scr_pd_stress(pd, rho, q)
```

Arguments

pd	Numeric vector of unconditional PDs.
rho	Asset correlation in $[0, 1)$.
q	Confidence level in $(0, 1)$; 0.5 returns the median-year PD.

Value

A numeric vector of conditional PDs.

References

Vasicek, O. (2002). The distribution of loan portfolio value. *Risk*, 15(12), 160-162. Gordy, M. B. (2003). A risk-factor model foundation for ratings-based bank capital rules. *Journal of Financial Intermediation*, 12(3), 199-232.

See Also

[scr_pd_pit_ttc\(\)](#), the same bridge with the systematic factor given as a value of z rather than a quantile q .

Other irb-capital: [scr_capital\(\)](#), [scr_ecl\(\)](#), [scr_el\(\)](#), [scr_irb_rw\(\)](#), [scr_sa_rw\(\)](#)

Examples

```
scr_pd_stress(0.02, rho = 0.15, q = c(0.5, 0.95, 0.99, 0.999))
```

scr_pd_validate	<i>Validate a PD model on a cohort panel</i>
-----------------	--

Description

Runs the standard battery on a monthly panel with the default flag and the grade (or the score) at every month: obligors non-defaulted at each cohort start form the population, the outcome is a default within horizon months, exactly as [scr_default_rate\(\)](#) does.

Usage

```
scr_pd_validate(
  x,
  newdata,
  id = "id",
  date = "date",
  default = "default",
  grade = NULL,
  score = NULL,
  auc_init = NULL,
  cv_init = NULL,
  tests = c("jeffreys", "binomial", "normal", "hl", "multi_period", "auc",
    "concentration", "psi", "migration"),
  alpha = 0.05,
  lights = NULL,
  pd_column = c("pd_final", "pd_moc", "pd_be"),
  horizon = 12L,
  by = NULL,
  n_boot = NULL,
  seed = NULL
)
```

Arguments

x	An scr_pd() object.
newdata	A data.frame/data.table panel, one row per id and month.
id, date, default	Column names.
grade	Column name of the grade at every month; NULL derives it from score with the cut points of x.
score	Column name of the production score at every month, optional.
auc_init	Development AUC; NULL uses the scorecard's hold-out AUC.

cv_init	Development coefficient of variation; NULL uses the one of x.
tests	Subset of the battery to run.
alpha	Significance level of the binomial critical count.
lights	Two p-value thresholds (red at or below the first, amber at or below the second, green above; the convention shared with the LGD and EAD validations); NULL reads config\$pd_lights.
pd_column	Grade PD tested: "pd_final" (default), "pd_moc" or "pd_be".
horizon, by	Cohort window in months and frequency (NULL reads config\$pd_dr_by).
n_boot, seed	Bootstrap resamples and seed of the discrimination interval.

Details

Calibration Per grade (pooled over cohorts) and per cohort and grade: Jeffreys $p = F_Beta(PD; D + 1/2, N - D + 1/2)$, the binomial $P(X \geq D)$ with its critical count at alpha, the normal z , and the traffic light on the Jeffreys p -value. Portfolio: the same tests on the totals, Hosmer-Lemeshow over the grades (K degrees of freedom: the grade PDs are not fitted on the validation sample), the multi-period normal test over the cohort differences $DR_t - PD_t$ (BCBS Working Paper 14, 2005) and the Brier score.

Discrimination AUC, Gini and KS with a bootstrap interval ([scr_metrics\(\)](#)) on the score when a score column exists, otherwise on the grade; the S statistic against auc_init ($(AUC_init - AUC_curr) / se$, with the DeLong standard error of the current AUC), $p = 1 - \Phi(S)$.

Stability PSI of the grade distribution against the development sample per cohort ([scr_psi\(\)](#)); the migration matrix pooled over the cohorts whose end date is observed ([scr_migration\(\)](#)); the concentration test on the coefficient of variation of the latest cohort against cv_init .

Value

An object of class `scr_pd_validation`: `calibration` (per grade, pooled), `calibration_cohort` (per cohort and grade), `portfolio` (per cohort), `portfolio_tests` (list: `n`, `d`, `dr`, `pd`, `p_jeffreys`, `p_binomial`, `hl_chi2`, `hl_df`, `hl_p`, `multi_period_z`, `multi_period_p`, `brier`), `discrimination`, `stability` (`psi` table, `migration`, `concentration`), `summary` (one row per test with statistic, `p_value`, `light`), `light` (the worst light of the summary), `n_cohorts`, `alpha`, `lights`. `portfolio_tests` also carries `critical`, `z`, `p_normal`, `n_cohorts` and `pd_column`; the object also has `horizon`, `by`, `pd_column` and `target`.

See Also

Other `irb-pd`: [predict.scr_grades\(\)](#), [predict.scr_pd\(\)](#), [scr_calibrate\(\)](#), [scr_grades\(\)](#), [scr_master_scale\(\)](#), [scr_migration\(\)](#), [scr_moc\(\)](#), [scr_pd\(\)](#), [scr_pd_pit_ttc\(\)](#)

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  use_lightgbm = FALSE, xgb_rounds = 40, n_boot = 10)
res <- scr_select(scr_demo, "default", config = cfg, drop = c("id", "churn"),
                  date_col = "ref_date")
sc <- scr_scorecard(res)
```

```
pd <- scr_pd(scr_moc(scr_grades(sc, n_grades = 6, min_defaults = 10), "C", method = "ci_binomial"))
# the validation panel: default flag at every month plus the grade at the
# cohort start; here the behavioural score of the panel is graded with the
# cut points of the PD model
d <- scr_default(scr_demo_panel, "id", "ref_date", dpd = "dpd", config = cfg)
pnl <- merge(d$flags, scr_demo_panel[, c("id", "ref_date", "score")],
            by.x = c("id", "date"), by.y = c("id", "ref_date"))
pnl$grade <- predict(pd, score = pnl$score, type = "grade")
v <- scr_pd_validate(pd, pnl, id = "id", date = "date", default = "default",
                    grade = "grade", score = "score", by = "quarter")

v
v$summary
```

scr_presets	<i>Selection presets, side by side</i>
-------------	--

Description

Returns the funnel keys resolved per preset, to compare before choosing. `target_min` and `iv_max` are shown for context; the presets leave them unchanged, as they do every other configuration key.

Usage

```
scr_presets()
```

Value

A `data.frame` with one row per preset.

See Also

Other configuration: [scr_config\(\)](#), [scr_config_keys\(\)](#), [scr_verbose\(\)](#)

Examples

```
scr_presets()
```

scr_psi	<i>Population stability index, with the fixed and the sample-size-adjusted threshold</i>
---------	--

Description

$PSI = \sum((p - q) * \ln(p / q))$ over bins frozen on the base. Reports both thresholds side by side: the traditional fixed one (< 0.10 "stable", $0.10-0.25$ "moderate", ≥ 0.25 "shift") and the sample-size-adjusted critical value of Yurdakul and Naranjo (2020), under which the PSI is asymptotically $(1/n + 1/m) * \text{chi-squared}(B - 1)$. With $n = m = 1000$ and ten bins the 5% critical value is 0.034, not 0.10; on a monthly base of a hundred thousand rows, $PSI = 0.01$ is already significant. The fixed threshold remains what the market knows; the adjusted one is what the statistics support.

Usage

```
scr_psi(
  base,
  compare,
  levels = NULL,
  breaks = NULL,
  n_groups = 10L,
  alpha = 0.05,
  thresholds = c(0.1, 0.25)
)
```

Arguments

base	Reference vector (the "development" distribution).
compare	Vector to compare.
levels	For categorical vectors: the levels to consider. NULL uses the union of the observed ones.
breaks	For numeric vectors: frozen cut points. NULL derives n_groups quantiles of base.
n_groups	Number of bands when breaks = NULL.
alpha	Significance level of the adjusted threshold.
thresholds	The two fixed thresholds: below the first the flag is "stable", below the second "moderate", otherwise "shift".

Details

Rows where base or compare is NA, or that fall outside breaks or levels, are not counted. A band empty in both samples is left out of the index and of the degrees of freedom $B - 1$; when a populated band is empty on one side only, 0.5 is added to every populated band of both samples.

Value

A list of class scr_psi with psi, flag_fixed, critical (adjusted critical value), flag_adjusted ("stable" or "shift"), n_base, n_compare, n_bins (bands declared; the degrees of freedom count only the populated ones) and table (per band: n_base, n_compare, pct_base, pct_compare, psi_band). The thresholds and alpha used are stored and printed.

References

Yurdakul, B. and Naranjo, J. (2020). Statistical properties of the population stability index. *Journal of Risk Model Validation*, 14(4), 89-100.

See Also

Other metrics: [scr_iv\(\)](#), [scr_metrics\(\)](#)

Examples

```
set.seed(2)
base <- stats::rnorm(5000)
new <- stats::rnorm(5000, mean = 0.15)
p <- scr_psi(base, new)
p
p$table
```

scr_reasons	<i>Reason codes: the variables that took the most points from each row</i>
-------------	--

Description

For each row of newdata, the k variables whose contribution in points fell furthest below the reference. The reference is the mean points of the variable on the training population ("mean", the Regulation B safe harbour referenced to the average) or the maximum points of the variable ("max"). Only applies to the additive scorecard; a tree challenger has no reason codes.

Usage

```
scr_reasons(x, newdata, k = 4L, reference = c("mean", "max"))
```

Arguments

x	An object from scr_scorecard() .
newdata	New table.
k	Number of reasons per row.
reference	"mean" (default) or "max".

Details

Under higher_is_riskier the shortfall is measured the other way round: the reasons are the variables that added the most points.

Value

A data.table with reason_1 ... reason_k (variable names) and shortfall_1 ... shortfall_k (points below the reference).

References

12 CFR 1002.9 (Regulation B), official commentary to paragraph 9(b)(2).

See Also

Other production: [predict.scr_align\(\)](#), [scr_apply\(\)](#), [scr_export\(\)](#), [scr_monitor\(\)](#), [scr_monitoring_plan\(\)](#), [scr_sql\(\)](#)

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
sc <- scr_scorecard(res)
scr_reasons(sc, head(scr_demo, 5), k = 3)
```

scr_reject

Stage 6: honest reject inference through a sensitivity band

Description

Does not ship parcelling as the default behaviour: instead of inventing a single multiplier and reweighting, it declares the **population scope** of the scorecard, measures the **coverage per band** (where an observed outcome exists, and in what volume) and presents a **sensitivity band**: the event rate each band would have if the population without an outcome were 2, 4 or 8 times worse than the observed one, with the effect on the total. The analyst reads the band; no single number is fabricated.

Usage

```
scr_reject(
  x,
  population = NULL,
  accepted = NULL,
  multipliers = NULL,
  sample = "holdout"
)
```

Arguments

x	An object from scr_scorecard() .
population	Optional: a table of the full population (accepted and rejected, without outcome), scored by scr_apply() . NULL restricts the scope to the population with an outcome.
accepted	Optional: a logical vector, of the length of population, marking the rows with an observed outcome. NULL treats the whole population as without an outcome beyond the development sample.
multipliers	Sensitivity band. NULL uses the configuration.
sample	Reference sample of the observed outcomes.

Value

An `scr_reject` object with scope, coverage (per band) and sensitivity (per band and multiplier, plus the TOTAL row).

See Also

Other stages: [scr_align\(\)](#), [scr_bin\(\)](#), [scr_cutoff\(\)](#), [scr_model\(\)](#), [scr_scorecard\(\)](#), [scr_select\(\)](#), [scr_split\(\)](#), [scr_strategy\(\)](#), [scr_triage\(\)](#)

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
sc <- scr_scorecard(res)
scr_reject(sc)
# with a through-the-door population: rows with an outcome are the hold-out
acc <- seq_len(nrow(scr_demo)) %in% res$split$holdout_idx
scr_reject(sc, population = scr_demo, accepted = acc)
```

scr_result

*Result of a selection***Description**

Object returned by [scr_select\(\)](#). The methods below are the supported way of inspecting the result in the console; to extract data, use the accessors ([scr_selected\(\)](#), [scr_funnel\(\)](#), [scr_gains\(\)](#)).

Usage

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

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

## S3 method for class 'scr_result'
as.data.frame(x, ...)

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

Arguments

x, object An scr_result object.

... Ignored, present for compatibility with the generic.

Value

print() and plot() return x invisibly; summary() returns an scr_summary object; as.data.frame() returns the funnel.

See Also

Other accessors: [scr_funnel\(\)](#), [scr_gains\(\)](#), [scr_leakage\(\)](#), [scr_score_gains\(\)](#), [scr_score_metrics\(\)](#), [scr_selected\(\)](#)

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
res                                # print: the funnel in one screen
summary(res)                      # full text report
head(as.data.frame(res))          # the funnel as a data.frame
plot(res)
```

scr_run

*Run the selection for several targets straight from the database***Description**

For each target: fetches the table, runs [scr_select\(\)](#) and writes the deliverables. A failure on one target is recorded and the loop continues.

Usage

```
scr_run(
  con,
  table,
  targets,
  config = scr_config(),
  drop = character(),
  date_col = config$oot_date_col,
  event_level = NULL,
  sample_frac = 1,
  max_rows = NULL,
  export = NULL
)
```

Arguments

con	A DBI connection, from scr_connect() .
table	Table name, with an optional {target}.
targets	Vector with the names of the target columns.
config	An object from scr_config() , used for every target.
drop	Columns that are never candidates.

date_col	Date column of the out-of-time split, passed on to <code>scr_select()</code> . Defaults to <code>config\$oot_date_col</code> ; an explicit NULL forces a random stratified split.
event_level	Passed on to <code>scr_select()</code> .
sample_frac	Sampling fraction. A scalar or a list named by target.
max_rows	Row cap per target. NULL switches it off.
export	Root output directory; each target writes to a subdirectory.

Value

An `scr_runset` object: a named list of `scr_result` (or, for the targets that failed, a list with error).

Table convention

`table` accepts the `{target}` placeholder, replaced by the lower-case target name. Without the placeholder, the same table is used for every target.

See Also

`scr_compare()` and `scr_core()` to read the run set.
Other portfolio: `scr_compare()`, `scr_core()`, `scr_runset`

Examples

```
con <- scr_connect(driver = RSQLite::SQLite(), dbname = ":memory:")
d <- scr_demo; d$ref_date <- as.character(d$ref_date) # SQLite has no Date type
DBI::dbWriteTable(con, "dtm", d)
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
rs <- scr_run(con, "dtm", targets = c("default", "churn"), config = cfg,
             drop = c("id", "ref_date", "default", "churn"))

rs
scr_compare(rs)
DBI::dbDisconnect(con)
```

scr_runset	<i>Set of runs, one per target</i>
------------	------------------------------------

Description

Object returned by `scr_run()`: a named list of `scr_result`, plus the errors of the targets that failed. Use `scr_compare()` for the comparison table and `scr_core()` for the variables that cross several targets.

Usage

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

Arguments

x An `scr_runset` object.
 ... Ignored.

Value

x, invisibly.

See Also

Other portfolio: [scr_compare\(\)](#), [scr_core\(\)](#), [scr_run\(\)](#)

Examples

```
con <- scr_connect(driver = RSQLite::SQLite(), dbname = ":memory:")
d <- scr_demo[, c("default", "ds_region", "ds_band", "vl_score_01",
                 "vl_score_02", "vl_score_05", "vl_hist_01")]
DBI::dbWriteTable(con, "dtm", d)
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                 use_lightgbm = FALSE, xgb_rounds = 40, n_boot = 10)
rs <- scr_run(con, "dtm", targets = "default", config = cfg)
rs
names(rs)
DBI::dbDisconnect(con)
```

scr_sa_rw

*Standardised risk weight of an exposure***Description**

Lookup in `params$sa_rw`: regulatory retail ("retail_other", "qrre_*": 75 %, or the transactor weight), residential mortgages by loan-to-value band (a missing LTV takes the highest band), corporates by external rating bucket ("AAA" to "AA-", "A", "BBB", "BB", below; NA is unrated; "IG" marks an unrated investment-grade obligor where ratings are not used) or the SME weight, banks and sovereigns through the corporate rating rows, and defaulted exposures by the specific provision ratio (or the mortgage row). Arguments are recycled.

Usage

```
scr_sa_rw(
  asset_class,
  ltv = NULL,
  rating = NULL,
  transactor = NULL,
  defaulted = NULL,
  provision_ratio = NULL,
  sme = NULL,
```

```

    granular = TRUE,
    params = scr_irb_params("bcb")
)

```

Arguments

asset_class	One of "corporate", "corporate_sme", "bank", "sovereign", "hvcre", "retail_mortgage", "qrre_revolver", "qrre_transactor", "retail_other"; a scalar or a vector.
ltv	Loan-to-value at origination, decimal (mortgages).
rating	External rating string (corporates), NA when unrated.
transactor	Logical: revolving facility repaid in full every month.
defaulted	Optional 0/1 or logical vector.
provision_ratio	Specific provisions over the outstanding amount (defaulted rows).
sme	Logical: corporate small or medium enterprise (also implied by asset_class = "corporate_sme").
granular	Logical (scalar or per exposure): whether the retail exposure belongs to a granular regulatory retail pool; FALSE applies the non-granular retail weight.
params	An <code>scr_irb_params()</code> object.

Value

A numeric vector of standardised risk weights (decimals).

References

Basel Committee on Banking Supervision (2023). *The Basel Framework*, CRE20 (standardised approach: individual exposures).

See Also

Other irb-capital: `scr_capital()`, `scr_ecl()`, `scr_el()`, `scr_irb_rw()`, `scr_pd_stress()`

Examples

```

scr_sa_rw(c("retail_other", "retail_mortgage", "corporate"), ltv = c(NA, 0.55, NA),
  rating = c(NA, NA, "A+"))
scr_sa_rw("retail_other", defaulted = TRUE, provision_ratio = c(0.1, 0.3))

```

scr_scorecard

Stages 4 and 5: points scorecard, aligned to the declared scale

Description

Fits a logistic regression on the WOE columns of the shortlist, checks the sign of the coefficients, aligns the logit to the declared scale with [scr_align\(\)](#) (always) and distributes the points per bin. Measures the score on train and hold-out with a bootstrap CI (always), builds the gains with bands **frozen on train**, the score PSI and the CSI per variable (fixed and n-adjusted thresholds), the calibration and the rank-order diagnostics. Optionally fits a tree challenger on the same WOE columns, aligned to the same scale, with an explicit `supports_scorecard = FALSE`: it compares, it never produces points or reason codes.

Usage

```
scr_scorecard(
  x,
  features = NULL,
  base_score = NULL,
  base_odds = NULL,
  pdo = NULL,
  direction = NULL,
  align_method = NULL,
  challenger = NULL,
  points_style = NULL,
  n_boot = NULL,
  seed = NULL
)
```

Arguments

<code>x</code>	An object from scr_select() .
<code>features</code>	Variables of the scorecard. Defaults to scr_selected() .
<code>base_score</code> , <code>base_odds</code> , <code>pdo</code> , <code>direction</code>	The scale; NULL uses the configuration of <code>x</code> . See scr_config() and scr_align() .
<code>align_method</code>	"regression" or "direct"; NULL uses the configuration.
<code>challenger</code>	NULL, "xgboost" or "lightgbm"; NULL uses the configuration.
<code>points_style</code>	"base_plus_deviation" or "distributed"; NULL uses the configuration.
<code>n_boot</code>	CI resamples; NULL uses the configuration.
<code>seed</code>	Seed; NULL uses the configuration.

Value

An `scr_scorecard` object. Main components: `features`, `coef`, `sign_check`, `alignment` (an `scr_align` object), `points`, `base_points`, `samples` (train and hold-out: `link`, `prob`, `score`, `score_points`, `y`, `date`), `metrics`, `gains`, `stability` (score and variables), `calibration`, `rank_order`, `challenger`, `model_card` and `sql`. Also `scale` (`base_score`, `base_odds`, `pdo`, `factor`, `offset`, `direction`, `odds_orientation`), `breaks` (the score bands frozen on train), `monitoring_plan` (see `scr_monitoring_plan()`), `holdout_bins`, `fit` and `ledger` (the frozen binning and pre-processing that `scr_apply()` and `scr_sql()` reproduce) and, after a lab commit, `decisions` and `provenance`.

Sign check

The engine's WOE is event-oriented, so every glm coefficient must be positive. A variable with a non-positive coefficient (or above `max_abs_coef` in absolute value) is explaining what another already explained, with the sign reversed; it is removed and the model refitted, one at a time, the most negative first, and each removal is recorded in `sign_check`. The last remaining variable is never removed: it is kept and flagged `NON_POSITIVE_COEF_KEPT_LAST`. The final shortlist of the scorecard (features) is what `scr_sql()` covers.

Points per bin

With $\text{score} = a + b * \text{logit}$ and $\text{logit} = \alpha + \sum(\beta_j * \text{woe}_{ij})$:

$$\text{points}_{ij} = b\beta_j \text{woe}_{ij}, \quad \text{base} = a + b\alpha.$$

`points_style = "distributed"` spreads `base / k` over each characteristic (Siddiqi, 2006, chapter 6), leaving `base_points = 0`. The exact points stay in `points_raw`; `points` is the rounded version when `points_round = TRUE`. The exact score (`score`) and the whole-points score (`score_points`) are both returned by `scr_apply()` and both emitted by `scr_sql()`. A row that falls in no fitted bin (a category never seen on train, a missing value without a missing bin) gets a WOE of 0 from the binning engine, hence the points of a WOE of 0: 0, or `base / k` under "distributed".

See Also

Other stages: `scr_align()`, `scr_bin()`, `scr_cutoff()`, `scr_model()`, `scr_reject()`, `scr_select()`, `scr_split()`, `scr_strategy()`, `scr_triage()`

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
sc <- scr_scorecard(res)
sc
head(sc$points[, c("variable", "bin", "woe", "points")])
sc$metrics
sc$alignment
```

scr_score_gains	<i>Score gains per frozen band</i>
-----------------	------------------------------------

Description

How the score behaves in each band: count, event rate, KS, lift, cumulative capture and the score interval of the band, which is what lets a cut-off be read straight from the table. The bands are the deciles of the score on **train**, applied frozen to the other samples.

Usage

```
scr_score_gains(x, sample = NULL)
```

Arguments

x	An object from scr_scorecard() .
sample	NULL (all), "train" or "holdout".

Value

A data.table with one row per sample and band, from the riskiest band to the safest.

See Also

Other accessors: [scr_funnel\(\)](#), [scr_gains\(\)](#), [scr_leakage\(\)](#), [scr_result](#), [scr_score_metrics\(\)](#), [scr_selected\(\)](#)

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
  date_col = "ref_date")
sc <- scr_scorecard(res)
scr_score_gains(sc, "holdout")[, .(band, n, event_rate, min_score, max_score, ks)]
scr_score_metrics(sc)
```

scr_score_metrics	<i>Score metrics per sample, with CI</i>
-------------------	--

Description

n, events, AUC, KS and Gini of the scorecard score on train and hold-out, with a bootstrap confidence interval and the direction used: the AUC is always reported above 0.5 when the score ranks correctly in its own direction.

Usage

```
scr_score_metrics(x)
```

Arguments

x An object from `scr_scorecard()`.

Value

A data.table with one row per sample.

See Also

Other accessors: `scr_funnel()`, `scr_gains()`, `scr_leakage()`, `scr_result`, `scr_score_gains()`, `scr_selected()`

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
sc <- scr_scorecard(res)
scr_score_metrics(sc)
```

scr_select

Select variables for the scorecard

Description

Shortcut that chains `scr_split()`, `scr_triage()`, `scr_bin()` and `scr_model()` on a table and a binary target, and returns an object with the shortlist, the complete audit funnel, the gains table and the production SQL of the approved variables. Every stage remains callable on its own for whoever wants more control (hybrid interface).

Usage

```
scr_select(
  data,
  target,
  config = scr_config(),
  drop = character(),
  date_col = config$boot_date_col,
  event_level = NULL,
  export = NULL,
  copy = TRUE
)
```

Arguments

data	A data.frame or data.table with the target, the candidates and, if any, the date column of the out-of-time cut.
target	Name of the target column (0/1, logical, or a two-level factor/character).
config	An object from scr_config() .
drop	Columns that are never candidates. They stay in the funnel as 00.config.
date_col	Date column of the out-of-time cut. Defaults to config\$oot_date_col.
event_level	Which target value counts as the event; see scr_split() .
export	Directory to write the deliverables to. NULL (default) writes nothing; use scr_export() later.
copy	If TRUE (default), works on a copy of data.

Value

An object of class `scr_result`. Read it with [scr_selected\(\)](#), [scr_funnel\(\)](#), [scr_gains\(\)](#), [scr_sql\(\)](#), [scr_leakage\(\)](#) and [summary\(\)](#); continue with [scr_scorecard\(\)](#); write it with [scr_export\(\)](#).

Reproducibility

With the same data, the same target and the same `config$seed`, the result is identical with one or several nthread: the seed governs the random split, the cross-validation, the classifier subsample, the trees and the bootstrap, and the binning is deterministic per column.

See Also

[scr_run\(\)](#) for several targets straight from the database, [scr_scorecard\(\)](#) for the next step.

Other stages: [scr_align\(\)](#), [scr_bin\(\)](#), [scr_cutoff\(\)](#), [scr_model\(\)](#), [scr_reject\(\)](#), [scr_scorecard\(\)](#), [scr_split\(\)](#), [scr_strategy\(\)](#), [scr_triage\(\)](#)

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
  date_col = "ref_date")
res
scr_selected(res)
head(scr_funnel(res, only_selected = TRUE))
```

scr_selected	<i>Variables approved for the scorecard</i>
--------------	---

Description

The final shortlist, in consensus order (the first is the strongest). It is exactly the list `scr_sql()` covers and `scr_scorecard()` fits.

Usage

```
scr_selected(x, which = c("final", "consensus", "manual"))
```

Arguments

x	An object from <code>scr_select()</code> .
which	"final" (default), "consensus" or "manual" (NULL when no manual choice was made).

Details

After `scr_classing_apply()` the result carries two lists: the automatic consensus and the analyst's final choice; which picks one, and the default is the final one so that every downstream function follows the analyst's decision.

Value

A character vector of column names.

See Also

Other accessors: `scr_funnel()`, `scr_gains()`, `scr_leakage()`, `scr_result`, `scr_score_gains()`, `scr_score_metrics()`

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
scr_selected(res)
```

scr_split

*Stage 0: type the data and split train and hold-out***Description**

First stage of the pipeline, callable on its own. Converts the target to 0/1 (resolving event_level), types the candidates (numerics become double, everything else becomes character) and splits train and hold-out **before** any supervised fit.

Usage

```
scr_split(
  data,
  target,
  date_col = NULL,
  ratio = 0.3,
  seed = NULL,
  event_level = NULL,
  drop = character(),
  copy = TRUE
)
```

Arguments

data	A data.frame or data.table with the target and the candidates.
target	Name of the target column. Binary: 0/1, logical, or a two-level factor/character.
date_col	Date column of the out-of-time cut. NULL uses a random stratified split.
ratio	Target hold-out fraction.
seed	Seed of the random split. NULL draws from the session's random stream (reproducible only through a set.seed() of your own; scr_select() passes the seed of scr_config()). A seed is applied locally: the session's random stream is restored on exit.
event_level	Which target value counts as the event. NULL uses the convention (1, or the second alphabetical level).
drop	Columns that are never candidates (identifiers, sibling targets, free text). They stay in the funnel as 00.config.
copy	If TRUE (default), works on a copy of data. FALSE modifies a data.table by reference (target and typing), saving memory; a data.frame is always converted, hence copied.

Details

The split prefers out-of-time by date_col: it is the only one that tests generalisation to a future period. The cut is made on the **distinct** date values, not by row quantile: it picks the smallest set of most recent periods that already reaches ratio of the population. Without a date column, or

with a single period, it falls back to random stratified by the target. The date column is never a candidate: it is the key of the split and leaves the contest. A text date column is read as an ISO date (YYYY-MM-DD, YYYY/MM/DD, YYYY-MM) or as all-digit periods (YYYYMM); rows with a missing date belong to no period and are left out of both train and hold-out, with a warning in the log.

Value

An `scr_split` object with `data` (typed), `target`, `train_idx`, `holdout_idx`, `method`, `cutoff`, `date_col` and `cols` (features, `var_num`, `var_cat`, `dropped`, `event`).

Event orientation

`event_level` changes **what is modelled**. Passing 0 makes class 0 the event: the sign of every WOE flips, the emitted SQL changes, the points change. For a text target, the second level in alphabetical order is the event by default, and the choice is always reported. Not to be confused with `config$objective`, which only orients the reading and the points scale.

See Also

Other stages: [scr_align\(\)](#), [scr_bin\(\)](#), [scr_cutoff\(\)](#), [scr_model\(\)](#), [scr_reject\(\)](#), [scr_scorecard\(\)](#), [scr_select\(\)](#), [scr_strategy\(\)](#), [scr_triage\(\)](#)

Examples

```
sp <- scr_split(scr_demo, "default", date_col = "ref_date", drop = "id")
sp
length(sp$train_idx); length(sp$holdout_idx)
```

scr_sql

Production SQL

Description

Code ready to run in the database, covering **exactly** the approved variables (or those of the scorecard), in blocks in this order:

Usage

```
scr_sql(x, table = NULL, dialect = NULL, file = NULL, ...)

## S3 method for class 'scr_capital'
scr_sql(
  x,
  table = NULL,
  dialect = NULL,
  file = NULL,
  level = c("exposure", "portfolio"),
  ...
```

```

)

## S3 method for class 'scr_ead'
scr_sql(x, table = NULL, dialect = NULL, file = NULL, ...)

## S3 method for class 'scr_lgd'
scr_sql(x, table = NULL, dialect = NULL, file = NULL, ...)

## S3 method for class 'scr_pd'
scr_sql(x, table = NULL, dialect = NULL, file = NULL, ...)

## S3 method for class 'scr_result'
scr_sql(x, table = NULL, dialect = NULL, file = NULL, output = NULL, ...)

## S3 method for class 'scr_scorecard'
scr_sql(
  x,
  table = NULL,
  dialect = NULL,
  file = NULL,
  what = c("score", "woe", "all"),
  keep_columns = NULL,
  ...
)

```

Arguments

x	An object from <code>scr_select()</code> , <code>scr_scorecard()</code> , <code>scr_pd()</code> , <code>scr_lgd()</code> , <code>scr_ead()</code> or <code>scr_capital()</code> .
table	Source table name, written verbatim (it may be qualified, <code>schema.table</code> , and is never quoted: pass only a trusted name). NULL uses <code>config\$sql_table</code> .
dialect	Dialect ("ansi", "databricks", "spark", "hive", "mysql", "mariadb", "sqlserver", "bigquery", "postgres", "oracle", "snowflake", "redshift", "duckdb", "sqlite"). NULL uses <code>config\$sql_dialect</code> .
file	Path to write to. NULL (default) returns the lines.
...	Passed on to the methods.
level	For <code>scr_capital</code> : "exposure" (default, one row per exposure with <code>el</code> , <code>k</code> , <code>rw</code> , <code>rwa</code>) or "portfolio" (the aggregate by segment).
output	For <code>scr_result</code> : "woe", "bin" or "both". NULL uses <code>config\$sql_output</code> .
what	For <code>scr_scorecard</code> : "score" (default: the three blocks, with the points per variable, the exact score and the whole-points score), "woe" (the WOE/BIN SQL of the scorecard variables only) or "all" (the three blocks plus, for every variable, its bin label, WOE and points side by side: the deployment layout that reports the band of each variable next to the score).
keep_columns	For <code>scr_scorecard</code> : key columns carried untransformed into the output (for example the customer identifier and the reference date); NULL uses <code>config\$sql_keep_columns</code> .

Details

1. CTE base_scr: reproduces the Stage 1 pre-processing - imputation of missing and sentinel by the **training** median, special-population flags, COALESCE of the categorical missing.
2. The WOE/BIN transformation, emitted by `OptimalBinningWoE::obwoe_sql()` from the authoritative cut points with full precision.
3. (Scorecard) CTE woe_scr with WOE and bin index, followed by the final SELECT with score (exact, $a + b * \text{logit}$), `<f>_points` per variable and `score_points` (whole points).

The order matters: without the first block, the WOE would be applied to data different from what was binned. Column names are quoted with the dialect's delimiters only when they are not plain identifiers or are reserved words, the same rule `OptimalBinningWoE::obwoe_sql()` applies, so every block names a column the same way. A row whose value falls in no fitted bin (a category never seen on train) takes a WOE of 0 and the points of a WOE of 0, in the SQL as in `scr_apply()`. The score computed by the SQL matches `scr_apply()` numerically, by an automated test that runs both paths.

Value

A character vector with the SQL (invisibly, when file is given).

IRB models

`scr_pd` wraps the scorecard SQL in a common table expression and adds a CASE on the score cut points that yields `grade` and `pd_final`. `scr_lgd` chains the driver bins of both stages, the logits, the pool CASE and the floored result. `scr_ead` computes the utilisation and the undrawn amount, assigns the pool from the frozen cut points and applies the greatest of the model, the drawn amount and the standardised floor. `scr_capital` carries the constants of every pool (PD, LGD, k, risk weight) in a `pool_params` table joined on segment and grade, so no normal quantile is evaluated at run time; `level` chooses the exposure or the portfolio output.

See Also

Other production: `predict.scr_align()`, `scr_apply()`, `scr_export()`, `scr_monitor()`, `scr_monitoring_plan()`, `scr_reasons()`

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
                  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
                  date_col = "ref_date")
cat(head(scr_sql(res, table = "prd.customers", dialect = "databricks"), 20), sep = "\n")
sc <- scr_scorecard(res)
cat(tail(scr_sql(sc), 12), sep = "\n")
```

scr_strategy

*Stage 6: strategy table per band, with marginal expected profit***Description**

Score bands (by default the deciles frozen on train) with volume, event rate, decision and the expected result per account:

$$EP = (1 - p) \text{revenue_good} - p \text{loss_bad},$$

which makes visible the band that is profitable **at the margin** even with a high event rate. The break-even event rate, where $EP = 0$, is $\text{revenue_good} / (\text{revenue_good} + \text{loss_bad})$.

Usage

```
scr_strategy(
  x,
  breaks = NULL,
  decisions = NULL,
  revenue_good = 1,
  loss_bad = 1,
  sample = "holdout"
)
```

Arguments

x	An object from scr_scorecard() .
breaks	Band cut points. NULL uses the deciles frozen on train.
decisions	Vector of decisions, one per band (from the safest to the riskiest). NULL derives them from break-even.
revenue_good	Expected revenue per account without the event (default 1).
loss_bad	Expected loss per account with the event (default 1; with both defaults the break-even event rate is 50%).
sample	"holdout" (default) or "train".

Details

The automatic decision approves a band whose rate is below break-even, sends to review a band up to 25% above it and declines the rest; pass decisions to fix the policy.

Value

An scr_strategy object with table, breakeven and the parameters.

See Also

Other stages: [scr_align\(\)](#), [scr_bin\(\)](#), [scr_cutoff\(\)](#), [scr_model\(\)](#), [scr_reject\(\)](#), [scr_scorecard\(\)](#), [scr_select\(\)](#), [scr_split\(\)](#), [scr_triage\(\)](#)

Examples

```
cfg <- scr_config(verbose = FALSE, nthread = 1, use_ranger = FALSE,
  xgb_rounds = 60, n_boot = 20)
res <- scr_select(scr_demo, "default", config = cfg, drop = "id",
  date_col = "ref_date")
sc <- scr_scorecard(res)
scr_strategy(sc, revenue_good = 1080, loss_bad = 4500)
```

scr_triage

Stage 1: descriptive triage and sentinel resolution

Description

Profiles every candidate **on the training rows only**, decides its fate and materialises the clean data for train and hold-out with the same values (training median, "MISSING" level). A sentinel or missing mass with weight (special_min_share) and signal (special_min_woe) becomes a categorical flag <column><flag_suffix>, which the engine bins and emits in SQL natively.

Usage

```
scr_triage(split, config = scr_config())
```

Arguments

split	An object from scr_split() .
config	An object from scr_config() .

Details

Failures at this stage: CONSTANT, NEAR_CONSTANT, TOO_MANY_MISSING, HIGH_CARDINALITY, NO_SIGNAL (coarse IV below min_iv_quick) and DUPLICATE_OF:<column>. A failed numeric whose special-population flag survives carries the suffix ;RESCUED_AS_FLAG.

Value

An scr_triage object with profile (one row per candidate and derived flag), ledger (the source of truth of the pre-processing the SQL reproduces), keep, derived, clean (target + survivors + flags, with no NA) and the originating split.

See Also

Other stages: [scr_align\(\)](#), [scr_bin\(\)](#), [scr_cutoff\(\)](#), [scr_model\(\)](#), [scr_reject\(\)](#), [scr_scorecard\(\)](#), [scr_select\(\)](#), [scr_split\(\)](#), [scr_strategy\(\)](#)

Examples

```
sp <- scr_split(scr_demo, "default", date_col = "ref_date", drop = "id")
tr <- scr_triage(sp, scr_config(verbose = FALSE))
tr
table(tr$profile$trriage_reason)
```

scr_verbose	<i>Switch progress messages on or off</i>
-------------	---

Description

Large tables take tens of minutes, and the pipeline reports every stage as it runs: stage name, input and output counts, elapsed time. Messages go through `message()` as single lines, with no progress bar that redraws itself, so that a scheduled job (Rscript in batch) produces a readable log. `suppressMessages()` works too; this function exists to switch them off persistently, without wrapping every call. The verbose key of `scr_config()` has the same effect per run.

Usage

```
scr_verbose(on = NULL)
```

Arguments

on	TRUE to switch on, FALSE to switch off, NULL (default) to query the current state only.
----	---

Value

The verbosity state in force *before* the call, invisibly, so that `old <- scr_verbose(FALSE); ...; scr_verbose(old)` restores it.

See Also

Other configuration: `scr_config()`, `scr_config_keys()`, `scr_presets()`

Examples

```
old <- scr_verbose(FALSE) # silence, keeping the previous state
scr_verbose(old)          # restore
scr_verbose()             # query
```

scr_workout	<i>Workout LGD: the reference data set from default events and cash flows</i>
-------------	---

Description

Builds the reference data set (RDS) of realised loss given default, one row per default event, from a table of default events and the long table of their post-default cash flows. Every cash flow is discounted to the default date at the reference rate in force at that date plus `lgd_discount_add_on`, with monthly compounding over whole months:

$$PV = \frac{A}{(1 + r/12)^t}$$

where t is the number of whole months between the default date and the cash-flow date. The realised LGD is the economic loss

$$LGD = \frac{E - PV(R) + PV(C) + PV(D) + C^{ind}}{E}$$

with E the exposure at default, R recoveries, C direct costs, D drawings after default and C^{ind} the indirect costs allocated by `lgd_cost_allocation`.

Usage

```
scr_workout(
  defaults,
  cashflows,
  rates = NULL,
  config = scr_config(),
  indirect_costs = 0,
  obs_date = NULL,
  keep_rows = FALSE
)
```

Arguments

<code>defaults</code>	A <code>data.frame</code> or <code>data.table</code> with one row per default event: <code>default_id</code> , <code>facility_id</code> , <code>default_date</code> , <code>ead</code> , <code>product</code> , <code>status</code> ("closed", "cured" or "open"), optionally <code>close_date</code> , plus any driver columns, which are carried into the RDS.
<code>cashflows</code>	Long table: <code>default_id</code> , <code>date</code> , <code>amount</code> , <code>type</code> ("recovery", "direct_cost" or "drawing").
<code>rates</code>	Optional table (<code>date</code> , <code>rate</code>) of the annual reference rate; the rate in force at the default date is used. <code>NULL</code> uses the flat <code>lgd_discount_rate</code> of the configuration.
<code>config</code>	A <code>scr_config()</code> ; keys <code>lgd_*</code> .
<code>indirect_costs</code>	Total indirect workout cost to allocate: a single number, or a table (<code>product</code> , <code>amount</code>) allocated within product.

obs_date	Observation date; NULL uses the latest date seen.
keep_rows	Keep the cash-flow table with its present values.

Value

An object of class `scr_workout`: `rds` (one row per default event: `identifiers`, `default_date`, `ead`, `product`, `drivers`, `status`, `months_in_default`, `discount_rate`, `pv_recovery`, `pv_cost`, `pv_drawing`, `cost_indirect`, `recovery_extrapolated`, `lgd_raw`, `lgd_real`, `is_cure`, `is_incomplete`, `merged_n`), `recovery_profile` (product x month: `cum_recovery`), `extrapolation` (per open event), `funnel` (rule, n, action), `summary` (n, `cure_rate`, `lra_default_weighted`, `lra_exposure_weighted`, `share_incomplete`, `discount_rate_mean`, `by_product`, `by_year`), `ledger`, `config`, `obs_date`, and cashflows with `keep_rows`. `rds` also carries `year`, `close_date`, `recovery_nominal`, `recovery_artificial`, `cost_nominal`, `drawing_nominal`, `closed_at_t_max`, `absorbed`, `n_cashflows` and `last_month`; `funnel` has `kept`; `summary` has `n_cure`, `lra_raw`, `ead_total` and `years`.

Rules

- **Cures.** An event with `status == "cured"` returns to performing: the balance outstanding at the cure date (`ead` plus the drawings after default, net of the cash recovered) enters as an artificial recovery on the cure date, so the cure carries its costs and the discount effect, never a zero loss by decree.
- **Multiple defaults.** Two defaults of one facility separated by fewer than `lgd_cure_window` months (from the close of the first to the start of the second), or a new default while the first is still open, are one event: the first default date and exposure are kept, the cash flows of both spells are pooled and the status of the last spell rules.
- **Incomplete workouts.** An open event younger than `lgd_t_max` months receives the expected further recovery read from the recovery profile of the closed defaults of the same product (cumulative discounted recovery rate by month in default); an open event at or beyond `lgd_t_max` is treated as closed with no further recovery.
- **Bounds.** With `lgd_floor_at_zero` the realised LGD used in the averages is floored at zero and with `lgd_cap_at_one` capped at one; `lgd_raw` always keeps the unbounded value.

The long-run average is reported default-weighted (the arithmetic mean over events) and exposure-weighted, overall, by product and by calendar year of default.

See Also

Other `irb-lgd`: `scr_elbe()`, `scr_lgd()`, `scr_lgd_downturn()`, `scr_lgd_floor()`, `scr_lgd_pools()`, `scr_lgd_validate()`

Examples

```
cfg <- scr_config(verbose = FALSE)
wo <- scr_workout(scr_demo_lgd, scr_demo_lgd_cashflows, rates = scr_demo_rates, config = cfg)
wo
wo$funnel
head(wo$rds[, c("default_id", "product", "status", "lgd_raw", "lgd_real", "is_cure")])
```

Index

- * **accessors**
 - scr_funnel, [56](#)
 - scr_gains, [57](#)
 - scr_leakage, [64](#)
 - scr_result, [88](#)
 - scr_score_gains, [95](#)
 - scr_score_metrics, [95](#)
 - scr_selected, [98](#)
- * **classing**
 - scr_classing_accept, [18](#)
 - scr_classing_apply, [19](#)
 - scr_classing_choose, [20](#)
 - scr_classing_propose, [21](#)
 - scr_classing_spec, [22](#)
 - scr_classing_view, [23](#)
 - scr_coarse_classing, [24](#)
 - scr_decisions, [32](#)
- * **configuration**
 - scr_config, [27](#)
 - scr_config_keys, [29](#)
 - scr_presets, [84](#)
 - scr_verbose, [105](#)
- * **database**
 - scr_connect, [30](#)
 - scr_fetch, [55](#)
- * **datasets**
 - scr_demo, [36](#)
 - scr_demo_ead, [37](#)
 - scr_demo_lgd, [38](#)
 - scr_demo_lgd_cashflows, [39](#)
 - scr_demo_panel, [40](#)
 - scr_demo_portfolio, [41](#)
 - scr_demo_rates, [42](#)
- * **data**
 - scr_demo, [36](#)
 - scr_demo_ead, [37](#)
 - scr_demo_lgd, [38](#)
 - scr_demo_lgd_cashflows, [39](#)
 - scr_demo_panel, [40](#)
 - scr_demo_portfolio, [41](#)
 - scr_demo_rates, [42](#)
- * **irb-capital**
 - scr_capital, [15](#)
 - scr_ecl, [49](#)
 - scr_el, [51](#)
 - scr_irb_rw, [61](#)
 - scr_pd_stress, [81](#)
 - scr_sa_rw, [91](#)
- * **irb-ead**
 - scr_bin_continuous, [12](#)
 - scr_ead, [43](#)
 - scr_ead_data, [44](#)
 - scr_ead_downturn, [46](#)
 - scr_ead_validate, [47](#)
- * **irb-lgd**
 - scr_elbe, [52](#)
 - scr_lgd, [65](#)
 - scr_lgd_downturn, [66](#)
 - scr_lgd_floor, [67](#)
 - scr_lgd_pools, [68](#)
 - scr_lgd_validate, [69](#)
 - scr_workout, [106](#)
- * **irb-parameters**
 - scr_default, [33](#)
 - scr_default_rate, [35](#)
 - scr_irb_params, [60](#)
- * **irb-pd**
 - predict.scr_grades, [5](#)
 - predict.scr_pd, [6](#)
 - scr_calibrate, [13](#)
 - scr_grades, [58](#)
 - scr_master_scale, [71](#)
 - scr_migration, [73](#)
 - scr_moc, [74](#)
 - scr_pd, [79](#)
 - scr_pd_pit_ttc, [80](#)
 - scr_pd_validate, [82](#)
- * **metrics**

- scr_iv, 63
- scr_metrics, 72
- scr_psi, 84
- * **portfolio**
 - scr_compare, 26
 - scr_core, 31
 - scr_run, 89
 - scr_runset, 90
- * **production**
 - predict.scr_align, 4
 - scr_apply, 9
 - scr_export, 53
 - scr_monitor, 77
 - scr_monitoring_plan, 78
 - scr_reasons, 86
 - scr_sql, 100
- * **stages**
 - scr_align, 7
 - scr_bin, 10
 - scr_cutoff, 31
 - scr_model, 76
 - scr_reject, 87
 - scr_scorecard, 93
 - scr_select, 96
 - scr_split, 99
 - scr_strategy, 103
 - scr_triage, 104
- as.data.frame.scr_result (scr_result), 88
- DBI::dbConnect(), 30
- DBI::dbDisconnect(), 30
- message(), 105
- odbc::odbc(), 30
- OptimalBinningWoE::obwoe_algorithms(), 28
- OptimalBinningWoE::obwoe_sql(), 102
- plot.scr_result (scr_result), 88
- predict.scr_align, 4
- predict.scr_align(), 7, 10, 54, 77, 78, 86, 102
- predict.scr_grades, 5
- predict.scr_grades(), 6, 15, 59, 71, 74, 75, 79, 81, 83
- predict.scr_pd, 6
- predict.scr_pd(), 5, 15, 59, 71, 74, 75, 79, 81, 83
- print.scr_result (scr_result), 88
- print.scr_runset (scr_runset), 90
- scr_align, 7
- scr_align(), 4, 11, 14, 32, 76, 88, 93, 94, 97, 100, 103, 104
- scr_apply, 9
- scr_apply(), 4, 6, 12, 17, 24, 44, 54, 77, 78, 86, 87, 94, 102
- scr_bin, 10
- scr_bin(), 8, 32, 76, 88, 94, 96, 97, 100, 103, 104
- scr_bin_continuous, 12
- scr_bin_continuous(), 43, 44, 46–48, 65
- scr_calibrate, 13
- scr_calibrate(), 5, 6, 58, 59, 71, 74, 75, 79, 81, 83
- scr_capital, 15
- scr_capital(), 10, 41, 50, 51, 53, 63, 81, 92, 101
- scr_classing_accept, 18
- scr_classing_accept(), 19–25, 33
- scr_classing_apply, 19
- scr_classing_apply(), 18, 20, 22–25, 33, 98
- scr_classing_choose, 20
- scr_classing_choose(), 18, 19, 22–25, 33, 57
- scr_classing_discard
 - (scr_classing_accept), 18
- scr_classing_discard(), 24
- scr_classing_import
 - (scr_classing_spec), 22
- scr_classing_import(), 22
- scr_classing_propose, 21
- scr_classing_propose(), 18–20, 23–25, 33
- scr_classing_read (scr_classing_spec), 22
- scr_classing_read(), 22
- scr_classing_spec, 22
- scr_classing_spec(), 18–20, 22, 24, 25, 33
- scr_classing_view, 23
- scr_classing_view(), 18–20, 22–25, 33
- scr_coarse_classing, 24
- scr_coarse_classing(), 18–24, 33, 53
- scr_compare, 26
- scr_compare(), 31, 90, 91

- `scr_config`, 27
- `scr_config()`, 11, 16, 29, 34, 36, 43, 45, 50, 65, 76, 78, 84, 89, 93, 97, 99, 104–106
- `scr_config_keys`, 29
- `scr_config_keys()`, 27, 28, 84, 105
- `scr_connect`, 30
- `scr_connect()`, 55, 89
- `scr_core`, 31
- `scr_core()`, 26, 90, 91
- `scr_cutoff`, 31
- `scr_cutoff()`, 8, 11, 76, 88, 94, 97, 100, 103, 104
- `scr_decisions`, 32
- `scr_decisions()`, 18–20, 22–25
- `scr_default`, 33
- `scr_default()`, 35, 36, 40, 61
- `scr_default_rate`, 35
- `scr_default_rate()`, 14, 34, 40, 59, 61, 82
- `scr_demo`, 36, 38–43
- `scr_demo_ead`, 37, 37, 39–43
- `scr_demo_lgd`, 37, 38, 38, 39–43
- `scr_demo_lgd_cashflows`, 37–39, 39, 41–43
- `scr_demo_panel`, 37–40, 40, 42, 43
- `scr_demo_portfolio`, 37–41, 41, 43
- `scr_demo_rates`, 37–42, 42
- `scr_ead`, 43
- `scr_ead()`, 13, 37, 46–48, 53, 101
- `scr_ead_data`, 44
- `scr_ead_data()`, 13, 37, 43, 44, 47, 48
- `scr_ead_downturn`, 46
- `scr_ead_downturn()`, 13, 44, 46, 48
- `scr_ead_validate`, 47
- `scr_ead_validate()`, 13, 37, 44, 46, 47, 54
- `scr_ecl`, 49
- `scr_ecl()`, 17, 41, 51, 63, 81, 92
- `scr_el`, 51
- `scr_el()`, 17, 50, 63, 81, 92
- `scr_elbe`, 52
- `scr_elbe()`, 54, 66–70, 107
- `scr_export`, 53
- `scr_export()`, 4, 10, 17, 19, 24, 27, 77–79, 86, 97, 102
- `scr_fetch`, 55
- `scr_fetch()`, 30
- `scr_funnel`, 56
- `scr_funnel()`, 57, 64, 88, 89, 95–98
- `scr_gains`, 57
- `scr_gains()`, 57, 64, 88, 89, 95–98
- `scr_grades`, 58
- `scr_grades()`, 5, 6, 15, 59, 71, 74, 75, 79, 81, 83
- `scr_irb_params`, 60
- `scr_irb_params()`, 16, 34, 36, 43, 62, 68, 79, 92
- `scr_irb_rw`, 61
- `scr_irb_rw()`, 15–17, 50, 51, 81, 92
- `scr_iv`, 63
- `scr_iv()`, 73, 85
- `scr_leakage`, 64
- `scr_leakage()`, 57, 89, 95–98
- `scr_lgd`, 65
- `scr_lgd()`, 38, 52, 53, 67–70, 101, 107
- `scr_lgd_downturn`, 66
- `scr_lgd_downturn()`, 52, 65, 66, 68–70, 107
- `scr_lgd_floor`, 67
- `scr_lgd_floor()`, 52, 65–67, 69, 70, 107
- `scr_lgd_pools`, 68
- `scr_lgd_pools()`, 52, 65–68, 70, 107
- `scr_lgd_validate`, 69
- `scr_lgd_validate()`, 52, 54, 66–69, 107
- `scr_master_scale`, 71
- `scr_master_scale()`, 5, 6, 15, 58, 59, 74, 75, 79, 81, 83
- `scr_metrics`, 72
- `scr_metrics()`, 64, 83, 85
- `scr_migration`, 73
- `scr_migration()`, 5, 6, 15, 59, 71, 75, 79, 81, 83
- `scr_moc`, 74
- `scr_moc()`, 5, 6, 15, 59, 71, 74, 79, 81, 83
- `scr_model`, 76
- `scr_model()`, 8, 11, 32, 88, 94, 96, 97, 100, 103, 104
- `scr_monitor`, 77
- `scr_monitor()`, 4, 10, 54, 78, 86, 102
- `scr_monitoring_plan`, 78
- `scr_monitoring_plan()`, 4, 10, 54, 77, 86, 94, 102
- `scr_pd`, 79
- `scr_pd()`, 5, 6, 15, 53, 59, 71, 74, 75, 81–83, 101
- `scr_pd_pit_ttc`, 80
- `scr_pd_pit_ttc()`, 5, 6, 15, 59, 71, 74, 75, 79, 81, 83
- `scr_pd_stress`, 81

`scr_pd_stress()`, 17, 50, 51, 63, 81, 92
`scr_pd_validate`, 82
`scr_pd_validate()`, 5, 6, 15, 54, 59, 71, 74, 75, 79, 81
`scr_presets`, 84
`scr_presets()`, 27–29, 105
`scr_psi`, 84
`scr_psi()`, 48, 64, 73, 83
`scr_reasons`, 86
`scr_reasons()`, 4, 10, 54, 77, 78, 102
`scr_reject`, 87
`scr_reject()`, 8, 11, 32, 76, 94, 97, 100, 103, 104
`scr_result`, 57, 64, 88, 95, 96, 98
`scr_run`, 89
`scr_run()`, 26, 31, 90, 91, 97
`scr_runset`, 26, 31, 90, 90
`scr_sa_rw`, 91
`scr_sa_rw()`, 17, 50, 51, 63, 81
`scr_score_gains`, 95
`scr_score_gains()`, 57, 64, 89, 96, 98
`scr_score_metrics`, 95
`scr_score_metrics()`, 57, 64, 89, 95, 98
`scr_scorecard`, 93
`scr_scorecard()`, 7–9, 11, 14, 19, 24, 32, 53, 58, 76–78, 86–88, 95–98, 100, 101, 103, 104
`scr_select`, 96
`scr_select()`, 8, 9, 11, 24, 25, 28, 32, 53, 56, 57, 64, 76, 88–90, 93, 94, 98–101, 103, 104
`scr_selected`, 98
`scr_selected()`, 57, 64, 88, 89, 93, 95–97
`scr_split`, 99
`scr_split()`, 8, 11, 27, 28, 32, 76, 88, 94, 96, 97, 103, 104
`scr_sql`, 100
`scr_sql()`, 4, 10, 12, 24, 54, 77, 78, 86, 94, 97, 98
`scr_strategy`, 103
`scr_strategy()`, 8, 11, 32, 76, 88, 94, 97, 100, 104
`scr_triage`, 104
`scr_triage()`, 8, 11, 13, 32, 76, 88, 94, 96, 97, 100, 103
`scr_verbose`, 105
`scr_verbose()`, 28, 29, 55, 84
`scr_workout`, 106
`scr_workout()`, 38, 42, 52, 65–70
`summary()`, 97
`summary.scr_result (scr_result)`, 88
`suppressMessages()`, 105
`tabulate()`, 63