# Package: rnaSentry (via r-universe)

November 25, 2026

**Type** Package

**Title** Guarded and Auditable Discovery of Prognostic RNA-Seq Signatures

**Version** 1.0.0

**Description** Orchestrates the statistical pipeline underlying prognostic biomarker discovery from bulk RNA-seq SummarizedExperiment objects: sample-identity and quality-control auditing, confounder and design-formula detection prior to differential expression, signature construction with explicit cross-validation stability reporting, survival modeling (Kaplan-Meier, log-rank, Cox proportional hazards), and external validation against public cohorts. Every automated statistical decision is reported with its supporting statistic and a plain-language rationale, and validation stages are gated rather than silently degraded when guardrail criteria are not met. The package records all guardrail decisions, flag ledgers, and validation results in a structured audit trail that can be rendered into a standalone HTML report.

**License** MIT + file LICENSE

**Encoding** UTF-8

**LazyData** false

**RoxygenNote** 7.3.3

**Depends** R (>= 4.5.0)

**Imports** S4Vectors, SummarizedExperiment, survival, withr, methods

**Suggests** Biobase, BiocParallel, BiocStyle, GEOquery, ggplot2, knitr, rmarkdown, testthat (>= 3.0.0)

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**biocViews** Software, GeneExpression, RNASeq, DifferentialExpression, Survival, QualityControl, BatchEffect, Normalization

**URL** https://ghrieb.github.io/rnaSentry/, https://github.com/Ghrieb/rnaSentry

**BugReports** https://github.com/Ghrieb/rnaSentry/issues

**Config/pak/sysreqs** zlib1g-dev

**Repository** Bioconductor 3.24

**Date/Publication** 2026-11-23 22:57:01 UTC

**RemoteUrl** https://github.com/bioc/rnaSentry

**RemoteRef** RELEASE_3_24

**RemoteSha** 2f473469ff40cea599b2f13831cc14f5939fea4f

# Contents

build_signature . . . . . 2
cox_model . . . . . 6
design_audit . . . . . 8
fetch_gse20685 . . . . . 10
generate_report . . . . . 11
km_curve . . . . . 12
load_counts . . . . . 14
lock_signature . . . . . 15
pca_audit . . . . . 16
plot_pca_audit . . . . . 17
qc_explore . . . . . 18
run_rnaSentry . . . . . 19
sex_check . . . . . 22
survival_parametric . . . . . 23
validate_external . . . . . 25

**Index** **28**

---

`build_signature` *Build a prognostic gene signature from a survival cohort*

---

## Description

`build_signature()` is the core signature-construction stage of the rnaSentry pipeline. From a `SummarizedExperiment` with per-sample survival metadata it:

1. *Screens* every variable gene with a univariate Cox proportional-hazards model against the event indicator, recording the hazard ratio, Wald p-value and a Benjamini-Hochberg adjusted p-value.
2. *Selects* the signature genes by the most significant univariate p-values (`method = "top_n"`) or by an adjusted p-value threshold (`method = "p_value"`).
3. *Estimates joint coefficients* by refitting a single multivariate Cox model on the selected genes. Genes whose coefficient is not finite (rank deficiency) are dropped and the model refit; the signature is never silently degraded, the drop is reported as a flag.

4. *Reports stability* with repeated, event-stratified k-fold cross-validation: within each fold the coefficients are re-estimated on the training samples and the concordance index of the resulting risk score is evaluated on the held-out samples. The mean and standard deviation of the per-fold concordance are returned as `cv_summary`. Concordance is Harrell's C in the risk-score convention: a higher risk score is concordant with an earlier event.

## Usage

```
build_signature(
  se,
  time_col,
  event_col,
  outcome_col = "overall_survival",
  method = c("top_n", "p_value"),
  top_n = 20,
  p_threshold = 0.05,
  repeats = 5,
  folds = 5,
  seed = NULL,
  design_terms = character(0),
  adjust_for_design = TRUE,
  min_events_per_parameter = 5,
  BPPARAM = NULL
)
```

## Arguments

| | |
|---|---|
| `se` | A `SummarizedExperiment` with a count or normalized expression assay and survival metadata. |
| `time_col` | Character. Column of `colData(se)` with follow-up time (numeric, non-negative). |
| `event_col` | Character. Column of `colData(se)` with the event indicator (0/1 or logical). |
| `outcome_col` | Character. A display label for the outcome, e.g. `"overall_survival"`. This is recorded in the signature and used in report titles; it does *not* need to name a column of `colData(se)`. |
| `method` | Character. Selection rule, one of `"top_n"` (default; take the `top_n` genes with the most significant univariate p-value) or `"p_value"` (take every gene with adjusted p-value below `p_threshold`). |
| `top_n` | Integer. Number of genes to keep when `method = "top_n"`. Defaults to `20`. |
| `p_threshold` | Numeric in (0, 1). Adjusted p-value cutoff when `method = "p_value"`. Defaults to `0.05`. |
| `repeats` | Integer. Number of cross-validation repeats. Defaults to `5`. |
| `folds` | Integer. Number of folds per repeat. Defaults to `5`. |
| `seed` | Optional integer. Seeds the cross-validation fold shuffling for reproducible results. Seeding is scoped with `withr::with_seed()`, so the caller's global random-number generator state is left unchanged. |

`design_terms` Character vector of `colData(se)` columns to adjust the univariate screening by. When non-empty and `adjust_for_design = TRUE`, each gene is tested with `Surv ~ gene + design_terms`, so that genes whose survival association is driven entirely by a confounder (for example batch) are not selected. The final signature weights (the joint Cox coefficients) are deliberately estimated from a gene-only model so that the risk score remains portable to cohorts that do not record the design covariates; adjusted inference on the chosen signature is provided by `cox_model()`. Defaults to `character(0)`.

`adjust_for_design`
Logical. When `TRUE` (default) and `design_terms` is non-empty, univariate screening is adjusted for the design terms as described above. Setting it to `FALSE` runs unadjusted screening and raises an `unadjusted_screening` warning, because the pipeline's confounder guardrails are then bypassed.

`min_events_per_parameter`
Numeric. Minimum acceptable number of events per signature gene. When the cohort has fewer events per final signature gene, the signature is flagged with an `events_per_parameter` warning, because cross-validated concordance and hazard ratios are unstable at very low event counts. The default of `5` sits below the classic 10 events-per-variable rule of thumb (Peduzzi et al., *J Clin Epidemiol* 1996) but at the lower end of the 5-9 events-per-variable range that Vittinghoff and McCulloch (*Am J Epidemiol* 2007) argued can be adequate in some settings; it is a defensible middle choice, not a guarantee of estimability. Raise it to `10` for the stricter convention, or lower it to suppress the warning on small but well-behaved cohorts.

`BPPARAM` Optional `BiocParallelParam` from the **BiocParallel** package (for example `BiocParallel::SnowParam(`
When `NULL` (the default) the repeated cross-validation is evaluated serially. When a non-`NULL` backend is supplied, the per-fold evaluations are distributed with `BiocParallel::bplapply()`; **BiocParallel** is only a suggested package and is loaded on demand. Parallelism is strictly opt-in and is bit-identical to the serial path: all randomness is confined to the fold assignment (seeded with `seed`), and the per-fold Cox fits and concordance scores use no random numbers, so the returned signature, cross-validation table, and flag ledger are unchanged regardless of backend.

## Details

**Reading the cross-validated concordance (important).** The univariate screening that selects the signature runs on the full cohort *before* the cross-validation split, so the CV fold is *not* fully untouched by the selection step. On large candidate panels the resulting concordance is therefore *optimistic* (screening-internal): even data with permuted survival yields a non-trivial null concordance (e.g. `~0.8` on a ~21k-gene panel, driven by winner's-curse selection), far above the 0.5 a fully out-of-sample estimate would give. Treat `cv_summary` as a screening-internal stability metric and interpret it *relative* to a matched null — a permutation-null or random-gene control — rather than against 0.5 in absolute terms. The only fully independent out-of-sample estimate is `validate_external()` on a separate cohort.

The return value has class `"rnaSentry_signature"` and is the input expected by the survival modeling and validation stages (`km_curve()`, `cox_model()`, `survival_parametric()`, `validate_external()`)

and by `lock_signature()`. The signature is returned *unlocked*; downstream stages that must not be rerun after publication can be protected with `lock_signature()`.

## Value

An object of class `"rnaSentry_signature"` (a list) with elements:

**genes** Character vector of the final signature genes.

**outcome** The `outcome_col` display label.

**time_col, event_col, design_terms** The metadata columns used.

**screening_terms** Character vector of the design terms actually used to adjust univariate screening (empty when screening was unadjusted).

**cox_stats** Data.frame with one row per screened gene (columns `gene`, `HR`, `p`, `adj_p`).

**coefficients** Named numeric vector of joint Cox coefficients for the signature genes.

**cv_results** Data.frame with columns `repeat_id`, `fold` and `c_index` (concordance on held-out samples).

**cv_summary** Named numeric vector `c(mean, sd)` of the per-fold concordance.

**selection** List with `method` and `n_genes`.

**locked** Always `FALSE`; set by `lock_signature()`.

**created, lock_time** Provenance timestamps.

**assay_used, n_screened, n_filtered** Analysis details.

**flags** Data.frame summarizing every issue raised, with columns `check`, `severity`, `detail` and `stage`.

## Assumptions and limitations

The signature is a linear Cox risk score (larger score = earlier event). The pipeline assumes standard right-censored survival from bulk RNA-seq with raw integer counts or a correctly-named `logcounts` (RNA-seq) or `exprs` (microarray) assay (pre-scaled data in a `counts`-named slot is flagged as `possibly_log_scaled`). It does not model competing risks, time-varying covariates, left truncation, single-cell data, or non-survival end points, and it never corrects batch or design effects. In underpowered cohorts (fewer than `min_events_per_parameter` events per signature gene) the returned concordance and hazard ratios are unstable and an `events_per_parameter` warning is raised; treat such results as hypothesis-generating rather than confirmatory. The discovery stage additionally refuses cohorts with fewer than two events; the downstream survival stages (`km_curve`, `cox_model`, `survival_parametric`, `validate_external`) use a one-event minimum and instead surface instability through their `sparse_events` guardrail.

## Examples

```
library(SummarizedExperiment)
set.seed(5)
counts <- matrix(rpois(400, lambda = 500), nrow = 20, ncol = 20,
                 dimnames = list(paste0("gene", 1:20), paste0("S", 1:20)))
sig_expr <- colMeans(counts[1:5, , drop = FALSE])
risk <- scale(sig_expr)[, 1] * 0.4
```

```
event_time <- rexp(20, rate = 0.03 * exp(0.8 * risk))
censor_time <- rexp(20, rate = 0.02)
time <- pmin(event_time, censor_time)
event <- as.integer(event_time < censor_time)
coldata <- S4Vectors::DataFrame(time = time, event = event,
                                condition = factor(rep(c("A", "B"), 10)),
                                row.names = colnames(counts))
se <- SummarizedExperiment(assays = list(counts = counts), colData = coldata)

sig <- build_signature(se, time_col = "time", event_col = "event",
                       top_n = 5, repeats = 1, folds = 2, seed = 1)
sig
```

---

`cox_model` *Adjusted Cox proportional-hazards model for a signature*

---

## Description

`cox_model()` fits a single multivariate Cox model of the survival outcome on the signature genes together with any design terms recorded in the signature at build time (`sig$design_terms`) and any extra `confounders` supplied here. It is the companion to the gene-only scoring model in `build_signature`: the signature's risk score deliberately ignores the design covariates so it stays portable, while this stage supplies inference *adjusted* for them on a cohort that records them.

## Usage

```
cox_model(sig, se, confounders = character(0))
```

## Arguments

| | |
|---|---|
| `sig` | An object of class `"rnaSentry_signature"` as returned by `build_signature`. |
| `se` | A `SummarizedExperiment` containing the signature genes in its rows and the survival metadata columns recorded in `sig` (`sig$time_col` and `sig$event_col`). Expression is taken from the same analysis assay used at signature-build time (see `build_signature`). |
| `confounders` | Character vector of additional `colData(se)` columns to adjust for, beyond `sig$design_terms`. Defaults to `character(0)`. |

## Details

For every fitted term the model reports the hazard ratio with a Wald confidence interval and p-value. For the signature genes it also reports a Benjamini-Hochberg adjusted p-value across the genes, so that gene-level multiplicity is controlled. Proportional-hazards diagnostics (`cox.zph`) are run on every term and the GLOBAL test; a violation flags the model.

The stage is read-only: it works on both locked and unlocked signatures and records the signature's lock state in its output.

## Value

An object of class "rnaSentry_cox_model" (a list) with elements:

**genes** The signature genes in the model.

**terms** All fitted terms: the genes, then sig$design_terms, then confounders.

**formula** The model formula.

**fit** The coxph fit object.

**coef_table** Data.frame with one row per fitted term: term, coefficient, se, HR, HR_low, HR_high and p.

**gene_summary** Data.frame of the signature-gene rows of coef_table with an added BH-adjusted adj_p.

**zph_summary** Data.frame from cox.zph with one row per term plus GLOBAL; NULL when the test could not be run.

**concordance** Harrell's concordance index of the fitted model.

**ph_violated** Logical; TRUE when any term (including GLOBAL) violates proportional hazards at alpha 0.05.

**n, events** Number of samples and events used by the fit.

**sig_locked** Whether the input signature was locked.

**created** Provenance timestamp.

**flags** Data.frame of issues raised (missing covariate values, non-estimable genes, proportional-hazards violations), with columns check, severity, detail and stage.

## Assumptions and limitations

The model assumes proportional hazards (tested and flagged via cox.zph) and standard right-censored survival. It does not model competing risks, time-varying covariates, or left truncation. Like the rest of the pipeline it makes no batch-effect correction; adjustment for recorded design covariates is supplied through sig$design_terms and confounders.

## Examples

```
library(SummarizedExperiment)
set.seed(7)
counts <- matrix(rpois(450, lambda = 500), nrow = 30, ncol = 15,
                 dimnames = list(paste0("gene", 1:30), paste0("S", 1:15)))
sig_expr <- colMeans(counts[1:4, , drop = FALSE])
risk <- scale(sig_expr)[, 1] * 0.4
event_time <- rexp(15, rate = 0.03 * exp(0.8 * risk))
censor_time <- rexp(15, rate = 0.02)
time <- pmin(event_time, censor_time)
event <- as.integer(event_time < censor_time)
coldata <- S4Vectors::DataFrame(time = time, event = event,
                                batch = factor(rep(c("B1", "B2"),
                                                   length.out = 15)),
                                row.names = colnames(counts))
se <- SummarizedExperiment(assays = list(counts = counts), colData = coldata)
```

```
sig <- build_signature(se, "time", "event", top_n = 4, repeats = 1,
                       folds = 2, seed = 1, design_terms = "batch")
cm <- cox_model(sig, se)
cm
plot(cm)
```

---

`design_audit` *Audit candidate design covariates before differential-expression modeling*

---

## Description

Given a `SummarizedExperiment`, `design_audit()` examines a set of candidate design variables (technical or biological covariates such as batch, sequencing lane or tissue) and recommends a design formula for downstream differential-expression analysis. It does this without fitting per-gene models: the leading principal component of the expression data serves as a surrogate for global expression structure, and every candidate variable is tested for association with that surrogate. A variable that tracks the surrogate significantly is flagged as a potential confounder, because omitting it from the design risks attributing expression structure to the variable of interest.

## Usage

```
design_audit(
  se,
  design_vars,
  outcome_col = NULL,
  surrogate_pc = 1,
  alpha = 0.05,
  interaction_alpha = alpha,
  redundant_effect_size = 0.8
)
```

## Arguments

| | |
|---|---|
| `se` | A `SummarizedExperiment` with a count or normalized expression assay. |
| `design_vars` | Character vector naming candidate design variables, all of which must be columns of `colData(se)`. |
| `outcome_col` | Optional character. Name of the biological variable of interest (for example condition), also a column of `colData(se)`. When supplied it is placed first in the recommended formula, is never dropped as redundant, and is included in the pairwise redundancy scan. Defaults to `NULL`. |
| `surrogate_pc` | Integer. Which principal component to use as the surrogate for global expression structure. Defaults to `1` (the leading PC); pass a different integer to override. |
| `alpha` | Numeric in (0, 1). Significance threshold for the confounder and pairwise association tests. Defaults to `0.05`. |

`interaction_alpha`
: Numeric in (0, 1). Significance threshold for the interaction likelihood-ratio test. Defaults to `alpha`.

`redundant_effect_size`
: Numeric in (0, 1). Minimum effect size (Cramer's V, absolute Pearson r or eta-squared) for a significant pairwise association to be declared redundant. Defaults to `0.8`.

## Details

Three passes are performed:

1. *Confounder scan*: each candidate variable is tested against the surrogate expression gradient (`surrogate_pc`). Categorical variables are tested with one-way ANOVA (or Kruskal-Wallis when the scores are not approximately normal) and numeric variables with linear regression; effect sizes are reported as eta-squared, epsilon-squared or R-squared.
2. *Redundancy scan*: every pair of variables (including `outcome_col` when supplied) is cross-tested with Cramer's V (categorical-categorical), Pearson correlation (numeric-numeric) or a one-way test (mixed), and pairs whose association is significant and whose effect size reaches `redundant_effect_size` are declared redundant.
3. *Interaction scan*: a likelihood-ratio test compares `other_pc ~ surrogate + v` with `other_pc ~ surrogate * v` for each candidate variable, where `other_pc` is the first PC that is not the surrogate. A significant interaction means the variable's influence on expression structure changes along the surrogate gradient; this is reported as interaction-type confounding rather than silently folded into the recommended formula.

The recommended `formula` is `~ outcome_col + design_variables`, with variables declared redundant against a later-listed variable dropped. Every statistical decision is returned alongside a plain-language `rationale`; nothing is dropped or included silently.

## Value

An object of class `"rnaSentry_design"` (a list) with elements:

**formula** The recommended design formula as a `stats::formula` object.

**formula_text** Character version of `formula`.

**rationale** Character vector of plain-language statements justifying every automated decision.

**confounder_table** Data.frame with one row per candidate variable (columns `variable`, `type`, `n`, `test`, `statistic`, `df`, `p_value`, `adj_p`, `effect_size`, `effect_size_type`, `flagged`). The `p_value` column holds the raw association p-value; `adj_p` is the Benjamini-Hochberg adjusted p-value across all tested candidate variables, and `flagged` is based on `adj_p`.

**interaction_tested** Logical; whether the interaction LRT could be performed (it requires at least two PCs and enough samples).

**interaction_table** Data.frame with one row per candidate variable (columns `variable`, `test`, `statistic`, `df`, `p_value`, `flagged`), or `NULL` when not tested.

**pairwise_table** Data.frame with one row per variable pair (columns `var1`, `var2`, `type_pair`, `test`, `statistic`, `df`, `p_value`, `effect_size`, `effect_size_type`, `redundant`), or `NULL` when fewer than two variables are available.

**dropped_vars** Character vector of design variables removed from the recommended formula as redundant.

**surrogate_pc** The surrogate PC actually used (after clamping to the available number of PCs).

**alpha, interaction_alpha, redundant_effect_size** The thresholds used.

**flags** Data.frame summarizing every issue raised, with columns `check`, `severity`, `detail` and `stage`.

## Examples

```
library(SummarizedExperiment)
set.seed(4)
counts <- matrix(rpois(360, lambda = 200), nrow = 30, ncol = 12,
                 dimnames = list(paste0("gene", 1:30), paste0("S", 1:12)))
batch <- factor(rep(c("B1", "B2"), each = 6))
condition <- factor(rep(c("A", "B"), times = 6))
counts[1:10, batch == "B2"] <- counts[1:10, batch == "B2"] * 3L
coldata <- S4Vectors::DataFrame(batch = batch, condition = condition,
                                age = runif(12, 40, 80),
                                row.names = paste0("S", 1:12))
se <- SummarizedExperiment(assays = list(counts = counts), colData = coldata)

design_audit(se, design_vars = c("batch", "age"), outcome_col = "condition")
```

---

# `fetch_gse20685` *Fetch and process GSE20685 breast cancer cohort*

---

## Description

Downloads GSE20685 (Li *et al.*, 2010) from GEO via `GEOquery::getGEO()` (which caches the series matrix itself), performs probe-to-gene collapse (largest mean expression per symbol), subsets to the top 3000 most variable genes, and returns a `SummarizedExperiment` with an `exprs` assay (microarray log2 intensities, used as-is by the pipeline) and clinical metadata (`time`, `event`, `age`, `subtype`).

## Usage

```
fetch_gse20685()
```

## Details

This helper packages the processing pipeline from `inst/scripts/repro_gse20685.R` for use in vignettes. The full series matrix is downloaded from GEO, probes are collapsed to gene symbols by largest mean expression, and the top 3000 most variable genes are retained. Clinical metadata includes overall-survival follow-up time (years), event indicator (1 = death), age at diagnosis, and breast-cancer subtype.

### Value

A SummarizedExperiment or NULL if the download or processing fails (e.g. network unavailable, GEOquery not installed).

### Examples

```
se <- fetch_gse20685()
se
```

---

## generate_report *Generate the rnaSentry analysis report*

---

### Description

generate_report() assembles the results of any collection of rnaSentry stages into a single self-contained HTML report. It renders the plain report_template.Rmd shipped with the package (via render), which prints a pipeline summary, a unified audit-trail table collecting every flag from every supplied stage (all rnaSentry stage results carry a flags data.frame with check/severity/detail/stage columns), and one section per stage using its print method.

### Usage

```
generate_report(
  stages,
  output_file = "rnaSentry_report.html",
  output_dir = ".",
  quiet = TRUE
)
```

### Arguments

| | |
|---|---|
| stages | A named list of rnaSentry stage results (for example elements of the stages list returned by run_rnaSentry or a hand-assembled set of audit, signature, survival-modeling and validation outputs). Names are used as report section titles. |
| output_file | Character. File name (without directory) of the rendered report. Defaults to "rnaSentry_report.html". |
| output_dir | Character. Directory to write the report into. Must exist. Defaults to the working directory. |
| quiet | Logical. Passed to rmarkdown::render(); suppresses the rendering console output when TRUE (default). |

### Value

The path of the rendered report, invisibly.

## Examples

```
library(SummarizedExperiment)
set.seed(9)
counts <- matrix(rpois(400, lambda = 500), nrow = 20, ncol = 20,
                 dimnames = list(paste0("gene", 1:20), paste0("S", 1:20)))
sig_expr <- colMeans(counts[1:5, , drop = FALSE])
risk <- scale(sig_expr)[, 1] * 0.4
event_time <- rexp(20, rate = 0.03 * exp(0.8 * risk))
censor_time <- rexp(20, rate = 0.02)
time <- pmin(event_time, censor_time)
event <- as.integer(event_time < censor_time)
coldata <- S4Vectors::DataFrame(time = time, event = event,
                                row.names = colnames(counts))
se <- SummarizedExperiment(assays = list(counts = counts), colData = coldata)

sig <- build_signature(se, "time", "event", top_n = 5, repeats = 1,
                       folds = 2, seed = 1)
stages <- list(
  km_curve = km_curve(sig, se),
  cox_model = cox_model(sig, se)
)
generate_report(stages, output_dir = tempdir())
```

---

`km_curve` *Kaplan-Meier survival analysis of a signature's risk groups*

---

## Description

Scores each sample of a `SummarizedExperiment` with the signature's joint Cox coefficients and splits the cohort into `"low"` and `"high"` risk groups. By default the split is at the median risk score of the supplied cohort; pass `cutpoint` to re-apply a cutpoint established elsewhere (for example the discovery-cohort median recorded in a previous `km_curve()` call), so risk groups are comparable across cohorts. Fits a Kaplan-Meier survival curve for each group with a log-rank test. The analysis is read-only: it works on both locked and unlocked signatures and records the signature's lock state in its output.

## Usage

```
km_curve(sig, se, cutpoint = NULL)
```

## Arguments

| | |
|---|---|
| `sig` | An object of class `"rnaSentry_signature"` as returned by `build_signature`. |
| `se` | A `SummarizedExperiment` containing the signature genes in its rows and the survival metadata columns recorded in `sig` (`sig$time_col` and `sig$event_col`). Expression is taken from the same analysis assay used at signature-build time (see `build_signature`). |

`cutpoint` Numeric risk-score threshold: samples with score >= `cutpoint` are `"high"` and the rest `"low"`. When `NULL` (default) the split is at the median risk score of the supplied cohort.

## Value

An object of class `"rnaSentry_km"` (a list) with elements:

**genes** The signature genes scored.

**score** Per-sample risk score (linear predictor).

**groups** Factor with levels `"low"` and `"high"` from the risk-score split.

**cutpoint** The risk-score cutpoint applied.

**cutpoint_type** `"median"` when the cohort median was used, `"custom"` when `cutpoint` was supplied.

**fit** The `survfit` object with one curve per group.

**log_rank_p** p-value from the log-rank test.

**median_survival** Named vector of median survival per group.

**events_low, events_high** Event counts per group.

**sig_locked** Whether the input signature was locked.

**created** Provenance timestamp.

**flags** Data.frame of issues raised (e.g. sparse events per group), with columns `check`, `severity`, `detail` and `stage`.

## Examples

```
library(SummarizedExperiment)
set.seed(5)
counts <- matrix(rpois(120, lambda = 500), nrow = 12, ncol = 10,
                 dimnames = list(paste0("gene", 1:12), paste0("S", 1:10)))
sig_expr <- colMeans(counts[1:3, , drop = FALSE])
risk <- scale(sig_expr)[, 1] * 0.4
event_time <- rexp(10, rate = 0.03 * exp(0.8 * risk))
censor_time <- rexp(10, rate = 0.02)
time <- pmin(event_time, censor_time)
event <- as.integer(event_time < censor_time)
coldata <- S4Vectors::DataFrame(time = time, event = event,
                                row.names = colnames(counts))
se <- SummarizedExperiment(assays = list(counts = counts), colData = coldata)
sig <- build_signature(se, time_col = "time", event_col = "event",
                       top_n = 3, repeats = 1, folds = 2, seed = 1)
km <- km_curve(sig, se)
km
```

load_counts *Validate and construct a SummarizedExperiment from count data*

## Description

Thin validation wrapper around SummarizedExperiment::SummarizedExperiment() that catches the most common input mistakes before they propagate into the pipeline: duplicate or missing sample IDs, non-integer counts, empty rows or columns, and non-syntactic gene identifiers.

## Usage

```
load_counts(counts, colData, assay_name = "counts", check_gene_ids = TRUE)
```

## Arguments

| | |
|---|---|
| counts | Integer or numeric matrix of gene expression values (genes in rows, samples in columns). Row names must be gene identifiers; column names must be sample identifiers. |
| colData | A data.frame or S4Vectors::DataFrame of sample-level metadata. Row names must match the column names of counts. When omitted, an empty DataFrame is created. |
| assay_name | Character. Name for the assay slot. Defaults to "counts". |
| check_gene_ids | Logical. When TRUE (default) the function warns if gene identifiers look like Ensembl accessions (starting with "ENS") or contain non-syntactic characters, because downstream stages rely on gene-symbol rownames. |

## Value

A SummarizedExperiment with one assay named assay_name. Any issues found are recorded in a flag ledger accessible via S4Vectors::metadata(se)$flags and surfaced via warning().

## Assumptions and limitations

This function validates structure; it does not normalise, filter, or transform the count matrix. Gene identifiers are checked but not converted. Sample-metadata columns used later by the pipeline (e.g. time_col, event_col, batch_col) are not validated here — that happens in the stage that first uses them.

## Examples

```
counts <- matrix(1:12, nrow = 3,
                 dimnames = list(c("TP53", "BRCA1", "MYC"),
                                 c("S1", "S2", "S3", "S4")))
coldata <- data.frame(
  time = c(10, 12, 8, 15),
  event = c(1, 0, 1, 0),
  row.names = colnames(counts))
```

```
se <- load_counts(counts, coldata)
se
```

---

`lock_signature` *Lock or unlock an rnaSentry signature*

---

## Description

Toggles the reproducibility guardrail on an `rnaSentry_signature` object. Locking records a timestamp and an audit entry in the signature's `flags` table; downstream stages that must not be rerun after publication can then be protected. The function is copy-on-write: the input object is never modified and the updated signature is returned.

## Usage

```
lock_signature(sig, lock = TRUE)
```

## Arguments

| | |
|---|---|
| `sig` | An object of class `"rnaSentry_signature"`, as returned by `build_signature`. |
| `lock` | Logical. `TRUE` (default) locks the signature, `FALSE` unlocks it. |

## Value

A new `rnaSentry_signature` object with updated `locked`, `lock_time` and `flags` fields.

## Enforceable lock

Locking stores a gene-set fingerprint in a session-level environment. While this fingerprint is set, `run_rnaSentry()` will refuse to run, preventing silent re-selection of genes after survival analysis. The lock can be reversed by calling `lock_signature(sig, lock = FALSE)`, but doing so after survival stages have been run is not recommended.

## Examples

```
sig <- structure(list(genes = "gene1", outcome = "overall_survival",
                      locked = FALSE),
                 class = "rnaSentry_signature")
locked <- lock_signature(sig)
locked$locked
locked$lock_time
unlocked <- lock_signature(locked, lock = FALSE)
unlocked$locked
```

pca_audit *PCA audit of expression data with optional batch association testing*

## Description

Runs principal component analysis on a variance-stabilized or log-transformed version of the count assay and reports how much variance each principal component (PC) explains. When `batch_col` is supplied, each PC score is tested against the batch variable and any PC where batch explains a significant share of variance is flagged. This feeds `design_audit()` by surfacing which batch/technical variables are associated with global expression structure.

## Usage

```
pca_audit(
  se,
  batch_col = NULL,
  top_n_pcs = 5,
  scale = TRUE,
  batch_alpha = 0.05
)
```

## Arguments

| | |
|---|---|
| `se` | A `SummarizedExperiment` with a count or normalized expression assay. |
| `batch_col` | Optional character. Name of a `colData(se)` column (categorical or numeric) to test against each PC score. Defaults to `NULL` (no batch testing). |
| `top_n_pcs` | Integer. Number of leading PCs to include in `scores` and to test against `batch_col`. Defaults to `5`. |
| `scale` | Logical. Whether to scale each gene to unit variance before PCA (`scale. = TRUE`). Defaults to `TRUE` so that genes are comparable on the log-expression scale. |
| `batch_alpha` | Numeric in (0, 1). Significance threshold used to flag a PC whose score associates with `batch_col`. Defaults to `0.05`. |

## Details

The analysis matrix is chosen by preference from an existing assay named `"logcounts"` or `"vst"` (RNA-seq) or `"exprs"` (microarray); if none exists, `log2(counts + 1)` is computed from the first assay. Genes with zero variance or missing values are removed because they carry no information for PCA and would make `stats::prcomp` fail; the number removed and the reason are reported in the result (nothing is dropped silently).

Unlike `design_audit`, no multiple-testing correction is applied to the per-PC batch tests: `batch_alpha` is applied to each test at its nominal level. This is a deliberate choice, because at most `top_n_pcs` (default 5) PCs are tested against a single batch variable, so the correction would be a near no-op; the raw p-values and a Benjamini-Hochberg adjusted p-value (`adj_p`) are both reported in `batch_tests` so users can apply their own threshold. Candidate confounder screening in `design_audit`, which tests many variables at once, is BH-corrected instead.

## Value

An object of class "rnaSentry_pca" (a list) with elements:

**pca** The stats::prcomp object.

**percent_variance** Named numeric vector of the percent of variance explained by every PC.

**scores** A data.frame of the first top_n_pcs PC scores, with sample IDs as row names.

**col_data** The colData(se) columns as a data.frame, kept so plot_pca_audit can color points by a metadata column.

**assay_used** Name of the assay used for the analysis.

**n_genes_analyzed** Number of genes retained after filtering.

**n_genes_filtered** Number of genes removed before PCA.

**filtered_reason** Character describing why genes were removed (empty string if none).

**batch_col** The batch column name used, or NULL.

**batch_tests** A data.frame with one row per tested PC (columns pc, test, statistic, df, p_value, adj_p, effect_size, effect_size_type, flagged), or NULL when batch_col is not supplied.

**flags** A data.frame summarizing every issue raised, with columns check, severity, detail and stage.

## Examples

```
library(SummarizedExperiment)
set.seed(1)
counts <- matrix(rpois(240, lambda = 200), nrow = 20, ncol = 12,
                 dimnames = list(paste0("gene", 1:20), paste0("S", 1:12)))
coldata <- DataFrame(condition = rep(c("A", "B"), each = 6),
                     batch = factor(rep(c("B1", "B2"), each = 6)),
                     row.names = paste0("S", 1:12))
se <- SummarizedExperiment(assays = list(counts = counts), colData = coldata)

pca_audit(se, batch_col = "batch")
```

---

# plot_pca_audit *Plot PCA scores from a pca_audit result*

---

## Description

Produces a ggplot2 scatter plot of two principal component score axes, optionally colored by a sample-metadata column. This is a purely visual convenience: the statistical results returned by pca_audit do not depend on this function being called.

## Usage

```
plot_pca_audit(pca_result, pc_x = 1, pc_y = 2, color_by = NULL)
```

### Arguments

| | |
|---|---|
| pca_result | An object of class "rnaSentry_pca" as returned by pca_audit. |
| pc_x, pc_y | Integers selecting the two PC axes to plot. Defaults to 1 and 2. |
| color_by | Optional character. Name of a sample-metadata column in the original colData(se) to color points by. |

### Value

A ggplot object.

### Examples

```
library(SummarizedExperiment)
set.seed(2)
counts <- matrix(rpois(240, lambda = 200), nrow = 20, ncol = 12,
                 dimnames = list(paste0("gene", 1:20), paste0("S", 1:12)))
coldata <- DataFrame(batch = factor(rep(c("B1", "B2"), each = 6)),
                     row.names = paste0("S", 1:12))
se <- SummarizedExperiment(assays = list(counts = counts), colData = coldata)
res <- pca_audit(se, batch_col = "batch")
plot_pca_audit(res)
```

---

## qc_explore *Quality-control audit of a bulk RNA-seq SummarizedExperiment*

---

### Description

Runs a battery of pre-analysis checks on a raw-counts SummarizedExperiment: duplicate sample identifiers, missing values in sample metadata, non-integer count values, and library-size outliers detected via a robust (median absolute deviation-based) rule. Every check is reported with the statistic that triggered it, not just a pass/fail flag, so the result can be inspected or logged rather than trusted blindly.

### Usage

```
qc_explore(se, mad_threshold = 3)
```

### Arguments

| | |
|---|---|
| se | A SummarizedExperiment with a raw-count assay as its first assay (assay(se, 1)). Counts should be non-negative integers (or values coercible to integers without loss). |
| mad_threshold | Numeric scalar. Samples whose log10 library size deviates from the median by more than mad_threshold robust median absolute deviations are flagged as library-size outliers. Defaults to 3. |

## Value

An object of class "rnaSentry_qc" (a list) with elements:

**n_samples** Number of samples (columns) in se.

**n_genes** Number of genes (rows) in se.

**duplicate_samples** Character vector of duplicated sample IDs, or character(0) if none.

**non_integer_counts** Logical; TRUE if the count assay contains non-integer values.

**missing_summary** A data.frame reporting the proportion of missing values per colData(se) column.

**library_size_outliers** Character vector of sample IDs flagged as library-size outliers.

**flags** A data.frame summarizing every issue raised, with columns check, severity, detail and stage.

## Examples

```
library(SummarizedExperiment)
set.seed(1)
counts <- matrix(rpois(60, lambda = 200), nrow = 6, ncol = 10,
                 dimnames = list(paste0("gene", 1:6), paste0("S", 1:10)))
counts[, 1] <- counts[, 1] * 50L  # inject one library-size outlier
coldata <- DataFrame(condition = rep(c("A", "B"), each = 5),
                     row.names = paste0("S", 1:10))
se <- SummarizedExperiment(assays = list(counts = counts), colData = coldata)

qc <- qc_explore(se)
qc$flags
```

---

# run_rnaSentry *Run the complete rnaSentry pipeline*

---

## Description

run_rnaSentry() executes the full rnaSentry pipeline on a single cohort and collects every stage result into one object:

1. design_audit over design_vars (when any are supplied).
2. pca_audit (with batch_col when supplied).
3. build_signature, then lock_signature so the survival-modeling stages operate on the published signature.
4. km_curve, cox_model and survival_parametric on the locked signature.
5. generate_report assembling all stages into an HTML report (when render_report = TRUE).

## Usage

```
run_rnaSentry(
  se,
  time_col,
  event_col,
  outcome_col = "overall_survival",
  design_terms = character(0),
  design_vars = design_terms,
  batch_col = NULL,
  method = c("top_n", "p_value"),
  top_n = 20,
  p_threshold = 0.05,
  repeats = 5,
  folds = 5,
  seed = NULL,
  adjust_for_design = TRUE,
  min_events_per_parameter = 5,
  BPPARAM = NULL,
  report_file = "rnaSentry_report.html",
  report_dir = ".",
  render_report = TRUE
)
```

## Arguments

| | |
|---|---|
| `se` | A `SummarizedExperiment` with a count or normalized expression assay and survival metadata. |
| `time_col` | Character. Column of `colData(se)` with follow-up time. |
| `event_col` | Character. Column of `colData(se)` with the event indicator (0/1 or logical). |
| `outcome_col` | Character. Display label for the outcome. Defaults to `"overall_survival"`. |
| `design_terms` | Character vector of `colData(se)` columns to adjust the signature's univariate screening by. Defaults to `character(0)`. |
| `design_vars` | Character vector of `colData(se)` columns scanned by the `design_audit` confounder scan. Defaults to `design_terms`. |
| `batch_col` | Optional character. `colData(se)` column tested by the `pca_audit` batch scan. Defaults to `NULL`. |
| `method, top_n, p_threshold, repeats, folds, seed, adjust_for_design, min_events_per_parameter, BPPARAM` | Passed to `build_signature`. |
| `report_file` | Character. Output file name for the HTML report. |
| `report_dir` | Character. Directory (which must exist) to write the report into. |
| `render_report` | Logical. When `TRUE` (default) the report is rendered as the final step. |

## Details

The result's `stages` list can be passed directly to `generate_report`, or any stage can be re-run individually (for example `validate_external` against an independent cohort, using the cutpoint recorded by the `km_curve` stage).

## Value

An object of class `"rnaSentry_run"` (a list) with elements:

**stages** Named list of the stage results, with entries `design_audit` (only when `design_vars` was supplied), `pca_audit`, `build_signature`, `km_curve`, `cox_model` and `survival_parametric`.

**report** Path of the rendered HTML report, or `NULL` when `render_report = FALSE`.

## Reproducibility lock

The run locks the discovered signature (see `lock_signature`), recording a session-level fingerprint. While any locked signature exists, `run_rnaSentry()` refuses to run again, preventing silent re-selection of signature genes after survival analysis. To run the pipeline a second time in the same session (for example on another cohort), release the lock first with `lock_signature(sig, lock = FALSE)` on the locked signature. Individual stages remain re-runnable while the lock is set.

## Assumptions and limitations

The pipeline targets bulk RNA-seq with standard right-censored survival and linear Cox risk scores (see the Assumptions sections of `build_signature` and `cox_model`). It detects and reports batch and design confounders but does not correct for them, and it makes no differential-expression calls. On small cohorts an `events_per_parameter` warning from `build_signature` is expected and should be interpreted as a call for caution, not ignored.

## Examples

```
library(SummarizedExperiment)
set.seed(8)
counts <- matrix(rpois(400, lambda = 500), nrow = 20, ncol = 20,
                 dimnames = list(paste0("gene", 1:20), paste0("S", 1:20)))
sig_expr <- colMeans(counts[1:5, , drop = FALSE])
risk <- scale(sig_expr)[, 1] * 0.4
event_time <- rexp(20, rate = 0.03 * exp(0.8 * risk))
censor_time <- rexp(20, rate = 0.02)
time <- pmin(event_time, censor_time)
event <- as.integer(event_time < censor_time)
coldata <- S4Vectors::DataFrame(time = time, event = event,
                                batch = factor(rep(c("B1", "B2"), 10)),
                                row.names = colnames(counts))
se <- SummarizedExperiment(assays = list(counts = counts), colData = coldata)

run <- run_rnaSentry(se, "time", "event", design_vars = "batch",
                     top_n = 5, repeats = 1, folds = 2, seed = 1,
                     report_dir = tempdir())
run
```

sex_check *Verify reported sex against inferred biological sex from expression*

## Description

Compares the sex recorded in sample metadata against sex inferred from expression of *XIST* and a panel of Y-chromosome marker genes. Designed to catch sample swaps and metadata entry errors before they propagate into downstream differential expression or survival analysis.

## Usage

```
sex_check(
  se,
  sex_col = "sex",
  xist_gene = "XIST",
  y_genes = c("RPS4Y1", "DDX3Y", "KDM5D")
)
```

## Arguments

| | |
|---|---|
| se | A SummarizedExperiment with gene symbols as rownames(se) and a numeric (counts or normalized) assay as its first assay. |
| sex_col | Character. Name of the colData(se) column holding reported sex, coded as "M" / "F". Defaults to "sex". |
| xist_gene | Character. Gene symbol for XIST. Defaults to "XIST". |
| y_genes | Character vector of Y-chromosome marker gene symbols. Defaults to c("RPS4Y1", "DDX3Y", "KDM5D"). |

## Details

Inference rule: a sample is called "F" when XIST expression is high relative to the Y-gene panel, "M" when the reverse holds, and "ambiguous" when neither signal is clearly dominant (for example, low-quality or highly degraded samples). The decision uses the sign of each sample's XIST-minus-Y-panel rank score: a score at least one rank unit above zero is called "F", at least one rank unit below zero "M", and a score within one rank unit of zero "ambiguous". Because the rule is based on ranks rather than hard-coded expression cutoffs, it is robust across different normalizations and sequencing depths.

## Value

A data.frame with one row per sample and columns sample_id, reported_sex, inferred_sex, and status (one of "OK", "MISMATCH", "AMBIGUOUS", or "MISSING" when reported sex is absent).

## Examples

```
library(SummarizedExperiment)
genes <- c("XIST", "RPS4Y1", "DDX3Y", "KDM5D", "GAPDH")
samples <- paste0("S", 1:6)
expr <- matrix(0, nrow = length(genes), ncol = length(samples),
               dimnames = list(genes, samples))
# samples 1-3 look female (high XIST, low Y genes)
expr["XIST", 1:3] <- c(950, 900, 875)
expr[c("RPS4Y1", "DDX3Y", "KDM5D"), 1:3] <- 5
# samples 4-6 look male, but S6's metadata will be mislabeled below
expr["XIST", 4:6] <- 5
expr[c("RPS4Y1", "DDX3Y", "KDM5D"), 4:6] <- 400
expr["GAPDH", ] <- 1000

coldata <- DataFrame(sex = c("F", "F", "F", "M", "M", "F"),
                     row.names = samples)  # S6 mislabeled as F
se <- SummarizedExperiment(assays = list(counts = expr), colData = coldata)

sex_check(se)
```

---

`survival_parametric` *Parametric survival models for a signature's risk score*

---

## Description

`survival_parametric()` scores each sample of a `SummarizedExperiment` with the signature's joint Cox coefficients and fits a set of parametric accelerated-failure-time models of the event time on that score, using `survreg`. It returns an AIC comparison across the fitted distributions, the fitted survival curves (evaluated at the mean risk score) for overlay on the Kaplan-Meier estimate, and a `plot()` method that draws the KM curve together with the parametric curves.

## Usage

```
survival_parametric(
  sig,
  se,
  dists = c("weibull", "exponential", "lognormal", "loglogistic")
)
```

## Arguments

| | |
|---|---|
| `sig` | An object of class `"rnaSentry_signature"` as returned by `build_signature`. |
| `se` | A `SummarizedExperiment` containing the signature genes in its rows and the survival metadata columns recorded in `sig` (`sig$time_col` and `sig$event_col`). Expression is taken from the same analysis assay used at signature-build time (see `build_signature`). |
| `dists` | Character vector of parametric distributions to fit, any subset of `"weibull"`, `"exponential"`, `"lognormal"` and `"loglogistic"`. Defaults to all four. |

## Details

The stage is read-only: it works on both locked and unlocked signatures and records the signature's lock state in its output.

## Value

An object of class `"rnaSentry_parametric"` (a list) with elements:

**genes** The signature genes scored.

**score** Per-sample risk score (linear predictor).

**fits** Named list of `survreg` fits, one per distribution.

**table** Data.frame comparing distributions on `loglik`, `npar`, `AIC`, `delta_AIC`, `weight` and `best`, ordered by increasing AIC.

**best** Name of the lowest-AIC distribution.

**km_fit** The `survfit` object of the whole cohort.

**curves** Named list of fitted survival curves (data.frame with `time` and `survival`), evaluated at the mean risk score.

**sig_locked** Whether the input signature was locked.

**created** Provenance timestamp.

**flags** Data.frame of issues raised (e.g. a model that failed to fit, or near-tied AIC values), with columns `check`, `severity`, `detail` and `stage`.

## Examples

```
library(SummarizedExperiment)
set.seed(9)
counts <- matrix(rpois(450, lambda = 500), nrow = 30, ncol = 15,
                 dimnames = list(paste0("gene", 1:30), paste0("S", 1:15)))
sig_expr <- colMeans(counts[1:4, , drop = FALSE])
risk <- scale(sig_expr)[, 1] * 0.4
event_time <- rexp(15, rate = 0.03 * exp(0.8 * risk))
censor_time <- rexp(15, rate = 0.02)
time <- pmin(event_time, censor_time)
event <- as.integer(event_time < censor_time)
coldata <- S4Vectors::DataFrame(time = time, event = event,
                                row.names = colnames(counts))
se <- SummarizedExperiment(assays = list(counts = counts), colData = coldata)
sig <- build_signature(se, "time", "event", top_n = 4, repeats = 1,
                       folds = 2, seed = 1)
sp <- survival_parametric(sig, se)
sp
plot(sp)
```

validate_external *Validate a signature on an external cohort*

## Description

validate_external() applies a locked-down signature to an independent cohort: it scores every external sample with the signature's joint Cox coefficients (the gene-only model, so the score is portable to cohorts that do not record the discovery cohort's design covariates), splits the cohort into "low" and "high" risk groups at the supplied *discovery* cutpoint, and reports the external discriminant validity: a log-rank test between the groups, per-group median survival and event counts, and the concordance index of the continuous risk score against the external survival outcome (Harrell's C in the risk-score convention: a higher risk score is concordant with an earlier event).

## Usage

```
validate_external(
  sig,
  external_se,
  time_col = NULL,
  event_col = NULL,
  cutpoint,
  min_gene_overlap_frac = 1,
  drop_missing = FALSE
)
```

## Arguments

| | |
|---|---|
| sig | An object of class "rnaSentry_signature" as returned by build_signature. |
| external_se | A SummarizedExperiment of an independent cohort, with an expression assay and survival metadata. |
| time_col, event_col | Names of the survival columns in colData(external_se). When NULL (default) the column names recorded in sig are used. |
| cutpoint | The risk-score threshold defining the "high" group (samples with score >= cutpoint). Must be a single finite number; it is never derived from the external cohort. |
| min_gene_overlap_frac | Numeric in (0, 1]. The minimum fraction of signature genes that must be present in the external cohort when drop_missing = TRUE. Defaults to 1. |
| drop_missing | Logical. When TRUE, signature genes missing from the external cohort are dropped (and flagged by name), provided the remaining overlap is at least min_gene_overlap_frac. When FALSE (default) any missing gene is an error. |

## Details

The stage *never* recomputes a within-cohort cutpoint. The `cutpoint` must be passed explicitly (for example the discovery-cohort median recorded in the `cutpoint` element returned by `km_curve`); passing no cutpoint is an error, so external risk groups are always defined by the same threshold that defined the discovery groups.

Missing genes are handled strictly by default: `drop_missing = FALSE` and `min_gene_overlap_frac = 1` (the defaults) require every signature gene to be present, and error otherwise, naming the missing genes. Set `drop_missing = TRUE` (with `min_gene_overlap_frac < 1` as needed) to explicitly allow scoring on a subset of genes; the dropped genes are named in a flag.

The stage is read-only: it works on both locked and unlocked signatures and records the signature's lock state in its output.

## Value

An object of class `"rnaSentry_external"` (a list) with elements:

**genes_used** The signature genes present in the external cohort.

**genes_missing** The signature genes absent from the external cohort (empty when none).

**score** Per-sample external risk score.

**groups** Factor with levels `"low"` and `"high"` from the discovery cutpoint.

**cutpoint, cutpoint_type** The applied cutpoint and `"custom"`.

**km_fit** The `survfit` object with one curve per group.

**log_rank_p** p-value from the log-rank test between groups.

**median_survival** Named vector of median survival per group.

**events_low, events_high** Event counts per group.

**concordance** Concordance index of the continuous risk score against the external survival outcome.

**sig_locked** Whether the input signature was locked.

**created** Provenance timestamp.

**flags** Data.frame of issues raised (dropped genes, sparse events, non-finite concordance), with columns `check`, `severity`, `detail` and `stage`.

## Assumptions and limitations

External risk groups are defined exclusively by the supplied discovery cutpoint; the stage never re-estimates a threshold from the external cohort. The cohort is assumed to be bulk RNA-seq with standard right-censored survival and the same assay convention as the discovery data (raw counts or a correctly-named `logcounts` assay; pre-scaled data in a `counts`-named slot is flagged as `possibly_log_scaled`). Competing risks and other non-standard survival structures are not modeled. Unlike the discovery stage (`build_signature`, which requires at least two events), a one-event minimum applies here; the resulting instability is reported through the `sparse_events` guardrail.

## Examples

```
library(SummarizedExperiment)
set.seed(11)
counts <- matrix(rpois(800, lambda = 500), nrow = 40, ncol = 20,
                 dimnames = list(paste0("gene", 1:40), paste0("S", 1:20)))
sig_expr <- colMeans(counts[1:5, , drop = FALSE])
risk <- scale(sig_expr)[, 1] * 0.4
event_time <- rexp(20, rate = 0.03 * exp(0.8 * risk))
censor_time <- rexp(20, rate = 0.02)
time <- pmin(event_time, censor_time)
event <- as.integer(event_time < censor_time)
coldata <- S4Vectors::DataFrame(time = time, event = event,
                                row.names = colnames(counts))
se <- SummarizedExperiment(assays = list(counts = counts), colData = coldata)
sig <- build_signature(se, "time", "event", top_n = 5, repeats = 1,
                       folds = 2, seed = 1)
cutoff <- km_curve(sig, se)$cutpoint
ext <- se
ext <- ext[, sample(ncol(ext))]
val <- validate_external(sig, ext, cutpoint = cutoff)
val
```

# Index

build_signature, 2, *6*, *12*, *15*, *19–21*, *23*, *25*, *26*

cox.zph, *6*
cox_model, *5*, 6, *19*, *21*

design_audit, 8, *16*, *19*, *20*

fetch_gse20685, 10

generate_report, 11, *19*, *21*

km_curve, *5*, 12, *19*, *26*

load_counts, 14
lock_signature, 15, *19*, *21*

pca_audit, 16, *17–20*
plot_pca_audit, 17, *17*

qc_explore, 18

render, *11*
run_rnaSentry, *11*, 19

sex_check, 22
SummarizedExperiment, *18*
survival_parametric, *5*, *19*, 23
survreg, *23*

validate_external, *4*, *5*, *21*, 25