

Package ‘forestsearch’

October 10, 2026

Title Exploratory Subgroup Identification in Survival and GLM Outcomes

Version 0.4.0

Description Implements statistical methods for exploratory subgroup identification in clinical trials. Provides tools for identifying patient subgroups with differential treatment effects using machine learning approaches including Generalized Random Forests (GRF), LASSO regularization, and exhaustive combinatorial search algorithms. Supports survival endpoints (Cox proportional hazards), binary outcomes (log odds ratio, log relative risk, risk difference), continuous outcomes (mean difference), and count / rate outcomes (log incidence rate ratio via Poisson, quasi-Poisson, or negative-binomial GLMs with optional person-time offset). Features bootstrap bias correction using infinitesimal jackknife methods to address selection bias in post-hoc analyses. Designed for clinical researchers conducting exploratory subgroup analyses in randomized controlled trials, particularly for multi-regional clinical trials (MRCT) requiring regional consistency evaluation. Methods are described in Leon et al. (2024) [doi:10.1002/sim.10163](https://doi.org/10.1002/sim.10163).

Post-selection inference for the identified subgroup follows León and Anderson (2026) [doi:10.48550/arXiv.2609.38361](https://doi.org/10.48550/arXiv.2609.38361).

License MIT + file LICENSE

Depends R (>= 4.1.0)

Encoding UTF-8

VignetteBuilder knitr, quarto

SystemRequirements Quarto command line tools
(<https://github.com/quarto-dev/quarto-cli>)

Imports data.table, doFuture, dplyr, foreach, future, future.apply, future.callr, ggplot2, glmnet (>= 5.0), grf, gt, Matrix, mvtnorm, parallel, patchwork, policytree, progressr, randomForest, rlang, stats, survival, utils, weightedSurv (>= 0.1.0)

Suggests DiagrammeR, doRNG, htmltools, MASS, sandwich, tidyr, forestploter, cubature, svglite, knitr, rmarkdown, katex, latentcor, quarto, testthat (>= 3.0.0)

URL <https://github.com/larry-leon/forestsearch>,
<https://larry-leon.github.io/forestsearch/>

BugReports <https://github.com/larry-leon/forestsearch/issues>

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Larry Leon [aut, cre]

Maintainer Larry Leon <larry.leon.05@post.harvard.edu>

Repository CRAN

Date/Publication 2026-10-10 04:40:02 UTC

Contents

analysis_covariates	7
analyze_subgroup	8
benchmark_spec	9
bootstrap_results	10
bootstrap_ystar	15
build_classification_table	16
build_cox_formula	18
build_estimation_table	19
calibrate_cens_adjust	22
calibrate_glm_interaction	25
calibrate_k_inter	27
calibrate_k_treat	29
calibrate_L_eff	31
check_censoring_dgm	32
collect_results	33
compare_detection_curves	36
compare_fs_subgroups	37
compare_multiple_survreg	38
compare_selection_rules	39
compare_subgroup_sims	43
compute_detection_probability	44
compute_detection_probability_glm	46
compute_dgm_cde	48
compute_frontier_cis	49
compute_node_metrics	51
confint.dina	52
consistency_resample	52
consistency_resample_compare	55
count_boot_id	56
cox_ahr_cde_analysis	57
cox_cs_fit	59

cox_summary	62
create_dgm_for_mrct	63
create_forest_theme	65
create_gbsg_dgm	68
create_glm_row	70
create_grf_config	71
create_summary_table	72
cv_compare_results	75
cv_metrics_tables	76
CV_sgs	77
cv_summary_tables	78
cv_summary_text	79
dina	80
dina_bagged	82
dina_fit	84
dina_fit_bagged	87
dina_frontier	90
dina_subgroup	93
dina_subgroup_bootstrap	98
dina_subgroup_refit	103
dummy_encode	106
d_eff_binary	106
d_eff_continuous	107
d_eff_count	108
d_eff_survival	109
early_stop_decision	109
estimate_propensity_scores	110
evaluate_comparison	112
evaluate_consistency_twostage	113
evaluate_cuts_once	115
evaluate_subgroup_consistency	116
expand_covariates	118
explain_pareto_selection	119
extract_grf_args	120
extract_selected_subgroups	122
figure_note	123
filter_by_lassokeep	124
filter_call_args	124
find_k_inter_for_target_hr	125
find_leaf_split	128
find_required_sample_size	129
fit_causal_forest	130
fit_cox_models	131
fit_policy_trees	132
forestsearch	132
forestsearch_bootstrap_dofuture	154
forestsearch_Kfold	158
forestsearch_KfoldOut	161

forestsearch_tenfold	162
forest_height	165
format_bootstrap_table	166
format_CI	167
format_fs_details	168
format_oc_results	169
fpr_approx	171
fpr_calibration	173
fpr_calibration_transfer	176
frontier_member_flags	178
fs_assert_membership	179
fs_declaration_calibration	180
fs_dgm_feasibility	183
fs_dgm_scale	186
fs_export_maxeff_family	188
fs_family_report	190
fs_fdr_report	191
fs_focus_tag	194
fs_identification_summary	195
fs_identification_table	197
FS_labels	197
fs_mr_oc_summary	198
fs_oc_family_enumerate	200
fs_oc_grid	203
fs_oc_invert	205
fs_oc_predict	207
fs_plot_bias_coverage	210
fs_rule_covariate_pairs	211
fs_scale_se	212
fs_sim_bias_coverage	212
fs_sym_root	214
fs_to_guohe	215
generate_aft_dgm_flex	217
generate_bootstrap_synthetic	222
generate_bootstrap_with_noise	223
generate_detection_curve	226
generate_gbsg_bootstrap_general	227
generate_glm_dgm	228
get_combinations_info	231
get_conf_force	232
get_conf_force_jq	232
get_covs_in	233
get_Cox_sg	234
get_dfpred	235
get_dfRes	236
get_FSdata	238
get_leff	240
get_split_hr_fast	241

get_subgroup_membership	242
gg_forest	242
glm_cs_fit	244
glm_effect_profile	247
grf.subg.eval	249
grf.subg.harm.glm	250
grf.subg.harm.survival	255
guohe_adaptive_r	259
guohe_algorithm3	261
hrCI_format	264
inject_mrct_structure	264
interpret_estimation_table	266
interpret_search_config	267
lasso_selection	269
latest_rds	270
load_leff_calibration	270
make_effect_estimator	271
make_region_model	273
mrct_region_sims	274
mrct_truth_by_region	277
mr_estimates_table	278
n_pcnt	279
pareto_frontier_table	279
plot_forestsearch	282
plot_fs_forestplot	283
plot_fs_sg_plot	284
plot_subgroup_sims_summary	285
plot_detection_curve	287
plot_effect_distribution	288
plot_fs_identification_involvement	290
plot_fs_identification_structure	291
plot_fs_rule_pairs	292
plot_km_band_forestsearch	292
plot_pareto_combined	296
plot_pareto_frontier	298
plot_region_modifier	300
plot_sg_distribution	300
plot_sg_glm_outcomes	302
plot_sg_results	305
plot_sg_weighted_km	307
plot_spline_treatment_effect	310
plot_subgroup_effects	310
plot_subgroup_results_forestplot	312
predict.dina	317
predict_fpr_corrected	317
prepare_subgroup_data	318
print.cox_ahr_cde	319
print.dina_bagged	320

print.dina_subgroup	320
print.dina_subgroup_bootstrap	321
print.dina_subgroup_refit	322
print.forestsearch	322
print.forestsearch_comparison	323
print.fpr_calibration	324
print.fs_bootstrap	324
print.fs_cv_summary	325
print.fs_dgm_scale	326
print.fs_fdr_report	326
print.fs_forestplot	327
print.fs_forest_theme	327
print.fs_kfold	328
print.fs_mr_oc	329
print.fs_sg_plot	329
print.fs_tenfold	330
print.fs_weighted_km	331
print.gbsg_dgm	331
print.glm_dgm	332
print.glm_effect_profile	332
print.guohe_a3	333
print.guohe_ar	333
print.hte_test	334
print.leff_calibration	334
print.multi_survreg_comparison	335
print_grf_details	335
process_conf_force_expr	336
quick_km_band_plot	337
render_forestplot	338
render_reference_table	339
reset_workers	340
rmst_calculation	341
run_null_calibration	342
run_simulation_analysis	344
run_single_consistency_split	349
run_subgroup_sims	350
save_forestplot	353
save_leff_calibration	354
select_best_subgroup	355
sensitivity_analysis_k_inter	356
setup_gbsg_dgm	357
setup_parallel_SGcons	359
sgdist_fig_height	360
SGplot_estimates	361
sg_consistency_out	363
sg_tables	364
SG_tab_estimates	366
simulate_from_dgm	368

simulate_from_gbsg_dgm	371
simulate_from_glm_dgm	372
sim_region_metrics	374
sort_subgroups	374
subgroup.consistency	376
subgroup.search	383
subgroup_cox	385
subgroup_glm	386
suggest_thresholds	389
summarize_bootstrap_events	391
summarize_bootstrap_results	392
summarize_bootstrap_subgroups	393
summarize_cv_results	395
summarize_simulation_results	398
summary.cox_ahr_cde	399
summary.dina	399
summary.forestsearch	400
summary.fpr_calibration	401
summary.subgroup_sims	401
summaryout_mrct	403
test_constant_cate	405
test_hte	406
test_hte_from_forestsearch	407
test_zero_cate	408
validate_grf_data	409
validate_k_inter_effect	409
validate_mrct_data	410
validate_subgroups	411
wilson_ci	412

Index	413
--------------	------------

analysis_covariates	<i>Covariate set handed to forestsearch()/mrct_region_sims() for the paper's observed vs unobserved analysis cases.</i>
---------------------	---

Description

Covariate set handed to forestsearch()/mrct_region_sims() for the paper's observed vs unobserved analysis cases.

Usage

```
analysis_covariates(
  dgm,
  case = c("observed", "unobserved"),
  drop = character(0)
)
```

Arguments

dgm	An MRCT data-generating mechanism returned by <code>inject_mrct_structure()</code> . Its <code>df_super</code> supplies the <code>z_</code> covariate columns and its <code>mrct</code> component supplies the region indicator name and the effect-modifier (X2) variables.
case	Character. "observed" (default) keeps every <code>z_</code> covariate except the region indicator and spline helper columns; "unobserved" additionally drops the effect-modifier (X2) covariates.
drop	Character vector of covariate names (without the <code>z_</code> prefix) to remove from the returned set. Default character (<code>∅</code>) removes nothing.

Value

Character vector of `z_`-prefixed covariate column names.

<code>analyze_subgroup</code>	<i>Analyze subgroup for summary table (OPTIMIZED)</i>
-------------------------------	---

Description

Analyzes a subgroup and returns formatted results for summary table. Uses optimized `cox_summary()` and reduces redundant calculations.

Usage

```
analyze_subgroup(  
  df_sub,  
  outcome.name,  
  event.name,  
  treat.name,  
  strata.name,  
  subgroup_name,  
  hr_a,  
  potentialOutcome.name,  
  return_medians,  
  N  
)
```

Arguments

<code>df_sub</code>	Data frame for subgroup.
<code>outcome.name</code>	Character. Name of outcome variable.
<code>event.name</code>	Character. Name of event indicator variable.
<code>treat.name</code>	Character. Name of treatment variable.
<code>strata.name</code>	Character. Name of strata variable (optional).
<code>subgroup_name</code>	Character. Subgroup name.

hr_a Character. Adjusted hazard ratio (optional).
 potentialOutcome.name Character. Name of potential outcome variable (optional).
 return_medians Logical. Use medians or RMST.
 N Integer. Total sample size.

Value

Character vector of results.

Examples

```
## Not run:
library(survival)
df <- data.frame(
  tte = gbsg$rfstime / 30.4375,
  event = gbsg$status,
  treat = gbsg$hormon
)
analyze_subgroup(
  df_sub = df,
  outcome.name = "tte",
  event.name = "event",
  treat.name = "treat",
  strata.name = NULL,
  subgroup_name = "All",
  hr_a = NA,
  potentialOutcome.name = NULL,
  return_medians = TRUE,
  N = nrow(df)
)

## End(Not run)
```

benchmark_spec	<i>Random-benchmark subgroup specification</i>
----------------	--

Description

Describes the size-matched random benchmark subgroups regenerated in every simulated trial. With the default nested = TRUE, a single index draw of max(sizes) patients is taken per trial and the smaller benchmarks are its prefixes (random15 $\subset$ random20 $\subset$ random40 $\subset$ random60), exactly as in the extreme-subgroups vignettes. Membership columns are named paste0(prefix, sizes) and take values 0/1.

Usage

```
benchmark_spec(sizes = c(60L, 40L, 20L, 15L), nested = TRUE, prefix = "random")
```

Arguments

sizes	Integer vector of benchmark sizes. Order determines the column-creation order only; the draw uses <code>max(sizes)</code> .
nested	Logical. TRUE (default): one draw, prefix nesting – the vignette scheme. FALSE: an independent draw per size, in the order given (consumes more RNG; not the vignette scheme).
prefix	Column-name prefix, default "random".

Value

An object of class "benchmark_spec".

See Also

[run_subgroup_sims\(\)](#)

Examples

```
benchmark_spec()
benchmark_spec(sizes = c(100, 50), prefix = "rnd")
```

bootstrap_results	<i>Bootstrap Results for ForestSearch with Bias Correction</i>
-------------------	--

Description

Runs bootstrap analysis for ForestSearch, fitting Cox models and computing bias-corrected estimates and valid CIs (see vignette for references)

Usage

```
bootstrap_results(
  fs.est,
  df_boot_analysis,
  cox.formula.boot,
  nb_boots,
  show_three,
  H_obs,
  Hc_obs,
  seed = 8316951L,
  estimator_fn = NULL,
  effect_measure = NULL,
  boot_index_mat = NULL,
  mr_in_replicates = FALSE
)
```

Arguments

<code>fs.est</code>	List. ForestSearch results object from forestsearch . Must contain: • <code>df.est</code>: Data frame with analysis data including <code>treat.recommend</code> • <code>confounders.candidate</code>: Character vector of confounder names • <code>args_call_all</code>: List of original forestsearch call arguments
<code>df_boot_analysis</code>	Data frame. Bootstrap analysis data with same structure as <code>fs.est\$df.est</code> . Must contain columns for outcome, event, treatment, and the <code>treat.recommend</code> flag.
<code>cox.formula.boot</code>	Formula. Cox model formula for bootstrap, typically created by build_cox_formula . Should be of form <code>Surv(outcome, event) ~ treatment</code> .
<code>nb_boots</code>	Integer. Number of bootstrap samples to generate (e.g., 500-1000). More iterations provide better bias correction but increase computation time.
<code>show_three</code>	Logical. If TRUE, prints detailed progress for the first three bootstrap iterations for debugging purposes. Default: FALSE.
<code>H_obs</code>	Numeric. Observed log hazard ratio for subgroup H (harm/questionable group, <code>treat.recommend == 0</code>) from original sample. Used as reference for bias correction.
<code>Hc_obs</code>	Numeric. Observed log hazard ratio for subgroup H ^c (complement/recommend, <code>treat.recommend == 1</code>) from original sample. Used as reference for bias correction.
<code>seed</code>	Integer. Random seed for reproducibility. Default 8316951L. Only used when <code>boot_index_mat</code> is NULL; when an explicit index matrix is supplied, it is the source of bootstrap row selection and seed has no effect on bootstrap indices.
<code>estimator_fn</code>	Closure or NULL. Effect-estimator closure from make_effect_estimator for GLM outcomes. When NULL (default), the Cox model via <code>cox.formula.boot</code> is used.
<code>effect_measure</code>	Character or NULL. Effect-measure label for GLM outcomes (e.g. "OR", "RR", "IRR", "MD", "RD", "IRD"), threaded from the bootstrap entry. Controls the scale of the per-resample subgroup-effect diagnostic (<code>hr_sg</code>) for DINA-selected subgroups: ratio measures ("OR", "RR", "IRR") and survival report it on the ratio scale, while identity measures ("MD", "RD", "IRD") report it on the difference scale. NULL (default) applies the ratio (survival) convention.
<code>boot_index_mat</code>	Integer matrix or NULL. Optional pre-generated $B \times N$ matrix of bootstrap row indices. When supplied, row b is used as the resampling vector for iteration b (<code>df_boot_analysis[boot_index_mat[b,],]</code>); the per-iteration call to <code>sample.int()</code> is bypassed. This is the path taken by forestsearch_bootstrap_dofuture , which generates the matrix once on the main process so that the same indices drive both the worker's bootstrap data construction and the <code>Ystar_mat</code> count matrix. When NULL (default), each iteration draws its own indices via <code>sample.int()</code> - retained for backward compatibility with direct callers.
<code>mr_in_replicates</code>	Logical. Whether multiplier resampling runs inside each replicate. Default FALSE, which strips <code>mr_inference</code> from the reconstructed per-replicate call.

When TRUE the flag propagates unchanged and each replicate's MR object is attached to the returned table as the "mr_replicates" attribute (a list of length nb_boots). See [forestsearch_bootstrap_dofuture](#) for the full contract, including why the retained objects are not aggregable.

Value

Data.table with one row per bootstrap iteration and columns:

boot_id Integer. Bootstrap iteration number (1 to nb_boots)

H_biasadj_1 Bias-corrected estimate for H using method 1: $H_{\text{obs}} - (H_{\text{star_star}} - H_{\text{star_obs}})$

H_biasadj_2 Bias-corrected estimate for H using method 2: $2 \cdot H_{\text{obs}} - (H_{\text{star}} + H_{\text{star_star}} - H_{\text{star_obs}})$

Hc_biasadj_1 Bias-corrected estimate for H^c using method 1

Hc_biasadj_2 Bias-corrected estimate for H^c using method 2

max_sg_est Numeric. Maximum subgroup hazard ratio found

L Integer. Number of candidate factors evaluated

max_count Integer. Maximum number of factor combinations

events_H_0 Integer. Number of events in control arm of original subgroup H on bootstrap sample

events_H_1 Integer. Number of events in treatment arm of original subgroup H on bootstrap sample

events_Hc_0 Integer. Number of events in control arm of original subgroup H^c on bootstrap sample

events_Hc_1 Integer. Number of events in treatment arm of original subgroup H^c on bootstrap sample

events_Hstar_0 Integer. Number of events in control arm of new subgroup H^* on original data

events_Hstar_1 Integer. Number of events in treatment arm of new subgroup H^* on original data

events_Hcstar_0 Integer. Number of events in control arm of new subgroup H^{c*} on original data

events_Hcstar_1 Integer. Number of events in treatment arm of new subgroup H^{c*} on original data

tmins_search Numeric. Minutes spent on subgroup search in this iteration

tmins_iteration Numeric. Total minutes for this bootstrap iteration

Pcons Numeric. Consistency p-value for top subgroup

any_found Integer 0/1. Identification flag; equivalent to `as.integer(!is.na(Pcons))`. Added in v0.2.0 to mirror the `fold_summary$any_found` column produced by [forestsearch_tenfold](#), so cross-API diagnostics (`sum(results$any_found == 1L)`) work identically for bootstrap and CV outputs.

hr_sg Numeric. Hazard ratio for top subgroup

N_sg Integer. Sample size of top subgroup

E_sg Integer. Number of events in top subgroup

K_sg Integer. Number of factors defining top subgroup

g_sg Numeric. Subgroup group ID

m_sg Numeric. Subgroup index

M.1 Character. First factor label

M.2 Character. Second factor label

M.3 Character. Third factor label

M.4 Character. Fourth factor label

M.5 Character. Fifth factor label

M.6 Character. Sixth factor label

M.7 Character. Seventh factor label

grf_cuts_b Character. GRF policy-tree cut expressions returned on this bootstrap sample, collapsed with " | " when multiple cuts are produced. NA_character_ when GRF was not used, the bootstrap forestsearch call errored, or GRF returned no cuts. Populated independently of whether a subgroup was identified: GRF may surface a candidate cut on bootstraps where the downstream consistency stage rejects all candidates. Mirrors the grf_cuts column that [forestsearch_tenfold](#) captures in its fold_summary return slot.

Rows where no valid subgroup was found will have NA for bias corrections. The returned object has a "timing" attribute with summary statistics.

Bias Correction Methods

Two bias correction approaches are implemented:

1. Method 1 (Simple Optimism):

$$H_{adj1} = H_{obs} - (H_*^* - H_{obs}^*)$$

where H_*^* is the new subgroup HR on bootstrap data and H_{obs}^* is the new subgroup HR on original data.

2. Method 2 (Double Bootstrap):

$$H_{adj2} = 2 \times H_{obs} - (H_* + H_*^* - H_{obs}^*)$$

where H_* is the original subgroup HR on bootstrap data.

where:

- H_{obs} : Original subgroup HR on original data
- H_{star} : Original subgroup HR on bootstrap data
- H_{star_obs} : New subgroup (found in bootstrap) HR on original data
- H_{star_star} : New subgroup (found in bootstrap) HR on bootstrap data

Computational Details

- Uses doFuture backend for parallel execution (configured externally)
- Sets reproducible seeds: $8316951 + \text{boot} * 100$ for each iteration
- Each bootstrap iteration runs full ForestSearch pipeline including variable selection, subgroup search, and consistency evaluation
- Sequential execution within each bootstrap prevents nested parallelization
- Failed bootstrap iterations generate warnings but don't stop execution
- Confounders are removed from bootstrap data to force fresh variable selection

Bootstrap Configuration

Each bootstrap iteration modifies ForestSearch arguments to:

- **Suppress output:** details, show_candidate_summary, plot.sg, plot.grf all set to FALSE
- **Force re-selection:** grf_res and grf_cuts set to NULL
- **Prevent nested parallel:** parallel_args\$plan = "sequential", workers = 1

Performance Considerations

- Typical runtime: 1-5 seconds per bootstrap iteration
- For 1000 bootstraps with 6 workers: ~3-10 minutes total
- Memory usage scales with dataset size and number of workers
- Consider reducing nb_boots for initial testing (e.g., 100)

Error Handling

The function gracefully handles three failure modes:

1. Bootstrap sample creation fails: Returns row with all NA
2. ForestSearch fails to run: Warns and returns row with all NA
3. ForestSearch runs but finds no subgroup: Returns row with all NA

All three cases ensure the foreach loop can still combine results via rbind.

Note

This function is designed to be called within a foreach loop with %dofuture% operator. It requires:

- All functions in [get_bootstrap_exports](#) to be available in the parallel workers
- Packages listed in BOOTSTRAP_REQUIRED_PACKAGES to be installed
- Proper parallel backend setup via [setup_parallel_SGcons](#)

See Also

[forestsearch_bootstrap_dofuture](#) for the wrapper function that sets up parallelization and calls this function [build_cox_formula](#) for creating the Cox formula [fit_cox_models](#) for initial Cox model fitting [get_Cox_sg](#) for Cox model fitting on subgroups [get_dfRes](#) for processing bootstrap results into confidence intervals [bootstrap_ystar](#) for generating the Ystar matrix

Examples

```
## Not run:
# Typically called via forestsearch_bootstrap_dofuture()
# Manual usage for debugging:

# 1. Fit initial ForestSearch model
fs_result <- forestsearch(
  df.analysis = mydata,
```

```

    outcome.name = "time",
    event.name = "status",
    treat.name = "treatment",
    confounders.name = c("age", "sex", "stage")
  )

# 2. Build Cox formula
cox_formula <- build_cox_formula("time", "status", "treatment")

# 3. Get observed estimates
cox_fits <- fit_cox_models(fs_result$df.est, cox_formula)

# 4. Set up parallel backend
library(doFuture)
plan(multisession, workers = 6)

# 5. Run bootstrap (note: this is already parallelized internally)
boot_results <- bootstrap_results(
  fs.est = fs_result,
  df_boot_analysis = fs_result$df.est,
  cox.formula.boot = cox_formula,
  nb_boots = 100,
  show_three = TRUE,
  H_obs = cox_fits$H_obs,
  Hc_obs = cox_fits$Hc_obs
)

# 6. Check results
summary(boot_results)

# Proportion of bootstraps that found a subgroup
mean(!is.na(boot_results$H_biasadj_2))

## End(Not run)

```

bootstrap_ystar

*Bootstrap Ystar Count Matrix***Description**

Returns the $B \times N$ integer count matrix consumed by the infinitesimal-jackknife variance estimator in [get_dfRes](#). Entry (b, j) is the number of times row j of df was selected on bootstrap iteration b .

Usage

```
bootstrap_ystar(df, nb_boots, seed = 8316951L)
```

Arguments

df	Data frame. Counts are computed by positional row index; df is used only for its row count.
nb_boots	Integer. Number of bootstrap samples.
seed	Integer or NULL. Random seed for reproducibility. Default 8316951L.

Value

Integer matrix with nb_boots rows and nrow(df) columns.

Implementation note (v0.2.0)

This function previously generated the count matrix via a parallel foreach pass, with counts derived by matching against a user-supplied id column. It now runs on the main process and counts by positional row index via `tabulate`, matching how `bootstrap_results` constructs bootstrap data (`df[in_boot, ]`). For data with unique subject ids (the standard clinical-trial case) the two implementations are equivalent. The new implementation is also robust to non-unique id values.

Reproducibility note

Bootstrap indices generated by this version are not bit-identical to those produced by v0.1.x for the same seed (the generator now uses the main-process Mersenne-Twister stream rather than per-task L'Ecuyer streams). Statistical properties of bootstrap summaries are preserved.

Examples

```
## Not run:
df <- data.frame(id = 1:50, tte = rexp(50), event = rbinom(50, 1, 0.6),
                 treat = rep(0:1, 25))
ystar <- bootstrap_ystar(df, nb_boots = 10)
dim(ystar)      # 10 x 50
rowSums(ystar)  # all equal to 50

## End(Not run)
```

build_classification_table

Build Classification Rate Table from Simulation Results

Description

Constructs a publication-quality gt table summarizing subgroup identification and classification rates across one or more data generation scenarios and analysis methods. The layout mirrors Table 4 of Leon et al. (2024) with metrics grouped by model scenario (null / alt) and columns for each analysis method.

Usage

```
build_classification_table(
  scenario_results,
  analyses = NULL,
  digits = 2,
  title = "Subgroup Identification and Classification Rates",
  n_sims = NULL,
  bold_threshold = 0.05,
  font_size = 12,
  subgroup_notation = c("harm", "benefit")
)
```

Arguments

scenario_results	Named list. Each element is itself a list with: results data.table from run_simulation_analysis . label Character scenario label, e.g., "M1". n_sample Integer sample size. dgm DGM object (for true HRs and subgroup prevalence). hypothesis Character: "null" or "alt".
analyses	Character vector of analysis labels to include (e.g., c("FS", "FSlg", "GRF")). When NULL, all unique values of results\$analysis across scenarios are used.
digits	Integer. Decimal places for proportions. Default: 2.
title	Character. Table title. Default: "Subgroup Identification and Classification Rates".
n_sims	Integer. Number of simulations (for subtitle). Default: NULL.
bold_threshold	Numeric. Type I error threshold above which the any(H) value is shown in bold. Set NULL to disable. Default: 0.05.
font_size	Numeric. Font size in pixels for table text. Default: 12. Increase to 14 or 16 for larger display.
subgroup_notation	Character. "harm" (default) labels subgroups as H/Hc; "benefit" labels as G/Gc for benefit-search analyses (treatment switching).

Details

For each scenario the function computes:

- any(H): Proportion of simulations identifying any subgroup.
- sens(H): Mean sensitivity (only under alternative).
- sens(Hc): Mean specificity.
- ppv(H): Mean positive predictive value (only under alternative).
- ppv(Hc): Mean negative predictive value.
- avg|H|: Mean size of identified subgroup (when found).

Under the null hypothesis the rows are reduced to any(H), sens(Hc), ppv(Hc), and avg|H|.

Value

A gt table object.

See Also

[format_oc_results](#), [summarize_simulation_results](#)

Examples

```
## Not run:
# Assemble results from H0 and H1 simulations
scenarios <- list(
  null = list(
    results = results_null, label = "M1",
    n_sample = 700, dgm = dgm_null, hypothesis = "null"
  ),
  alt = list(
    results = results_alt, label = "M1",
    n_sample = 700, dgm = dgm_calibrated, hypothesis = "alt"
  )
)

build_classification_table(scenarios, n_sims = 100)

## End(Not run)
```

build_cox_formula	<i>Build Cox Model Formula</i>
-------------------	--------------------------------

Description

Constructs a Cox model formula from variable names, optionally adjusted for additional covariates.

Usage

```
build_cox_formula(
  outcome.name,
  event.name,
  treat.name,
  adjust_covariates = NULL
)
```

Arguments

outcome.name	Character. Name of outcome variable.
event.name	Character. Name of event indicator variable.
treat.name	Character. Name of treatment variable.

adjust_covariates

Character vector or NULL. Additional model terms appended to the right-hand side after the treatment term. Terms are pasted verbatim, so survival modelling functions such as `strata()` and `pspline()` may be used directly, e.g. `adjust_covariates = "strata(x1)"` produces a stratified Cox model rather than including `x1` as a linear covariate. NULL (default) reproduces the unadjusted, treatment-only formula.

Value

An R formula object for Cox regression. When `adjust_covariates` is supplied, the treatment term is always placed first on the right-hand side.

Examples

```
build_cox_formula("time_months", "event", "treat")
build_cox_formula("time_months", "event", "treat",
  adjust_covariates = c("strata(site)", "age"))
```

build_estimation_table

Build Estimation Properties Table from Simulation Results

Description

Constructs a publication-quality gt table summarizing estimation properties for hazard ratios in the identified subgroup and its complement. The layout mirrors Table 5 of Leon et al. (2024), showing average estimate, empirical SD, min, max, and relative bias for each estimator.

Usage

```
build_estimation_table(
  results,
  dgm,
  analysis_method = "FSlg",
  n_boots = NULL,
  digits = 2,
  title = "Estimation Properties",
  subtitle = NULL,
  font_size = 12,
  cde_H = NULL,
  cde_Hc = NULL,
  subgroup_notation = c("harm", "benefit"),
  trim_threshold = 1000,
  trim_fraction = 0.01
)
```

Arguments

results	data.table or data.frame. Simulation results from <code>run_simulation_analysis</code> , optionally enriched with bootstrap bias-corrected columns (<code>hr.H.bc</code> , <code>hr.Hc.bc</code>).
dgm	DGM object. Used for true parameter values (<code>hr.H.true</code> , <code>hr.Hc.true</code> , and AHR truth via <code>get_dgm_hr</code>).
analysis_method	Character. Which analysis method to tabulate (e.g., "FSlg"). Default: "FSlg".
n_boots	Integer or NULL. Number of bootstraps. When non-NULL, appended to the subtitle as "(B = n_boots bootstraps)". Default: NULL.
digits	Integer. Decimal places. Default: 2.
title	Character. Table title.
subtitle	Character or NULL. Optional user-supplied subtitle text. Auto-populated statistics (method, estimable count, proportion) are always shown. When non-NULL, the custom text is displayed first with stats appended in parentheses, e.g. "My title (FSlg: 18/20 (90%) estimable)".
font_size	Numeric. Font size in pixels for table text. Default: 12. Increase to 14 or 16 for larger display.
cde_H	Numeric or NULL. Controlled direct effect (θ -ddagger(H)) for the true harm subgroup. When non-NULL, an additional b-ddagger bias column is shown. When NULL (default), auto-detected from <code>dgm\$cde_H</code> or <code>dgm\$hazard_ratios\$CDE_harm</code> .
cde_Hc	Numeric or NULL. Controlled direct effect for the complement. Auto-detected analogously.
subgroup_notation	Character. "harm" (default) labels subgroups as H/Hc; "benefit" labels as G/Gc for benefit-search analyses (treatment switching).
trim_threshold	Numeric or NULL. When the raw mean of an estimator's values exceeds this absolute value, that row's Avg / SD / Min / Max are recomputed on the central $1 - 2 * \text{trim_fraction}$ of estimates, and a small footnote marker is placed on the row's Avg cell. The Estimator label is left unchanged so the Unicode notation (e.g., $\hat{\theta}(Q)$) renders normally; only the Avg cell carries the trim indicator. Set NULL to disable trimming entirely. Default: 1000 (a value typical simulation results will never approach unless a small subgroup produces a degenerate effect estimate).
trim_fraction	Numeric in (0, 0.5). Fraction of estimates to trim from each tail (per row) when trimming triggers. Default: 0.01 (lower and upper 1\ used). Has no effect when <code>trim_threshold = NULL</code> .

Details

Uses the paper's notation conventions:

- θ -dagger: Marginal (causal) HR truth
- θ -ddagger: Controlled direct effect (CDE) truth
- θ -hat(H-hat): Plugin Cox estimate in identified subgroup

- $\theta\text{-hat}^*(H\text{-hat})$: Bootstrap bias-corrected estimate

Includes both Cox-based HR and AHR (Average Hazard Ratio from loghr_po) estimators when AHR columns are present in the results.

For each subgroup (H and Hc) the function reports:

- **Avg**: Mean of the estimates across estimable simulations.
- **SD**: Empirical standard deviation.
- **Min / Max**: Range.
- **b-dagger**: Relative bias (percent) vs marginal truth, $100 * (\text{Avg} - \theta_{\text{dagger}}) / \theta_{\text{dagger}}$.
- **b-ddagger** (conditional): Relative bias (percent) vs CDE truth, shown when CDE values are available.

When bootstrap-corrected columns (hr.H.bc, hr.Hc.bc) are present in results, an additional bias-corrected row ($\theta\text{-hat}^*(H\text{-hat})$) is added per subgroup.

When AHR columns (ahr.H.hat, ahr.Hc.hat) are present, AHR estimation rows are appended using the DGM's true AHR values for relative bias calculation.

When CDE columns (cde.H.hat, cde.Hc.hat) are present and CDE truth values are available, CDE estimation rows ($\theta\text{-ddagger}(H\text{-hat})$) are appended. The b-dagger column for CDE rows reports bias relative to the CDE truth rather than the marginal HR.

Value

A gt table object, or NULL if no estimable realizations exist.

See Also

[build_classification_table](#), [format_oc_results](#), [get_dgm_hr](#)

Examples

```
## Not run:
# Basic usage (marginal truth only)
build_estimation_table(
  results = results_alt,
  dgm = dgm_calibrated,
  analysis_method = "FSlg"
)

# With CDE truth (full Table 5 alignment)
build_estimation_table(
  results = results_alt,
  dgm = dgm_calibrated,
  cde_H = 2.25,
  cde_Hc = 0.60
)

## End(Not run)
```

calibrate_cens_adjust *Calibrate Censoring Adjustment to Match DGM Reference Distribution*

Description

Uses root-finding to select a value of cens_adjust for [simulate_from_dgm](#) such that a chosen censoring summary statistic in the simulated data matches the corresponding statistic from the DGM reference data (dgm\$df_super).

Usage

```
calibrate_cens_adjust(
  dgm,
  target = c("rate", "km_median"),
  n = 1000,
  rand_ratio = 1,
  analysis_time = 48,
  max_entry = 24,
  seed = 42,
  interval = c(-3, 3),
  tol = 1e-04,
  tol_rel = 2.5,
  n_eval = 2000,
  verbose = TRUE,
  ...
)
```

Arguments

dgm	An "aft_dgm_flex" object from generate_aft_dgm_flex .
target	Character. Calibration target: "rate" (default) or "km_median".
n	Integer. Sample size passed to simulate_from_dgm . Default 1000.
rand_ratio	Numeric. Randomisation ratio passed to simulate_from_dgm . Default 1.
analysis_time	Numeric. Calendar analysis time passed to simulate_from_dgm . Must be on the DGM time scale. Default 48.
max_entry	Numeric. Maximum staggered entry time passed to simulate_from_dgm . Default 24.
seed	Integer. Base random seed. Each evaluation of the objective function uses this seed for reproducibility. Default 42.
interval	Numeric vector of length 2. Search interval for cens_adjust on the log scale. Default c(-3, 3) (corresponding roughly to a 20-fold decrease/increase in censoring times).
tol	Numeric. Root-finding tolerance. Default 1e-4.

tol_rel	Numeric. Maximum tolerated relative error, in percent; the call stops if the achieved metric is not within tol_rel percent of the reference. Default 2.5.
n_eval	Integer. Sample size used inside the objective function during root-finding. Smaller values are faster but noisier; increase for precision. Default 2000.
verbose	Logical. Print search progress and final result. Default TRUE.
...	Additional arguments passed to simulate_from_dgm (e.g. strata_rand, time_eos).

Details

Two calibration targets are supported:

"rate" Overall censoring rate (proportion censored). Finds cens_adjust such that $\text{mean}(\text{event_sim} == 0)$ in simulated data equals $\text{mean}(\text{event} == 0)$ in `dgm$df_super`.

"km_median" KM-based median censoring time, estimated by reversing the event indicator so censored observations become the "event" of interest. Finds cens_adjust such that the simulated KM median matches the reference KM median.

How the objective function works:

At each candidate cens_adjust value, the objective function:

1. Calls `simulate_from_dgm()` with `n = n_eval` and the candidate cens_adjust.
2. Calls `check_censoring_dgm()` with `verbose = FALSE` to extract the target metric.
3. Returns `sim_metric - ref_metric`.

`uniroot` finds the zero crossing, i.e. the cens_adjust at which simulated and reference metrics are equal.

Monotonicity:

The objective is monotone in cens_adjust for both targets:

- Larger cens_adjust $\rightarrow$ longer censoring times $\rightarrow$ lower censoring rate and higher KM median.
- Smaller cens_adjust $\rightarrow$ shorter censoring times $\rightarrow$ higher censoring rate and lower KM median.

If `uniroot` fails (the target lies outside the search interval), the boundary values are printed and a wider interval should be tried.

Stochastic noise:

Because the objective function involves simulation, there is Monte Carlo noise. Setting a fixed seed and a sufficiently large `n_eval` (≥ 2000) reduces noise enough for reliable root-finding. The `tol` argument controls the root-finding tolerance on the cens_adjust scale (not the metric scale).

Value

A named list with elements:

`cens_adjust` Calibrated cens_adjust value.

`target` Calibration target used.

ref_value Reference metric value from dgm\$df_super.
 sim_value Achieved metric value in simulated data at the calibrated cens_adjust.
 residual Absolute difference between sim_value and ref_value.
 iterations Number of uniroot iterations.
 diagnostic Output of [check_censoring_dgm](#) at the calibrated value (invisibly).

See Also

[simulate_from_dgm](#), [check_censoring_dgm](#), [generate_aft_dgm_flex](#)

Examples

```
## Not run:
library(survival)

# Build DGM on months scale
gbsg$time_months <- gbsg$rfstime / 30.4375

dgm <- generate_aft_dgm_flex(
  data          = gbsg,
  continuous_vars = c("age", "size", "nodes", "pgr", "er"),
  factor_vars    = c("meno", "grade"),
  outcome_var    = "time_months",
  event_var      = "status",
  treatment_var  = "hormon",
  subgroup_vars  = c("er", "meno"),
  subgroup_cuts  = list(er = 20, meno = 0)
)

# Calibrate so simulated censoring rate matches reference
cal_rate <- calibrate_cens_adjust(
  dgm          = dgm,
  target       = "rate",
  n            = 1000,
  analysis_time = 84,
  max_entry    = 24
)
cat("Calibrated cens_adjust (rate):", cal_rate$cens_adjust, "\n")

# Calibrate to KM median censoring time instead
cal_km <- calibrate_cens_adjust(
  dgm          = dgm,
  target       = "km_median",
  n            = 1000,
  analysis_time = 84,
  max_entry    = 24
)
cat("Calibrated cens_adjust (km_median):", cal_km$cens_adjust, "\n")

# Use calibrated value in simulation
sim <- simulate_from_dgm(
```



```

    dgm          = dgm,
    n            = 1000,
    analysis_time = 84,
    max_entry    = 24,
    cens_adjust  = cal_rate$cens_adjust,
    seed         = 123
  )
  mean(sim$event_sim) # event rate
  mean(sim$event_sim == 0) # censoring rate - should match ref

## End(Not run)

```

calibrate_glm_interaction

Calibrate GLM Interaction for a Target Subgroup Effect Size

Description

Searches over a grid of interaction multipliers (k_{inter}) to find the value that produces a target effect size in the subgroup Q . Returns a modified "glm_dgm" object with the calibrated interaction and updated true effects.

Usage

```

calibrate_glm_interaction(
  data,
  factor_vars,
  continuous_vars = NULL,
  outcome_var,
  treatment_var,
  target_effect,
  outcome_type = c("binary", "continuous", "count"),
  effect_measure = NULL,
  offset_var = NULL,
  subgroup_vars = NULL,
  subgroup_cuts = NULL,
  k_treat = 1,
  adverse_outcome = FALSE,
  k_inter_range = c(-10, 10),
  grid_step = 0.05,
  tol_rel = 2.5,
  n_super = 5000L,
  seed = 8316951L,
  verbose = FALSE
)

```

Arguments

<code>data</code>	The source data frame (same as passed to generate_glm_dgm).
<code>factor_vars</code>	Character vector of factor variable names.
<code>continuous_vars</code>	Character vector of continuous prognostic variable names. Passed through to generate_glm_dgm . Default NULL.
<code>outcome_var</code>	Character string naming the outcome variable.
<code>treatment_var</code>	Character string naming the treatment variable.
<code>target_effect</code>	Numeric. Target effect size in Q on the scale determined by <code>effect_measure</code> (e.g., OR = 2.0 for a binary outcome).
<code>outcome_type</code>	Character. One of "binary", "continuous", or "count".
<code>effect_measure</code>	Character. Effect measure. Default NULL (selects default per <code>outcome_type</code>).
<code>offset_var</code>	Character or NULL. Name of the exposure / follow-up time column for count outcomes with an offset (IRR, IRD). Default NULL.
<code>subgroup_vars</code>	Character vector of subgroup-defining variables.
<code>subgroup_cuts</code>	Named list of cutpoint specifications.
<code>k_treat</code>	Numeric. Treatment effect scaling factor passed to generate_glm_dgm . Default 1 (preserve fitted treatment effect). Set to 0 to zero out the ITT effect.
<code>adverse_outcome</code>	Logical. When TRUE for binary outcomes with a ratio effect measure (OR, RR), the calibration searches for $1 / \text{target_effect}$ on the actual Y scale because generate_glm_dgm negates <code>k_inter</code> internally. The returned DGM stores <code>adverse_outcome = TRUE</code> so that downstream functions (e.g., run_simulation_analysis) can auto-coordinate the analysis direction. Default FALSE.
<code>k_inter_range</code>	Numeric vector of length 2. Search range for <code>k_inter</code> . Default <code>c(0, 10)</code> .
<code>grid_step</code>	Numeric. Grid resolution. Default 0.05.
<code>tol_rel</code>	Numeric. Maximum tolerated relative error, in percent; the call stops (fails loudly) if the achieved OR is not within <code>tol_rel</code> percent of the target. Default 2.5.
<code>n_super</code>	Integer. Super-population size. Default 5000L.
<code>seed</code>	Integer. Random seed. Default 8316951L.
<code>verbose</code>	Logical. Print calibration progress. Default FALSE.

Details

The function constructs a grid of `k_inter` values, calls [generate_glm_dgm](#) for each, and selects the value whose subgroup effect is closest to `target_effect`.

For binary outcomes with `effect_measure = "OR"`, the target is on the switched-treatment OR scale ($\text{OR} > 1$ = treatment increases the outcome).

Value

An object of class "glm_dgm" with the calibrated `k_inter` and updated `hazard_ratios`.

See Also

[generate_glm_dgm](#), [simulate_from_glm_dgm](#)

Examples

```
## Not run:
dgm <- calibrate_glm_interaction(
  data          = actg_df,
  factor_vars   = paste0("z", 1:12),
  outcome_var   = "y_binary",
  treatment_var = "treat",
  target_effect = 2.0,
  outcome_type  = "binary",
  subgroup_vars = c("z1", "z2"),
  subgroup_cuts = list(z1 = 1L, z2 = 1L),
  verbose       = TRUE
)
print(dgm)

## End(Not run)
```

calibrate_k_inter

Calibrate k_inter for Target Subgroup Hazard Ratio

Description

Finds the interaction effect multiplier (`k_inter`) that achieves a target hazard ratio in the harm subgroup.

Usage

```
calibrate_k_inter(
  target_hr_harm,
  model = "alt",
  k_treat = 1,
  cens_type = "weibull",
  k_inter_range = c(-100, 100),
  tol = 1e-06,
  tol_rel = 2.5,
  use_ahr = FALSE,
  verbose = FALSE,
  ...
)
```

Arguments

target_hr_harm	Numeric. Target hazard ratio for the harm subgroup
model	Character. Model type ("alt" only). Default: "alt"
k_treat	Numeric. Treatment effect multiplier. Default: 1
cens_type	Character. Censoring type. Default: "weibull"
k_inter_range	Numeric vector of length 2. Search range for k_inter. Default: c(-100, 100)
tol	Numeric. Tolerance for root finding. Default: 1e-6
tol_rel	Numeric. Maximum tolerated relative error, in percent; the call stops if the achieved HR is not within tol_rel percent of the target. Default 2.5.
use_ahr	Logical. If TRUE, calibrate to AHR instead of Cox-based HR. Default: FALSE
verbose	Logical. Print diagnostic information. Default: FALSE
...	Additional arguments passed to create_gbsg_dgm

Details

This function uses uniroot to find the k_inter value such that the empirical HR (or AHR) in the harm subgroup equals target_hr_harm.

Value

Numeric value of k_inter that achieves the target HR

Examples

```
## Not run:
# Find k_inter for HR = 1.5 in harm subgroup
k <- calibrate_k_inter(target_hr_harm = 1.5, verbose = TRUE)

# Verify
dgm <- create_gbsg_dgm(model = "alt", k_inter = k, verbose = TRUE)
print(dgm$hr_H_true) # Should be close to 1.5

# Calibrate to AHR instead
k_ahr <- calibrate_k_inter(target_hr_harm = 1.5, use_ahr = TRUE, verbose = TRUE)
dgm_ahr <- create_gbsg_dgm(model = "alt", k_inter = k_ahr, verbose = TRUE)
print(dgm_ahr$AHR_H_true) # Should be close to 1.5

## End(Not run)
```

calibrate_k_treat	<i>Calibrate k_{treat} for Target Overall Hazard Ratio</i>
-------------------	---

Description

Finds the treatment effect multiplier (k_{treat}) that achieves a target overall hazard ratio in a DGM built by [generate_aft_dgm_flex](#). The root is found by `uniroot` over k_{treat} with every other argument to `generate_aft_dgm_flex()` held fixed.

Usage

```
calibrate_k_treat(
  target_hr_overall,
  base_args,
  k_treat_range = c(-5, 5),
  tol = 1e-06,
  tol_rel = 2.5,
  use_ahr = FALSE,
  verbose = FALSE
)
```

Arguments

<code>target_hr_overall</code>	Numeric. Target overall hazard ratio (must be positive).
<code>base_args</code>	Named list of arguments to <code>generate_aft_dgm_flex()</code> excluding <code>k_treat</code> . Every other parameter (data, covariates, model, subgroup_vars, seed, n_super, etc.) is taken from this list. See Details below for reproducibility considerations.
<code>k_treat_range</code>	Numeric vector of length 2. Search range for <code>k_treat</code> passed to <code>uniroot</code> . Default <code>c(-5, 5)</code> is typically wide enough; widen if the boundary HRs do not bracket the target (<code>uniroot</code> will error otherwise).
<code>tol</code>	Numeric. Tolerance for root finding. Default <code>1e-6</code> .
<code>tol_rel</code>	Numeric. Maximum tolerated relative error, in percent; the call stops if the achieved HR is not within <code>tol_rel</code> percent of the target. Default <code>2.5</code> .
<code>use_ahr</code>	Logical. If TRUE, calibrate to the average hazard ratio (<code>dgm\$hazard_ratios\$AHR</code>) instead of the Cox-based overall HR (<code>dgm\$hazard_ratios\$overall</code>). Default FALSE.
<code>verbose</code>	Logical. Print diagnostic information. Default FALSE.

Details

This is the natural sibling of [calibrate_k_inter](#) (which targets the harm-subgroup HR under `model = "alt"`) and [calibrate_cens_adjust](#) (which targets the censoring rate). It is typically used with `model = "null"` to fix the overall HR for a uniform-effect DGM, but also works with `model = "alt"` when the marginal HR is the calibration target rather than the subgroup-specific HR.

Reproducibility requires the uniroot objective to be deterministic in k_treat. This is achieved by including an integer seed field in base_args so that every call to generate_aft_dgm_flex() during the search uses the same random-number stream. If base_args\$seed is NULL the objective becomes stochastic and uniroot may not converge; the function emits a warning in this case.

Value

Numeric scalar. The calibrated k_treat value. Stops with an error if the target cannot be reached – the search interval fails to bracket it even after automatic extension, or the achieved HR lies outside tol_rel percent of the target.

See Also

[calibrate_k_inter](#), [calibrate_cens_adjust](#), [generate_aft_dgm_flex](#)

Examples

```
## Not run:
library(survival)
data(gbsg)
gbsg$time_months <- gbsg$rfstime / 30.4375

base_args <- list(
  data          = gbsg,
  continuous_vars = c("age", "size", "nodes", "pgr", "er"),
  factor_vars   = c("meno", "grade"),
  outcome_var   = "time_months",
  event_var     = "status",
  treatment_var = "hormon",
  model         = "null",
  n_super       = 5000,
  seed          = 99,
  verbose       = FALSE
)

# Calibrate k_treat so that the overall HR is exactly 0.70
k <- calibrate_k_treat(target_hr_overall = 0.70,
                      base_args         = base_args,
                      verbose            = TRUE)

# Verify
dgm <- do.call(generate_aft_dgm_flex,
              c(base_args, list(k_treat = k)))
print(dgm$hazard_ratios$overall) # ~ 0.70

# Calibrate to AHR instead
k_ahr <- calibrate_k_treat(target_hr_overall = 0.70,
                          base_args         = base_args,
                          use_ahr           = TRUE)
dgm_ahr <- do.call(generate_aft_dgm_flex,
                  c(base_args, list(k_treat = k_ahr)))
print(dgm_ahr$hazard_ratios$AHR) # ~ 0.70
```

```
## End(Not run)
```

calibrate_L_eff	<i>Calibrate L_{eff} from Simulation Results</i>
-----------------	--

Description

Given per-subgroup approximation probabilities and simulated FPR values at multiple sample sizes, fits the power-law model $L_{\text{eff}}(N) = C \cdot (N/n_{\min})^\alpha$.

Usage

```
calibrate_L_eff(N, P1, sim_fpr, n_min = 60)
```

Arguments

N	Integer vector. Sample sizes at which simulations were run.
P1	Numeric vector. Per-subgroup detection probability from compute_detection_probability_glm at each N.
sim_fpr	Numeric vector. Simulated procedure-level FPR at each N.
n_min	Integer. Minimum subgroup size used in ForestSearch. Default: 60.

Details

At each sample size, the implied L_{eff} is:

$$L_{\text{eff}} = \frac{\log(1 - \text{FPR}_{\text{sim}})}{\log(1 - P_1)}$$

A log-linear model is then fit: $\log(L_{\text{eff}}) = \log(C) + \alpha \log(N/n_{\min})$.

Value

A list of class "leff_calibration" with components:

C Numeric. Fitted constant.

alpha Numeric. Fitted power-law exponent.

n_min Integer. Minimum subgroup size.

r_squared Numeric. R-squared of the log-linear fit.

data Data frame with N, P1, sim_fpr, L_eff, and fitted values.

fit The lm object.

Examples

```
# From binary threshold calibration results
cal <- calibrate_L_eff(
  N      = c(200, 500, 700, 1000),
  P1     = c(0.152, 0.109, 0.091, 0.071),
  sim_fpr = c(0.185, 0.220, 0.270, 0.280),
  n_min  = 60
)
print(cal)
```

check_censoring_dgm	<i>Diagnose Censoring Consistency Between DGM Source Data and Simulated Data</i>
---------------------	--

Description

Compares the censoring distribution observed in the data used to build the DGM against the censoring generated by [simulate_from_dgm](#). Reports censoring rates, time quantiles, KM-based median censoring times, and flags substantial discrepancies.

Usage

```
check_censoring_dgm(
  sim_data,
  dgm,
  treat_var = "treat_sim",
  rate_tol = 0.1,
  median_tol = 0.25,
  verbose = TRUE
)
```

Arguments

sim_data	A data.frame returned by simulate_from_dgm .
dgm	An "aft_dgm_flex" object from generate_aft_dgm_flex . The super population (dgm\$df_super) provides reference censoring times and event indicators on the DGM time scale.
treat_var	Character. Name of the treatment column in sim_data used for arm-stratified comparisons. Default "treat_sim".
rate_tol	Numeric. Absolute tolerance (proportion scale) for flagging a censoring-rate discrepancy. Default 0.10 (10 pp).
median_tol	Numeric. Relative tolerance for flagging a KM median censoring-time discrepancy. Default 0.25 (25 percent).
verbose	Logical. If TRUE, prints the full diagnostic table. Default TRUE.

Details

The reference censoring distribution is derived from `dgm$df_super`, sampled with replacement from the data passed to `generate_aft_dgm_flex()`. Columns `y` (observed time) and `event` (event indicator) in `df_super` reflect the original observed censoring process on the DGM time scale.

The KM median censoring time is estimated by reversing the event indicator ($1 - \text{event}$), treating events as censored and censored observations as the event of interest. This gives a non-parametric estimate of the censoring time distribution unconfounded by event occurrence.

Common causes of discrepancy: (1) time-scale mismatch (DGM built on days, `analysis_time` in months); check `exp(dgm$model_params$mu)` against your `analysis_time`. (2) Large `cens_adjust` shifting censoring substantially from the fitted model. (3) Short `analysis_time` or `time_eos` making administrative censoring dominate the censoring process.

Value

Invisibly returns a named list. Elements are: `rates` (data frame of censoring rates overall and by arm); `quantiles` (data frame of censoring-time quantiles among censored subjects); `km_medians` (data frame of KM-based median censoring times); and `flags` (character vector of triggered warnings, empty if none).

See Also

[simulate_from_dgm](#), [generate_aft_dgm_flex](#)

Examples

```
## Not run:
dgm <- setup_gbsg_dgm(model = "null", verbose = FALSE)
sim_data <- simulate_from_dgm(dgm, n = 200)
check_censoring_dgm(sim_data, dgm = dgm)

## End(Not run)
```

collect_results

Collect Results from foreach with Error Handling

Description

Separates successful results from errors in the output of a [foreach](#) loop run with `.errorhandling = "pass"`. Successful results are combined via [rbindlist](#) (with an `rbind` fallback if **data.table** is not available) and failed tasks are counted, reported, and optionally escalated to a hard error.

Usage

```
collect_results(
  raw_list,
  label = "",
  stop_on_all_fail = TRUE,
```

```

    max_error_messages = 3L,
    keep_diagnostics = FALSE
  )

```

Arguments

<code>raw_list</code>	List returned by <code>foreach(..., .errorhandling = "pass")</code> . Each element is either a data frame, <code>data.table</code> , or row-bindable object (success) or a condition object (failure).
<code>label</code>	Character. Optional label included in the warning message to identify which simulation phase produced the errors (e.g. "H1 alt", "Bootstrap"). Default "".
<code>stop_on_all_fail</code>	Logical. If TRUE (default), raise an error when every task in <code>raw_list</code> failed. If FALSE, return an empty data frame and only emit a warning. Default TRUE.
<code>max_error_messages</code>	Integer. Maximum number of distinct error messages to print in the diagnostic output. Default 3L.
<code>keep_diagnostics</code>	Logical. If TRUE, scan each successful element for a "diagnostics" attribute and collect them into a list attached to the returned object as <code>attr(result, "diagnostics")</code> . Used with <code>run_simulation_analysis</code> 's <code>keep/keep_first_n</code> mechanism to retain per-replicate heavy objects (FS or GRF result objects, candidate tables) that <code>data.table::rbindlist()</code> would otherwise drop. Default FALSE (preserves minimal-memory behavior).

Details

This replaces the fragile pattern of calling `nrow()` or `rbind()` directly on a raw `foreach` result list, which fails silently when error objects are mixed in with data-frame rows.

Design rationale for the reporting strategy:

- A **warning** is emitted so that calling code (and `tryCatch` / `withCallingHandlers` blocks) can capture the failure condition programmatically.
- The error summary is also printed with `cat` to `stderr`, because **warning** and **message** output is sometimes buffered or suppressed inside `future/foreach` parallel workers or when `options(warn = -1)` is set by user code upstream. The `cat()` output is the visible-no-matter-what fallback.
- When `stop_on_all_fail = TRUE` (the default), a completely empty result list is escalated to an error rather than silently returning an empty data frame. This avoids downstream code proceeding with zero rows and producing misleading summaries. Calibration code that may legitimately tolerate some failures (e.g. Monte Carlo FPR estimation) can set `stop_on_all_fail = FALSE`.

The returned object is always a data frame – never NULL – because attributes (`n_failed`, `n_total`) attached to NULL are silently discarded by R, whereas attributes on an empty data frame are preserved and inspectable by the caller.

Value

A data frame of row-bound successful results. Always a data frame; **never** NULL – an empty data frame is returned when every task failed and `stop_on_all_fail = FALSE`. Two attributes are always attached:

`n_failed` Integer. Number of tasks that returned errors.

`n_total` Integer. Total number of tasks (successes + failures).

When `keep_diagnostics = TRUE`, a third attribute "diagnostics" may be attached – a named list of per-replicate diagnostic objects, keyed by "sim<ID>" when the source element carries an "sim_id" attribute ([run_simulation_analysis](#) sets this automatically).

Preserving per-replicate diagnostics

`data.table::rbindlist()` drops per-element attributes during row-binding. When the caller has populated each replicate with heavy diagnostic objects (e.g. via [run_simulation_analysis](#) with a non-empty `keep` argument), set `keep_diagnostics = TRUE` to retain them. The diagnostics attribute on each input element must carry `attr(..., "sim_id")` for stable keying; that is set automatically by `run_simulation_analysis()`.

See Also

[reset_workers](#) for resetting the **future** plan and releasing worker memory between simulation phases.

Examples

```
library(foreach)
library(doFuture)
future::plan("sequential")

raw <- foreach(i = 1:5, .errorhandling = "pass") %dofuture% {
  if (i == 3L) stop("simulated failure")
  data.frame(task = i, value = rnorm(1))
}
res <- collect_results(raw, label = "demo")
# Warning: demo: 1 of 5 tasks failed.
nrow(res)
# [1] 4
attr(res, "n_failed")
# [1] 1
```

`compare_detection_curves`*Compare Detection Curves Across Sample Sizes*

Description

Generates and compares detection probability curves for multiple subgroup sample sizes.

Usage

```
compare_detection_curves(  
  n_sg_values,  
  prop_cens = 0.3,  
  hr_threshold = 1.25,  
  hr_consistency = 1,  
  theta_range = c(0.5, 3),  
  n_points = 40L,  
  verbose = TRUE  
)
```

Arguments

<code>n_sg_values</code>	Integer vector. Subgroup sample sizes to compare.
<code>prop_cens</code>	Numeric. Proportion censored. Default: 0.3
<code>hr_threshold</code>	Numeric. HR threshold. Default: 1.25
<code>hr_consistency</code>	Numeric. HR consistency threshold. Default: 1.0
<code>theta_range</code>	Numeric vector of length 2. Range of HR values. Default: c(0.5, 3.0)
<code>n_points</code>	Integer. Number of points per curve. Default: 40
<code>verbose</code>	Logical. Print progress. Default: TRUE

Value

A data.frame with all curves combined, including `n_sg` as a factor.

Examples

```
## Not run:  
comparison <- compare_detection_curves(  
  n_sg_values = c(40, 60, 80, 100),  
  prop_cens = 0.2  
)  
  
# Plot with ggplot2  
library(ggplot2)  
ggplot(comparison, aes(x = theta, y = probability, color = factor(n_sg))) +  
  geom_line(linewidth = 1) +
```

```

labs(x = "True HR", y = "P(Detect)", color = "n_sg") +
theme_minimal()

## End(Not run)

```

compare_fs_subgroups *Compare Subgroup Membership Across Two Analyses*

Description

Cross-tabulates subgroup assignments from two analyses and produces a formatted gt summary table with concordance statistics. Accepts forestsearch result objects, GRF result lists, or raw indicator vectors. Non-detection (no subgroup found) is handled automatically: all subjects are classified to the complement.

Usage

```

compare_fs_subgroups(
  fs1,
  fs2,
  label1 = "Analysis 1",
  label2 = "Analysis 2",
  id.name = NULL,
  sg0_label = NULL,
  sg1_label = NULL,
  subgroup_notation = c("harm", "benefit"),
  title = "Subgroup Membership Comparison",
  subtitle = NULL,
  font_size = 12
)

```

Arguments

fs1	Either a forestsearch result object, a grf.subg.harm result (with data\$treat.recommend), or an integer/logical vector of subgroup indicators (0 = in H/Q, 1 = in Hc/Qc).
fs2	Same types as fs1 for the second analysis.
label1	Character. Display label for the first analysis.
label2	Character. Display label for the second analysis.
id.name	Character. Subject ID column for merge alignment. Only used when both inputs are result objects. If NULL, row-position alignment is assumed.
sg0_label	Character or NULL. Label for H/Q (treat.recommend == 0). If NULL, auto-generated from subgroup_notation.
sg1_label	Character or NULL. Label for Hc/Qc (treat.recommend == 1). If NULL, auto-generated.

subgroup_notation	Character. "harm" uses H/Hc labels; "benefit" uses Q/Qc labels. Only used when sg0_label/sg1_label are NULL.
title	Character. Table title.
subtitle	Character or NULL. Auto-generated if NULL.
font_size	Numeric. Font size for gt table.

Value

A list with:

table A gt cross-tabulation table.

crosstab The raw cross-tabulation matrix.

concordance Agreement, kappa, counts.

membership Per-subject assignments from both analyses.

Examples

```
## Not run:
# Harm search: Cox vs Poisson (both forestsearch objects)
result <- compare_fs_subgroups(fs_cox, fs_pois,
  label1 = "Cox PH", label2 = "Poisson",
  subgroup_notation = "harm")

# Benefit search: FS vs GRF (handles non-detection automatically)
result <- compare_fs_subgroups(fs_result, grf_result,
  label1 = "ForestSearch", label2 = "GRF",
  subgroup_notation = "benefit")

# Raw vectors
result <- compare_fs_subgroups(
  c(rep(0L, 30), rep(1L, 70)),
  c(rep(0L, 25), rep(1L, 75)),
  label1 = "Method A", label2 = "Method B")

## End(Not run)
```

compare_multiple_survreg

Compare Multiple Survival Regression Models

Description

Performs comprehensive comparison of multiple survreg models including convergence checking, information criteria comparison, and model selection.

Usage

```
compare_multiple_survreg(
  ...,
  model_names = NULL,
  verbose = TRUE,
  criteria = c("AIC", "BIC")
)
```

Arguments

<code>...</code>	survreg model objects to compare
<code>model_names</code>	Optional character vector of model names
<code>verbose</code>	Logical, whether to print detailed output (default: TRUE)
<code>criteria</code>	Character vector of criteria to use ("AIC", "BIC", or both)

Value

A list of class "multi_survreg_comparison" containing:

models Named list of input models
convergence Convergence status for each model
comparison Model comparison statistics
rankings Model rankings by different criteria
best_model Name of the best model
recommendation Text recommendation

Examples

```
## Not run:
fit1 <- survreg(Surv(time, status) ~ x, dist = "weibull")
fit2 <- survreg(Surv(time, status) ~ x, dist = "lognormal")
comparison <- compare_multiple_survreg(fit1, fit2)

## End(Not run)
```

compare_selection_rules

Compare forestsearch Runs Across Selection-Rule Combinations

Description

Runs [forestsearch](#) once per combination of `sg_focus[i]` + `selection_rule[i]` (tuple semantics; the two vectors are paired element-by-element and must have the same length). All other arguments are held fixed across runs and passed through via `...`

Usage

```
compare_selection_rules(
  df.analysis,
  sg_focus,
  selection_rule,
  compute_cis = TRUE,
  n_splits = 1000L,
  ci_seed = 1L,
  plot_xlim = NULL,
  show_band = TRUE,
  combo_labels = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

<code>df.analysis</code>	Analysis-ready data frame; passed to <code>forestsearch()</code> .
<code>sg_focus</code>	Character vector of <code>sg_focus</code> values to try.
<code>selection_rule</code>	Character vector of <code>selection_rule</code> values to try. Must have the same length as <code>sg_focus</code> .
<code>compute_cis</code>	Logical. If TRUE (default), runs <code>compute_frontier_cis()</code> for each fit so the plot includes CI bars. Set to FALSE to skip the (often expensive) CI step.
<code>n_splits</code>	Integer. Passed to <code>compute_frontier_cis(n_splits =)</code> . Default 1000L.
<code>ci_seed</code>	Integer. Passed to <code>compute_frontier_cis(seed =)</code> . Default 1L.
<code>plot_xlim</code>	Numeric vector of length 2 (or NULL). Common x-axis range to apply to all plots so they're directly comparable. Default NULL = let each plot auto-range.
<code>show_band</code>	Logical. Whether to draw the effect-band shading on each plot. Default TRUE.
<code>combo_labels</code>	Character vector of length <code>length(sg_focus)</code> , used as the title for each plot and key for the returned lists. Default NULL = auto, e.g. "hrMaxSG / pareto".
<code>verbose</code>	Logical. Print one line per combination during execution. Default TRUE.
<code>...</code>	Additional arguments passed through to every <code>forestsearch()</code> call (data names, thresholds, parallel args, etc.). Do NOT pass <code>sg_focus</code> or <code>selection_rule</code> here – they're handled by the wrapper.

Details

For each combination the wrapper:

1. Captures the in-flight console output from `forestsearch` (including the pre-consistency **Candidate Evaluation Preview** and post-consistency **Candidate Evaluation Summary** when `show_candidate_summary = TRUE` is passed).
2. Optionally computes the frontier CIs via `compute_frontier_cis`.
3. Builds the Pareto-frontier plot via `plot_pareto_frontier`.

The captured stdout is returned per combination so the caller can `cat()` it later for side-by-side inspection in a Quarto chunk. Plot objects are returned individually and (when **patchwork** is installed) composed into a single side-by-side figure. The full `fs` object for each combination is returned so downstream diagnostics ([pareto_frontier_table](#), [explain_pareto_selection](#)) can be run without re-fitting.

Value

A list with class "forestsearch_comparison" and components:

`combos` `data.frame` with columns `sg_focus`, `selection_rule`, `label`.

`fs` Named list of forestsearch result objects, one per combination. `NULL` entries mark fits that errored.

`ci_tab` Named list of frontier-CI tables (or all `NULL` if `compute_cis = FALSE`).

`plots` Named list of `ggplot` objects, one per combination.

`plot_grid` A single **patchwork** object placing the plots side by side, or `NULL` if `patchwork` isn't installed.

`plot_combined` A single `ggplot` composing all configurations onto one Pareto plot, with each winner labeled `S1: <combo_label>`, `S2: <combo_label>`, etc. Returned only when ALL successful configurations share an identical passing set (see [plot_pareto_combined](#) for the equality criterion); `NULL` otherwise. See `plot_combined_subsets` below for the per-subset fallback when some – but not all – configurations match.

`plot_combined_subsets` Named list of `ggplot` objects, one per equivalence class of configurations sharing an identical passing set. Only groups of size ≥ 2 are included (a singleton group has no peer to combine with). Names are derived from the shared `sg_focus` when constant within a group (e.g., "effMaxSG"); from the concatenated combo labels otherwise. When all valid combos share one passing set this list contains exactly one element, equal to `plot_combined`. The list is empty when no two combos match.

`combined_skip_reason` Character scalar or `NULL`. When `plot_combined` is `NULL` because the equality precondition failed, this captures the specific reason – size mismatch, definition-set mismatch, or value drift on a named column – as a single string. `NULL` when the combined plot was built or when fewer than two fits succeeded.

`console` Named list of character vectors – the captured stdout from each forestsearch run (full output).

`diagnostics` Named list of per-combo slices of the captured output: each element is a list with character-scalar fields `preview` (the CANDIDATE EVALUATION PREVIEW block), `summary` (the CANDIDATE EVALUATION SUMMARY block), and `full` (the entire captured stdout, same content as `console[[i]]` collapsed to a single string). `preview` and `summary` are `NA_character_` when the corresponding banner is absent (e.g. `show_candidate_summary` was `FALSE`, or the run errored before reaching it). This slice is what a comparison document should display as the primary diagnostic; `full` (or `console[[i]]`) remains available for an expandable / on-demand view of the complete run output.

`errors` Named list of error messages (or `NULL`) for combos that failed.

Tuple semantics

With `sg_focus = c("hrMaxSG", "hrMaxSG")` and `selection_rule = c("pareto", "both")`, two runs are performed: one with (hrMaxSG, pareto) and one with (hrMaxSG, both). The length-mismatch case raises an error.

Console capture

All stdout from each `forestsearch()` call is captured (not just the diagnostic tables) via `capture.output()`. The PREVIEW / SUMMARY blocks are clearly delineated by banner separators within the captured text. Use `cat(out$console[[i]])` to replay the output in a Quarto chunk; `results = "asis"` works but is not required since the captured text already includes newlines.

See Also

[forestsearch](#), [plot_pareto_frontier](#), [plot_pareto_combined](#), [pareto_frontier_table](#), [explain_pareto_selection](#), [compute_frontier_cis](#).

Examples

```
## Not run:
out <- compare_selection_rules(
  df.analysis      = actg_df,
  sg_focus         = c("hrMaxSG", "hrMaxSG"),
  selection_rule    = c("pareto", "both"),
  # All other forestsearch args:
  confounders.name = confounders.name,
  outcome.name     = adverse_outcome,
  treat.name       = treat.name,
  id.name          = id.name,
  outcome_type     = "continuous",
  effect_measure   = "MD",
  adverse_outcome  = TRUE,
  hr.threshold     = 10,
  hr.consistency   = 5,
  pconsistency.threshold = 0.90,
  show_candidate_summary = TRUE,
  details          = TRUE
)

# Inspect captured console output for combo 1
cat(out$console[[1]])

# Side-by-side plot
print(out$plot_grid)

# Use fs object for downstream diagnostics
pareto_frontier_table(out$fs[[1]], ci_table = out$ci_tab[[1]])
explain_pareto_selection(out$fs[[2]], ci_table = out$ci_tab[[2]])

## End(Not run)
```

compare_subgroup_sims *Compare two extreme-subgroups simulation studies*

Description

Builds the per-subgroup comparison frame between two `run_subgroup_sims()` results (or RDS payloads loaded from the vignettes – any list carrying `sim_hrs` / `sim_ubs` / `sim_ns` matrices with identical column names). For each input the per-design statistics replicate the vignettes' Section 6.6.1 definitions exactly: tail probabilities are percentages denominated over converged fits (`na.rm = TRUE`), medians are unconditional, `N` is the across-trial mean subgroup size, and structurally empty subgroups (all-NA columns) are mapped from `NaN` to `NA`.

Usage

```
compare_subgroup_sims(
  x,
  y,
  expect_designs = NULL,
  suffixes = c("_r", "_f"),
  est_thresholds = NULL,
  ub_thresholds = NULL
)
```

Arguments

<code>x, y</code>	The two studies to compare, in display order (the memo passes random-X then fixed-X).
<code>expect_designs</code>	Optional length-2 character: required design labels for <code>x</code> and <code>y</code> respectively (e.g. <code>c("resample", "fixed")</code>); <code>NULL</code> skips the check.
<code>suffixes</code>	Length-2 character appended to each statistic's column for <code>x</code> and <code>y</code> ; the default <code>c("_r", "_f")</code> reproduces the memo's column names (<code>N_r</code> , <code>N_f</code> , <code>ub2_r</code> , ...).
<code>est_thresholds</code>	Length-2 numeric <code>c(low, high)</code> for the <code>hr05</code> -style (<code>est < low</code>) and <code>hr1</code> -style (<code>est > high</code>) percentages; <code>NULL</code> resolves via effect metadata, then the HR legacy <code>c(0.5, 1.0)</code> .
<code>ub_thresholds</code>	Length-2 numeric for the <code>ub2</code> -style and <code>ub3</code> -style (<code>UB >= t</code>) percentages; resolution as above (HR legacy <code>c(2, 3)</code>).

Details

Column names are structural and retained across outcome types (`hr05`, `hr1`, `ub2`, `ub3`, `mHR`, `mUB`), exactly as `sim_hrs` serves generic duty on GLM results: for `subgroup_glm()` fits they hold estimate-scale statistics at the resolved thresholds. Threshold resolution: explicit arguments win; otherwise the inputs' effect metadata supplies them; otherwise the HR legacy `c(0.5, 1.0)` / `c(2, 3)`. An `NA` threshold yields an all-NA column. The two inputs must carry compatible metadata (both none, or the same effect measure) – comparing an MD study against an HR study is a hard error, raised before the panel-alignment guard so the categorical incompatibility is reported first.

Two validations guard the row-wise alignment (formerly the memo's Guard 1 and Guard 2): the subgroup panels must be identical, and, when `expect_designs` is supplied, each input's design label must match it exactly.

The medians here use `median()`, matching the memo; they are equal to `summary.subgroup_sims()`'s type-7 `quantile(..., 0.50)` values up to floating-point formulation ($(a + b) / 2$ versus $a + 0.5 * (b - a)$), which is why the memo's numbers and the vignettes' tables have always agreed to the printed digit.

Value

A plain `data.frame` (one row per subgroup, matching the memo's static-mode frame type) with columns `subgroup`, then for each of `N`, `ub2`, `ub3`, `mUB`, `hr05`, `hr1`, `mHR` the `x` and `y` columns interleaved. The inputs' `n_sims` and `design` values are attached as attributes "`n_sims`" and "`designs`"; when the inputs carry effect metadata it is attached as attribute "`effect`" (informational; attributes are dropped by most transformations).

See Also

`run_subgroup_sims()`, `summary.subgroup_sims()`, `subgroup_glm()`

Examples

```
## Not run:
pl_r <- readRDS("results/extreme_sims_resample_10000_payload.rds")
pl_f <- readRDS("results/extreme_sims_fixed_10000_payload.rds")
cmp <- compare_subgroup_sims(pl_r, pl_f,
                             expect_designs = c("resample", "fixed"))

## End(Not run)
```

compute_detection_probability

Compute Probability of Detecting True Subgroup

Description

Calculates the probability that a true subgroup with given hazard ratio will be detected using the ForestSearch consistency-based criteria.

Usage

```
compute_detection_probability(
  theta,
  n_sg,
  prop_cens = 0.3,
  hr_threshold = 1.25,
  hr_consistency = 1,
  method = c("cubature", "monte_carlo"),
```

```

    n_mc = 100000L,
    tol = 1e-04,
    verbose = FALSE
  )

```

Arguments

theta	Numeric. True hazard ratio in the subgroup. Can be a vector for computing detection probability across multiple HR values.
n_sg	Integer. Subgroup sample size.
prop_cens	Numeric. Proportion censored (0-1). Default: 0.3
hr_threshold	Numeric. HR threshold for detection (e.g., 1.25). This is the threshold that the average HR across splits must exceed.
hr_consistency	Numeric. HR consistency threshold (e.g., 1.0). This is the threshold each individual split must exceed. Default: 1.0
method	Character. Integration method: "cubature" (recommended for accuracy) or "monte_carlo" (faster for exploration). Default: "cubature"
n_mc	Integer. Number of Monte Carlo samples if method = "monte_carlo". Default: 100000
tol	Numeric. Relative tolerance for cubature integration. Default: 1e-4
verbose	Logical. Print progress for vector inputs. Default: FALSE

Details

This function computes $P(\text{detect} \mid \theta)$ using the asymptotic normal approximation for the log hazard ratio estimator. The detection criterion is based on ForestSearch's split-sample consistency evaluation:

1. The subgroup HR estimate must exceed `hr_threshold` on average
2. Each split-half must individually exceed `hr_consistency`

The approximation assumes:

- Large sample sizes (CLT applies)
- $\text{Var}(\log(\text{HR})) \sim 4/d$ for the full subgroup (d = total subgroup events); each 50/50 split-half ($d/2$ events) has variance $8/d$
- Independence between split-halves (conditional on true effect)

Value

If `theta` is scalar, returns a single probability. If `theta` is a vector, returns a data.frame with columns: `theta`, `probability`.

Examples

```
## Not run:
# Single HR value
prob <- compute_detection_probability(
  theta = 1.5,
  n_sg = 60,
  prop_cens = 0.2,
  hr_threshold = 1.25
)

# Vector of HR values for power curve
hr_values <- seq(1.0, 2.5, by = 0.1)
results <- compute_detection_probability(
  theta = hr_values,
  n_sg = 60,
  prop_cens = 0.2,
  hr_threshold = 1.25,
  verbose = TRUE
)

# Plot detection probability curve
plot(results$theta, results$probability, type = "l",
      xlab = "True HR", ylab = "P(detect)")

## End(Not run)
```

```
compute_detection_probability_glm
```

Compute Detection Probability for GLM Outcomes

Description

Extends the Section 2.1 approximation (Leon et al., 2024, eq. 3) to arbitrary GLM outcomes via the effective information d_{eff} . This is the GLM generalization of [compute_detection_probability](#).

Usage

```
compute_detection_probability_glm(
  theta,
  d_eff,
  c1 = 1.25,
  c2 = 1,
  effect_scale = c("ratio", "difference"),
  method = c("cubature", "monte_carlo"),
  n_mc = 500000L,
  seed = 42L,
  verbose = FALSE
)
```

Arguments

theta	Numeric (scalar or vector). True treatment effect in the subgroup on the natural scale: Ratio measures (HR, OR, IRR) The ratio itself, e.g., 2.0 for doubling. Difference measures (MD) The mean difference, e.g., 0.5.
d_eff	Numeric. Effective information in the subgroup. Use the <code>d_eff_*</code> helpers: d_eff_survival , d_eff_binary , d_eff_count , d_eff_continuous .
c1	Numeric. Screening threshold on the natural scale. For ratio measures: the ratio (e.g., 1.25). For difference measures: the difference (e.g., 0.2).
c2	Numeric. Consistency threshold on the natural scale. Default: 1.0 for ratio, 0.0 for difference.
effect_scale	Character. "ratio" (default) for log-scale effects (HR, OR, IRR); "difference" for identity-scale (MD).
method	Character. "cubature" or "monte_carlo".
n_mc	Integer. Monte Carlo samples if method = "monte_carlo".
seed	Integer. RNG seed for Monte Carlo.
verbose	Logical. Progress messages for vector theta.

Details

The detection criterion (Leon et al., 2024, Section 2.1):

1. Screening: $W_1 + W_2 \geq 2k_1$
2. Consistency: $\min(W_1, W_2) \geq k_2$

with $W_1, W_2 \sim N(\beta, 8/d_{\text{eff}})$ independently.

For ratio-scale effects, $\beta = \log(\theta)$, $k_j = \log(c_j)$. For difference-scale effects, $\beta = \theta$, $k_j = c_j$.

Value

If theta is scalar, a single probability. If vector, a data.frame with columns theta, probability.

See Also

[compute_detection_probability](#) for the survival-specific version.

Examples

```
# Survival: reproduce Figure 2
d <- d_eff_survival(n_sg = 60, prop_cens = 0.45)
compute_detection_probability_glm(theta = 2.0, d_eff = d)

# Binary: OR = 2.0, event rate 30%
d <- d_eff_binary(n_sg = 100, p_event = 0.30)
compute_detection_probability_glm(theta = 2.0, d_eff = d)

# Count: IRR = 2.0, 80 total events
```

```

d <- d_eff_count(total_events = 80)
compute_detection_probability_glm(theta = 2.0, d_eff = d)

# Continuous: mean difference = 0.5, sigma = 1.5
d <- d_eff_continuous(n_sg = 100, sigma_y = 1.5)
compute_detection_probability_glm(
  theta = 0.5, d_eff = d,
  c1 = 0.2, c2 = 0.0,
  effect_scale = "difference"
)

# Power curve
d <- d_eff_binary(100, 0.30)
result <- compute_detection_probability_glm(
  theta = seq(0.5, 3.0, by = 0.1), d_eff = d
)
plot(result$theta, result$probability, type = "l")

```

compute_dgm_cde

*Compute and Attach CDE Values to a DGM Object***Description**

Calculates Controlled Direct Effect (CDE) hazard ratios from the super-population potential outcomes (θ_0 , θ_1) and attaches them to the DGM's `hazard_ratios` list. This enables automatic CDE detection by [build_estimation_table](#).

Usage

```
compute_dgm_cde(dgm, harm_col = NULL)
```

Arguments

<code>dgm</code>	A DGM object (e.g., from <code>create_gbsg_dgm</code> or <code>generate_aft_dgm_flex</code>). Must contain <code>df_super_rand</code> with columns <code>theta_0</code> and <code>theta_1</code> .
<code>harm_col</code>	Character. Name of the subgroup indicator column in <code>dgm\$df_super_rand</code> . If <code>NULL</code> (default), auto-detected from <code>flag.harm</code> , <code>flag_harm</code> , or <code>H</code> .

Details

The CDE for subgroup S is defined as:

$$\text{CDE}(S) = \frac{\text{mean}(\exp(\theta_1[S]))}{\text{mean}(\exp(\theta_0[S]))}$$

which is the ratio of average hazard contributions on the natural scale. This differs from the AHR ($\exp(\text{mean}(\log\text{hr}_{po}))$) due to Jensen's inequality. In the notation of Leon et al. (2024), CDE corresponds to θ -ddagger.

The function detects the subgroup indicator column automatically, checking for `flag.harm`, `flag_harm`, and `H` in the super-population data frame.

Value

The DGM object with CDE values added to `dgm$hazard_ratios` (CDE, CDE_harm, CDE_no_harm) and to top-level fields (`dgm$CDE`, `dgm$cde_H`, `dgm$cde_Hc`).

See Also

[build_estimation_table](#), [get_dgm_hr](#)

Examples

```
## Not run:
dgm <- create_gbsg_dgm(model = "alt", k_inter = 2.0)
dgm <- compute_dgm_cde(dgm)
dgm$hazard_ratios$CDE_harm # theta-ddagger(H)
dgm$hazard_ratios$CDE      # theta-ddagger overall

## End(Not run)
```

compute_frontier_cis	<i>Compute Naive and Split-Derived 95% CIs for Pareto Frontier Members</i>
----------------------	--

Description

For each candidate subgroup on `fs$grp.consistency$out_sg$pareto_frontier`, computes three 95\

Usage

```
compute_frontier_cis(
  fs,
  n_splits = 1000L,
  ci_level = 0.95,
  seed = NULL,
  verbose = FALSE
)
```

Arguments

<code>fs</code>	A forestsearch result object.
<code>n_splits</code>	Integer. Number of 50/50 random splits per frontier member for the Split CI. Default 1000L.
<code>ci_level</code>	Numeric in (0, 1). Default 0.95.
<code>seed</code>	Integer or NULL. Seed for reproducibility of the per-split halving. Default NULL (no explicit seed).
<code>verbose</code>	Logical. Default FALSE. When TRUE, prints per-frontier-member progress.

Details

- **Naive CI:** full-sample Wald CI from a Cox or GLM model refit on the subgroup’s data. This is the textbook CI that ignores the multiple-testing / subgroup-search context. Anti-conservative by construction.
- **Split CI:** a 95% estimate, with SE estimated by the empirical standard deviation of the $2S$ individual half-sample effect estimates produced by `n_splits` random 50/50 splits. This is a half-jackknife SE (Shao1996). Naming: the $\sim$ in column labels marks this as a resampling-derived approximation, not a model-based CI.
- **FSBC-mimic CI:** a bias-corrected interval inspired by the bootstrap bias-correction algorithm of Leon2024fs (eq 7), with the selection-on-bootstrap-data term $\eta_b^*(\hat{H}_b^*)$ zeroed out (the selected subgroup is treated as fixed across half-jackknife replicates). The bias-corrected estimate is $\hat{\beta}^{\text{FSBC}} = 2\hat{\beta} - \hat{\beta}^{(h)}$ where $\hat{\beta}^{(h)}$ is the mean of the $2S$ half-sample estimates; the SE is the same half-jackknife SE as for the Split CI. See “Details — FSBC-mimic interpretation” below.

All three CIs are computed post-hoc on the returned `forestsearch` object; this function does not modify any internal state.

Value

A data table keyed by frontier row index (`m`) with columns:

`m` Candidate id (matches `pareto_frontier$m`).

`estimate` Refit effect estimate on the natural scale.

`naive_lcl`, `naive_ucl` Naive Wald CI bounds.

`split_sd` Half-jackknife SE estimate: empirical SD of the $2S$ per-half effect estimates on the SE scale (log for ratio measures, natural for differences).

`split_lcl`, `split_ucl` Split-derived CI bounds.

`fsbc_estimate` Bias-corrected effect estimate on the natural scale ($2\hat{\beta} - \overline{\hat{\beta}^{(h)}}$ on the SE scale, then back-transformed).

`fsbc_lcl`, `fsbc_ucl` FSBC-mimic CI bounds.

`n_valid_splits` Number of splits that produced valid per-half estimates.

Rows where the refit fails (e.g., zero events in one arm) have NA in all CI columns.

Details — Half-jackknife SE

For each of `n_splits` random 50/50 splits we fit the effect model separately on each half, obtaining $2S$ per-half estimates $\{\hat{\beta}^{(s,h)} : s = 1, \dots, S; h = 1, 2\}$. The half-jackknife SE estimator is

$$\widehat{\text{SE}} = \text{sd}\{\hat{\beta}^{(s,h)}\},$$

i.e., the plain empirical SD of the $2S$ estimates. No $\sqrt{2}$ correction is applied: simulation studies confirm that this estimator’s CI achieves nominal coverage in a no-selection setting (see the package’s pressure-test script). Earlier versions of this function used the average $(\hat{\beta}^{(s,1)} + \hat{\beta}^{(s,2)})/2$ per split; that approach is invalid because the two halves are complementary and their average is nearly constant.

Details — FSBC-mimic interpretation

The FSBC-mimic CI is a *quick approximation* to the bootstrap bias-corrected estimator of *Leon2024fs*. Substituting the $2S$ per-half estimates for bootstrap replicates, and treating the selected subgroup $\hat{H}$ as fixed (i.e., omitting the $\eta_b^*(\hat{H}_b^*) = \hat{\beta}_b^*(\hat{H}_b^*) - \hat{\beta}(\hat{H}_b^*)$ term that captures variability in FS selection on the bootstrap data), the bias-corrected estimator from paper eq (7) collapses to

$$\hat{\beta}^{\text{FSBC}}(\hat{H}) = \hat{\beta}(\hat{H}) - \frac{1}{2S} \sum_{s,h} [\hat{\beta}^{(s,h)} - \hat{\beta}(\hat{H})] = 2\hat{\beta}(\hat{H}) - \overline{\hat{\beta}^{(h)}}.$$

The SE is the same half-jackknife SE as for the Split CI.

The FSBC-mimic CI is *not* a full FSBC CI: it omits the selection-on-bootstrap-data variance. Use this column as a quick bias-corrected diagnostic, not for inference. For full FSBC CIs use [forestsearch_bootstrap_dofuture](#).

See Also

[pareto_frontier_table](#), [frontier_member_flags](#), [plot_pareto_frontier](#).

compute_node_metrics	Compute node metrics for a policy tree
----------------------	--

Description

Aggregates scores by leaf node and calculates treatment effect differences

Usage

```
compute_node_metrics(data, dr.scores, tree, X, n.min)
```

Arguments

data	Data frame. Original data
dr.scores	Matrix. Doubly robust scores
tree	Policy tree object
X	Matrix. Covariate matrix
n.min	Integer. Minimum subgroup size

Value

Data frame with node metrics

Examples

```
## Not run:
# compute_node_metrics() is called internally by grf.subg.harm.survival().
# See grf.subg.harm.survival() for the standard entry point.

## End(Not run)
```

confint.dina	<i>Wald confidence intervals for DINA coefficients.</i>
--------------	---

Description

Computes normal-approximation confidence intervals $\hat{\beta}_j \pm z_{1-\alpha/2} \sqrt{\hat{V}_{jj}}$ using the sandwich (or Cox model-based) variance returned by `dina_fit()`.

Usage

```
## S3 method for class 'dina'
confint(object, parm, level = 0.95, ...)
```

Arguments

<code>object</code>	an object of class "dina".
<code>parm</code>	optional character or integer vector selecting which coefficients to return intervals for; defaults to all.
<code>level</code>	confidence level (default 0.95).
<code>...</code>	unused.

Value

a matrix with two columns giving lower and upper limits, named in standard `confint()` style (e.g. "2.5 %", "97.5 %").

consistency_resample	<i>Resampling approximation to the splitting consistency rate</i>
----------------------	---

Description

Approximates the forestsearch consistency rate for a candidate subgroup without repeated sample splits and refits. The subgroup is fit once; the random-split pair of half-sample treatment effects is represented as `beta_hat +/- D`, a multiplier sum of the per-subject treatment `dfbeta` contributions (see the file header). Handles the Cox (survival) path and the GLM paths (binary OR/RR/RD, continuous MD, count IRR).

Usage

```
consistency_resample(
  df,
  hr.consistency = 1,
  consistency_threshold = NULL,
  comparison_threshold = NULL,
  outcome_type = c("survival", "binary", "continuous", "count"),
  effect_measure = NULL,
  method = c("both", "closed", "mc"),
  multiplier = c("rademacher", "normal", "poisson"),
  draws = 1000L,
  tte.name = "Y",
  event.name = "Event",
  treat.name = "Treat",
  outcome.name = "Y",
  offset.name = NULL,
  cox_init = 0,
  adjust_covariates = NULL,
  adverse_outcome = TRUE,
  seed = NULL
)
```

Arguments

<code>df</code>	data.frame/data.table with the relevant outcome columns.
<code>hr.consistency</code>	Numeric; survival HR consistency threshold (also the default GLM threshold when <code>consistency_threshold</code> is NULL). Default 1.0. This is <code>c2</code> , an effect threshold on the half-sample effect – <code>forestsearch()</code> 's <code>consistency.threshold</code> / <code>hr.consistency</code> . It is not a proportion: the fraction of splits that must clear it is <code>pconsistency.threshold</code> (p^*), which this function does not take – it returns the rate rather than thresholding it.
<code>consistency_threshold</code>	Numeric or NULL; GLM consistency threshold on the natural scale (OR/RR/IRR, or RD/MD/IRD). NULL uses <code>hr.consistency</code> .
<code>comparison_threshold</code>	Numeric or NULL; the consistency threshold already resolved to the comparison scale <code>c</code> ($\log(\text{threshold})$ for ratio measures, identity for identity measures). When supplied it is used directly and the internal transform is bypassed. This is the value <code>forestsearch()</code> threads through for GLM outcomes (it resolves the scale upstream), so callers avoid a double log-transform.
<code>outcome_type</code>	Character; "survival" (default), "binary", "continuous", or "count".
<code>effect_measure</code>	Character or NULL; GLM effect measure (resolved to the outcome-type default when NULL: binary "OR", continuous "MD", count "IRR"). Ignored for survival.
<code>method</code>	Character; "closed", "mc", or "both" (default).
<code>multiplier</code>	Character; "rademacher" (default), "normal", "poisson".
<code>draws</code>	Integer; multiplier draws for the Monte Carlo path.

`tte.name, event.name, treat.name`
 Character; survival/treatment column names.
`outcome.name, offset.name`
 Character; GLM outcome and (rate) offset column names.
`cox_init`
 Numeric; Cox warm-start.
`adjust_covariates`
 Character vector or NULL; adjustment terms.
`adverse_outcome`
 Logical; GLM adverse-outcome convention (see `.consistency_glm_pieces()`).
`seed`
 Integer or NULL; RNG seed for the Monte Carlo draws.

Details

`multiplier = "rademacher"` (default) reproduces the 50/50 Bernoulli split of `run_single_consistency_split()`; "normal" is the Lin (1997) Gaussian multiplier and "poisson" a centred unit-Poisson (DBP) multiplier.

Value

List with `rate_closed`, `rate_mc`, `beta_hat`, `sigma_D`, `n`, `d`, `delta`, `measure`; all-NA rates (with available summaries) if the fit fails or the configuration is unsupported.

See Also

[run_single_consistency_split\(\)](#), [consistency_resample_compare\(\)](#)

Examples

```

## Not run:
# survival (Cox)
consistency_resample(sub, hr.consistency = 1.0)
# binary odds ratio
consistency_resample(sub, outcome_type = "binary", effect_measure = "OR",
                      consistency_threshold = 1.0,
                      outcome.name = "Y", treat.name = "Treat")
# count incidence-rate ratio
consistency_resample(sub, outcome_type = "count", effect_measure = "IRR",
                      consistency_threshold = 1.0,
                      outcome.name = "events", offset.name = "fu_time")

## End(Not run)

```

consistency_resample_compare

Validate the consistency approximation against literal splitting (Cox)

Description

Calls the package's `run_single_consistency_split()` `R_true` times and compares the Monte Carlo rate to the closed-form and multiplier approximations from `consistency_resample()`. Survival path only; for the GLM paths use the standalone GLM validator.

Usage

```
consistency_resample_compare(
  df,
  hr.consistency = 1,
  R_true = 400L,
  draws = 1000L,
  multiplier = "rademacher",
  tte.name = "Y",
  event.name = "Event",
  treat.name = "Treat",
  cox_init = 0,
  adjust_covariates = NULL,
  seed = NULL
)
```

Arguments

<code>df</code>	data.frame/data.table with the relevant outcome columns.
<code>hr.consistency</code>	Numeric; survival HR consistency threshold (also the default GLM threshold when <code>consistency_threshold</code> is NULL). Default 1.0. This is <code>c2</code> , an effect threshold on the half-sample effect – <code>forestsearch()</code> 's <code>consistency.threshold</code> / <code>hr.consistency</code> . It is not a proportion: the fraction of splits that must clear it is <code>pconsistency.threshold</code> (p^*), which this function does not take – it returns the rate rather than thresholding it.
<code>R_true</code>	Integer; number of literal random splits used for the truth.
<code>draws</code>	Integer; multiplier draws for the Monte Carlo path.
<code>multiplier</code>	Character; "rademacher" (default), "normal", "poisson".
<code>tte.name, event.name, treat.name</code>	Character; survival/treatment column names.
<code>cox_init</code>	Numeric; Cox warm-start.
<code>adjust_covariates</code>	Character vector or NULL; adjustment terms.
<code>seed</code>	Integer or NULL; RNG seed for the Monte Carlo draws.

Value

One-row data.frame with n, d, beta_hat, sigma_D, rate_true, valid_true, rate_closed, rate_mc, err_closed, err_mc.

See Also

[consistency_resample\(\)](#), [run_single_consistency_split\(\)](#)

count_boot_id	<i>Count ID Occurrences in Bootstrap Sample</i>
---------------	---

Description

Counts the number of times an ID appears in a bootstrap sample.

Usage

```
count_boot_id(x, dfb)
```

Arguments

- x ID value.
- dfb Data frame of bootstrap sample.

Value

Integer count of occurrences.

Examples

```
df_boot <- data.frame(id = c(1, 2, 1, 3, 1), id_boot = 1:5)
count_boot_id(1, df_boot) # returns 3
count_boot_id(4, df_boot) # returns 0
```

cox_ahr_cde_analysis	<i>Comprehensive Wrapper for Cox Spline Analysis with AHR and CDE Plotting</i>
----------------------	--

Description

This wrapper function combines Cox spline fitting with comprehensive visualization of Average Hazard Ratios (AHRs) and Controlled Direct Effects (CDEs) as described in the MRCT subgroups analysis documentation.

Usage

```
cox_ahr_cde_analysis(
  df,
  tte_name = "os_time",
  event_name = "os_event",
  treat_name = "treat",
  z_name = "biomarker",
  loghr_po_name = "loghr_po",
  theta1_name = "theta_1",
  theta0_name = "theta_0",
  spline_df = 3,
  alpha = 0.2,
  hr_threshold = 0.7,
  plot_style = c("combined", "separate", "grid"),
  plot_select = c("all", "profile_ahr", "ahr_only"),
  save_plots = FALSE,
  output_dir = "plots",
  verbose = TRUE
)
```

Arguments

df	Data frame containing survival data with potential outcomes.
tte_name	Character string specifying time-to-event variable name. Default: "os_time".
event_name	Character string specifying event indicator variable name. Default: "os_event".
treat_name	Character string specifying treatment variable name. Default: "treat".
z_name	Character string specifying continuous covariate/biomarker name. Default: "biomarker".
loghr_po_name	Character string specifying potential outcome log HR variable. Default: "loghr_po".
theta1_name	Optional: variable name for theta_1 (treated potential outcome). Default: "theta_1".
theta0_name	Optional: variable name for theta_0 (control potential outcome). Default: "theta_0".
spline_df	Integer degrees of freedom for spline fitting. Default: 3.
alpha	Numeric significance level for confidence intervals. Default: 0.20.

hr_threshold	Numeric hazard ratio threshold for subgroup identification, or NULL to suppress threshold-based elements. When NULL, plots omit the HR threshold line, optimal cutpoint line, and subgroup-dependent panels (Plots 5–6); subgroup statistics report only the overall population. Default: 0.7.
plot_style	Character: "combined", "separate", or "grid" for plot layout. Default: "combined".
plot_select	Character controlling which panels to display: "all" (default) shows the full grid layout; "profile_ahr" shows only the treatment effect profile (top-left) and the AHR curve for $z \geq \text{threshold}$ (top-middle) side by side, with the AHR panel's y-axis scaled to the data range; "ahr_only" shows only the AHR curve for $z \geq \text{threshold}$ as a single panel with self-contained y-axis.
save_plots	Logical whether to save plots to file. Default: FALSE.
output_dir	Character directory for saving plots. Default: "plots".
verbose	Logical for diagnostic output. Default: TRUE.

Value

List of class "cox_ahr_cde" containing:

cox_fit Results from `cox_cs_fit` function.

ahr_results AHR calculations for different subgroup definitions.

cde_results CDE calculations if theta variables available.

optimal_cutpoint Optimal biomarker cutpoint, or NULL when `hr_threshold` is NULL.

subgroup_stats Statistics for recommended and questionable subgroups, or overall-only when `hr_threshold` is NULL.

data List with `z_values`, `loghr_po`, and subgroup assignments.

Examples

```
## Not run:
# With threshold - full subgroup analysis
results <- cox_ahr_cde_analysis(
  df = df_large, z_name = "z_bm",
  hr_threshold = 1.25, plot_style = "grid"
)

# Without threshold - pure AHR/CDE curves
results <- cox_ahr_cde_analysis(
  df = df_large, z_name = "z_bm",
  hr_threshold = NULL, plot_style = "grid"
)

# Compact two-panel without threshold
results <- cox_ahr_cde_analysis(
  df = df_large, z_name = "z_bm",
  hr_threshold = NULL,
  plot_select = "profile_ahr"
)
```

```
# Single AHR panel only
results <- cox_ahr_cde_analysis(
  df = df_large, z_name = "z_bm",
  hr_threshold = 1.25,
  plot_select = "ahr_only"
)

## End(Not run)
```

cox_cs_fit

Fit Cox Model with Cubic Spline for Treatment Effect Heterogeneity

Description

Estimates treatment effects as a function of a continuous covariate using a Cox proportional hazards model with natural cubic splines. The function models treatment-by-covariate interactions to detect effect modification.

Usage

```
cox_cs_fit(
  df,
  tte_name = "os_time",
  event_name = "os_event",
  treat_name = "treat",
  strata_name = NULL,
  z_name = "bm",
  alpha = 0.2,
  spline_df = 3,
  z_max = Inf,
  z_by = 1,
  z_window = 0,
  z_quantile = 0.9,
  show_plot = TRUE,
  plot_params = NULL,
  truebeta_name = NULL,
  verbose = TRUE
)
```

Arguments

df	Data frame containing survival data
tte_name	Character string specifying time-to-event variable name. Default: "os_time"
event_name	Character string specifying event indicator variable name (1=event, 0=censored). Default: "os_event"

treat_name	Character string specifying treatment variable name (1=treated, 0=control). Default: "treat"
strata_name	Character string specifying stratification variable name. If NULL, no stratification is used. Default: NULL
z_name	Character string specifying continuous covariate name for effect modification. Default: "bm"
alpha	Numeric value for confidence level (two-sided). Default: 0.20 (80% confidence intervals)
spline_df	Integer specifying degrees of freedom for natural spline. Default: 3
z_max	Numeric maximum value for z in predictions. Values beyond this are truncated. Default: Inf (no truncation)
z_by	Numeric increment for z values in prediction grid. Default: 1
z_window	Numeric half-width for counting observations near each z value. Default: 0.0 (exact matches only)
z_quantile	Numeric quantile (0-1) for upper limit of z profile. Default: 0.90 (90th percentile)
show_plot	Logical indicating whether to display plot. Default: TRUE
plot_params	List of plotting parameters (see Details). Default: NULL
truebeta_name	Character string specifying variable containing true log(HR) values for validation/simulation. Default: NULL
verbose	Logical indicating whether to print diagnostic information. Default: TRUE

Details

Model Structure:

The function fits:

$$h(t|Z, A) = h_0(t) \exp(\beta_0 A + f(Z) + g(Z) \cdot A)$$

Where:

- A is treatment (0/1)
- Z is the continuous effect modifier
- f(Z) is modeled with natural splines (main effect)
- g(Z) is modeled with natural splines (interaction)
- The log hazard ratio is: $\beta(Z) = \beta_0 + g(Z)$

Plot Parameters:

The plot_params argument accepts a list with:

- xlab: x-axis label
- main_title: plot title
- ylimit: y-axis limits c(min, max)
- y_pad_zero: padding below zero line
- y_delta: extra space for count labels
- cex_legend: legend text size

- `cex_count`: count text size
- `show_cox_primary`: show standard Cox estimate line
- `show_null`: show null effect line ($\log(\text{HR})=0$)
- `show_target`: show target effect line (e.g., $\log(0.80)$)

Value

List containing:

z_profile Vector of z values where treatment effect is estimated

loghr_est Point estimates of $\log(\text{HR})$ at each z value

loghr_lower Lower confidence bound

loghr_upper Upper confidence bound

se_loghr Standard errors of $\log(\text{HR})$ estimates

counts_profile Number of observations near each z value

cox_primary $\log(\text{HR})$ from standard Cox model (no interaction)

model_fit The fitted coxph model object

spline_basis The natural spline basis object

Examples

```
## Not run:
# Simulate data
set.seed(123)
df <- data.frame(
  os_time = rexp(500, 0.01),
  os_event = rbinom(500, 1, 0.7),
  treat = rbinom(500, 1, 0.5),
  bm = rnorm(500, 50, 10)
)

# Fit model
result <- cox_cs_fit(df, z_name = "bm", alpha = 0.20)

# Custom plotting
result <- cox_cs_fit(
  df,
  z_name = "bm",
  plot_params = list(
    xlab = "Biomarker Level",
    main_title = "Treatment Effect by Biomarker",
    cex_legend = 1.2
  )
)

## End(Not run)
```

cox_summary

Cox model summary for subgroup (OPTIMIZED)

Description

Called in `analyze_subgroup() <- SG_tab_estimates`

Usage

```
cox_summary(
  Y,
  E,
  Treat,
  Strata = NULL,
  use_strata = !is.null(Strata),
  return_format = c("formatted", "numeric")
)
```

Arguments

Y	Numeric vector of outcome.
E	Numeric vector of event indicators.
Treat	Numeric vector of treatment indicators.
Strata	Vector of strata (optional).
use_strata	Logical. Whether to use strata in the model (default: TRUE if Strata provided).
return_format	Character. "formatted" (default) or "numeric" for downstream use.

Details

Calculates hazard ratio and confidence interval for a subgroup using Cox regression. Optimized version with reduced overhead and better error handling.

Value

Character string with formatted HR and CI (or numeric vector if `return_format="numeric"`).

Examples

```
## Not run:
library(survival)
cox_summary(
  Y = gbsg$rfstime / 30.4375,
  E = gbsg$status,
  Treat = gbsg$hormon
)

## End(Not run)
```

create_dgm_for_mrct *Create Data Generating Mechanism for MRCT Simulations*

Description

Wrapper function to create a data generating mechanism (DGM) for MRCT simulation scenarios using [generate_aft_dgm_flex](#).

Usage

```
create_dgm_for_mrct(
  df_case,
  model_type = c("alt", "null"),
  log_hrs = NULL,
  confounder_var = NULL,
  confounder_effect = NULL,
  include_regA = TRUE,
  verbose = FALSE
)
```

Arguments

df_case	Data frame containing case study data
model_type	Character. Either "alt" (alternative hypothesis with heterogeneous treatment effects) or "null" (uniform treatment effect)
log_hrs	Numeric vector. Log hazard ratios for spline specification. If NULL, defaults are used based on model_type
confounder_var	Character. Name of a confounder variable to include with a forced prognostic effect. Default: NULL (no forced effect)
confounder_effect	Numeric. Log hazard ratio for confounder_var effect. Only used if confounder_var is specified
include_regA	Logical. Include regA as a factor in the model. Default: TRUE
verbose	Logical. Print detailed output. Default: FALSE

Details

Model Types:

- alt** Alternative hypothesis: Treatment effect varies by biomarker level (heterogeneous treatment effect). Default log_hrs create HR ranging from 2.0 (harm) to 0.5 (benefit) across biomarker range
- null** Null hypothesis: Uniform treatment effect regardless of biomarker level. Default log_hrs = log(0.7) uniformly

Confounder Effects:

By default, NO prognostic confounder effect is forced. The `confounder_var` and `confounder_effect` parameters allow optionally specifying ANY baseline covariate to have a fixed prognostic effect in the outcome model.

The `regA` variable (region indicator) is included as a factor by default but without a forced effect - its coefficient is estimated from data.

Value

An object of class "aft_dgm_flex" for use with [simulate_from_dgm](#) and [mrct_region_sims](#)

See Also

[generate_aft_dgm_flex](#) for underlying DGM creation [mrct_region_sims](#) for running simulations with the DGM

Examples

```
## Not run:
# Prepare data
df_case <- read.csv("data/dfsynthetic.csv")
df_case$regA <- df_case$region_asia

# Alternative hypothesis (heterogeneous treatment effect)
dgm_alt <- create_dgm_for_mrct(
  df_case = df_case,
  model_type = "alt",
  log_hrs = log(c(3, 1.25, 0.50)),
  verbose = TRUE
)

# Null hypothesis (uniform effect)
dgm_null <- create_dgm_for_mrct(
  df_case = df_case,
  model_type = "null",
  verbose = TRUE
)

# With forced confounder effect
dgm_conf <- create_dgm_for_mrct(
  df_case = df_case,
  model_type = "alt",
  confounder_var = "prior_treat",
  confounder_effect = log(1.5),
  verbose = TRUE
)

## End(Not run)
```

create_forest_theme	<i>Create Forest Plot Theme with Size Controls</i>
---------------------	--

Description

Creates a forestploter theme with parameters that control overall plot sizing and appearance. This is the primary way to control how large the forest plot renders.

Usage

```
create_forest_theme(
  base_size = 10,
  scale = 1,
  row_padding = NULL,
  ci_pch = 15,
  ci_lwd = NULL,
  ci_Theight = NULL,
  ci_col = "black",
  header_fontsize = NULL,
  body_fontsize = NULL,
  footnote_fontsize = NULL,
  footnote_col = "darkcyan",
  title_fontsize = NULL,
  cv_fontsize = NULL,
  cv_col = "gray30",
  refline_lwd = NULL,
  refline_lty = "dashed",
  refline_col = "gray30",
  vertline_lwd = NULL,
  vertline_lty = "dashed",
  vertline_col = "gray20",
  arrow_type = "closed",
  arrow_col = "black",
  summary_fill = "black",
  summary_col = "black"
)
```

Arguments

base_size	Numeric. Base font size in points. This is the primary scaling parameter - increasing it will proportionally scale all fonts, row padding, and line widths. Default: 10.
scale	Numeric. Additional scaling multiplier applied on top of base_size. Use for quick overall scaling. Default: 1.0.
row_padding	Numeric vector of length 2. Padding around row content in mm as c(vertical, horizontal). If NULL, auto-calculated from base_size. Default: NULL.

ci_pch	Integer. Point character for CI. 15=square, 16=circle, 18=diamond. Default: 15.
ci_lwd	Numeric. Line width for CI lines. If NULL, auto-calculated from base_size. Default: NULL.
ci_Theight	Numeric. Height of T-bar ends on CI. If NULL, auto-calculated from base_size. Default: NULL.
ci_col	Character. Color for CI lines and points. Default: "black".
header_fontsize	Numeric. Font size for column headers. If NULL, auto-calculated as base_size * scale + 1. Default: NULL.
body_fontsize	Numeric. Font size for body text. If NULL, auto-calculated as base_size * scale. Default: NULL.
footnote_fontsize	Numeric. Font size for footnotes. If NULL, auto-calculated as base_size * scale - 1. Default: NULL.
footnote_col	Character. Color for footnote text. Default: "darkcyan".
title_fontsize	Numeric. Font size for title. If NULL, auto-calculated as base_size * scale + 4. Default: NULL.
cv_fontsize	Numeric. Font size for CV annotation text. If NULL, auto-calculated as base_size * scale. Default: NULL.
cv_col	Character. Color for CV annotation text. Default: "gray30".
refline_lwd	Numeric. Reference line width. If NULL, auto-calculated. Default: NULL.
refline_lty	Character. Reference line type. Default: "dashed".
refline_col	Character. Reference line color. Default: "gray30".
vertline_lwd	Numeric. Vertical line width. If NULL, auto-calculated. Default: NULL.
vertline_lty	Character. Vertical line type. Default: "dashed".
vertline_col	Character. Vertical line color. Default: "gray20".
arrow_type	Character. Arrow type: "open" or "closed". Default: "closed".
arrow_col	Character. Arrow color. Default: "black".
summary_fill	Character. Fill color for summary diamonds. Default: "black".
summary_col	Character. Border color for summary diamonds. Default: "black".

Details

The base_size parameter is the primary way to control plot size. When you change base_size, the following are automatically scaled:

- All font sizes (body, header, footnote, CV, title)
- Row padding (vertical and horizontal)
- CI line width and T-bar height
- Reference and vertical line widths

The scaling formula uses base_size = 10 as the reference point:

- base_size = 10: Default sizing
- base_size = 12: 20% larger
- base_size = 14: 40% larger
- base_size = 16: 60% larger

You can override any individual parameter by specifying it explicitly.

The theme does NOT set row background colors - those are determined automatically by `plot_subgroup_results_forestplot` based on row types (ITT, reference, posthoc, etc.).

Value

A list of class "fs_forest_theme" containing all theme parameters.

See Also

[plot_subgroup_results_forestplot](#), [render_forestplot](#)

Examples

```
## Not run:
# Simple: just increase base_size for larger plot
large_theme <- create_forest_theme(base_size = 14)

# Or use scale for quick adjustment
large_theme <- create_forest_theme(base_size = 10, scale = 1.4)

# Fine-tune specific elements
custom_theme <- create_forest_theme(
  base_size = 14,
  cv_fontsize = 12, # Override auto-calculated CV font size
  ci_lwd = 2.5      # Override auto-calculated CI line width
)

# Use with plot_subgroup_results_forestplot
result <- plot_subgroup_results_forestplot(
  fs_results = list(fs.est = fs, fs_bc = fs_bc),
  df_analysis = df.analysis,
  outcome.name = "time",
  event.name = "status",
  treat.name = "treatment",
  theme = large_theme
)

render_forestplot(result)

## End(Not run)
```

create_gbsg_dgm

Create GBSG-Based AFT Data Generating Mechanism

Description

Creates a data generating mechanism (DGM) for survival simulations based on the German Breast Cancer Study Group (GBSG) dataset. Supports heterogeneous treatment effects via treatment-subgroup interactions.

Usage

```
create_gbsg_dgm(
  model = c("alt", "null"),
  k_treat = 1,
  k_inter = 1,
  k_z3 = 1,
  z1_quantile = 0.25,
  n_super = DEFAULT_N_SUPER,
  cens_type = c("weibull", "uniform"),
  use_rand_params = FALSE,
  seed = SEED_BASE,
  verbose = FALSE
)
```

Arguments

model	Character. Either "alt" for alternative hypothesis with heterogeneous treatment effects, or "null" for uniform treatment effect. Default: "alt"
k_treat	Numeric. Treatment effect multiplier applied to the treatment coefficient from the fitted AFT model. Values > 1 strengthen the treatment effect. Default: 1
k_inter	Numeric. Interaction effect multiplier for the treatment-subgroup interaction ($z1 * z3$). Only used when model = "alt". Higher values create more heterogeneity between HR(H) and HR(Hc). Default: 1
k_z3	Numeric. Effect multiplier for the z3 (menopausal status) coefficient. Default: 1
z1_quantile	Numeric. Quantile threshold for z1 (estrogen receptor). Observations with ER <= quantile are coded as z1 = 1. Default: 0.25
n_super	Integer. Size of super-population for empirical HR estimation. Default: 5000
cens_type	Character. Censoring distribution type: "weibull" or "uniform". Default: "weibull"
use_rand_params	Logical. If TRUE, modifies confounder coefficients using estimates from randomized subset (meno == 0). Default: FALSE
seed	Integer. Random seed for super-population generation. Default: 8316951
verbose	Logical. Print diagnostic information. Default: FALSE

Details

This version is aligned with `generate_aft_dgm_flex()` and `calculate_hazard_ratios()` methodology, computing individual-level potential outcomes and average hazard ratios (AHR).

Subgroup Definition:

The harm subgroup H is defined as: $z1 = 1$ AND $z3 = 1$, where:

- $z1$: Low estrogen receptor ($ER \leq 25$ th percentile by default)
- $z3$: Premenopausal status ($meno == 0$)

Model Specification:

The AFT model uses covariates: `treat`, $z1$, $z2$, $z3$, $z4$, $z5$, and (for "alt") the interaction $zh = treat * z1 * z3$.

Interaction Effect (k_inter):

The k_inter parameter modifies the zh coefficient in the AFT model:

```
gamma[zh] <- k_inter * gamma[zh]
```

This affects the hazard ratio for the harm subgroup:

- $HR(H) = \exp(-\gamma[treat]/\sigma - \gamma[zh]/\sigma)$
- $HR(Hc) = \exp(-\gamma[treat]/\sigma)$

When $k_inter = 0$, $HR(H) = HR(Hc)$ (no heterogeneity).

Alignment with `generate_aft_dgm_flex`:

This function now computes:

- θ_0 : Log-hazard contribution under control
- θ_1 : Log-hazard contribution under treatment
- $\loghr_po$: Individual causal log hazard ratio ($\theta_1 - \theta_0$)
- AHR metrics: $\exp(\text{mean}(\loghr_po))$ for overall and subgroups

Value

A list of class "gbsg_dgm" containing:

df_super_rand Data frame with randomized super-population including potential outcomes (θ_0 , θ_1 , $\loghr_po$)

hr_H_true Empirical hazard ratio in harm subgroup (Cox-based)

hr_Hc_true Empirical hazard ratio in complement subgroup (Cox-based)

hr_causal Overall causal (ITT) hazard ratio (Cox-based)

AHR Overall average hazard ratio (from $\loghr_po$)

AHR_H_true Average hazard ratio in harm subgroup

AHR_Hc_true Average hazard ratio in complement subgroup

hazard_ratios List matching `generate_aft_dgm_flex` output format

model_params List with AFT model parameters (μ , σ , γ , etc.)

cens_params List with censoring model parameters

subgroup_info List with subgroup definitions and true factor names

analysis_vars Character vector of analysis variable names

model_type Character indicating "alt" or "null"

See Also

[simulate_from_gbsg_dgm](#) for generating data from the DGM [calibrate_k_inter](#) for finding `k_inter` to achieve target HR

Examples

```
## Not run:
# Alternative hypothesis with default parameters
dgm_alt <- create_gbsg_dgm(model = "alt", verbose = TRUE)

# Null hypothesis
dgm_null <- create_gbsg_dgm(model = "null", verbose = TRUE)

# Custom subgroup HR via k_inter
dgm_custom <- create_gbsg_dgm(
  model = "alt",
  k_treat = 1.2,
  k_inter = 2.0,
  verbose = TRUE
)

# Access AHR metrics (aligned with generate_aft_dgm_flex)
dgm_alt$hazard_ratios$AHR_harm
dgm_alt$hazard_ratios$AHR_no_harm

## End(Not run)
```

create_glm_row

Compute GLM Effect Estimate for a Single Subgroup

Description

Fits a GLM within the specified subgroup and returns a one-row data frame containing the effect estimate, confidence interval, sample size, and event count. Supports binary, continuous, and count (with offset) outcomes.

Usage

```
create_glm_row(
  df,
  outcome.name,
  treat.name = "treat",
  effect_measure = "log_OR",
  offset.name = NULL,
  overdispersion = "none",
  conf.level = 0.95,
  min_arm_n = 5L,
```

```

    min_arm_events = 3L,
    verbose = FALSE
  )

```

Arguments

<code>df</code>	Data frame for the subgroup (already subset).
<code>outcome.name</code>	Character. Name of the outcome column.
<code>treat.name</code>	Character. Name of the treatment indicator.
<code>effect_measure</code>	Character. Effect measure; see grf.subg.harm.glm for options.
<code>offset.name</code>	Character or NULL. Name of the person-time column. When supplied, <code>log(offset)</code> is added as an offset term. Default: NULL.
<code>overdispersion</code>	Character. One of "none", "quasi", "negbin". Default: "none".
<code>conf.level</code>	Numeric. Confidence level. Default: 0.95.
<code>min_arm_n</code>	Integer. Minimum observations per arm; returns NA row if either arm is below this threshold. Default: 5L.
<code>min_arm_events</code>	Integer. For binary/count outcomes, minimum events per arm; returns NA row if check fails. Default: 3L.
<code>verbose</code>	Logical. Default: FALSE.

Value

A one-row data frame with columns: `N`, `n_treat`, `n_control`, `events_treat`, `events_control` (for binary/count), `est`, `lower`, `upper`, `se`, `effect_measure`.

See Also

[grf.subg.harm.glm](#)

<code>create_grf_config</code>	<i>Helper Functions for GRF Subgroup Analysis</i>
--------------------------------	---

Description

This file contains helper functions used by `grf.subg.harm.survival()` to improve readability and modularity. Create GRF configuration object

Usage

```

create_grf_config(
  frac.tau,
  n.min,
  dmin.grf,
  RCT,
  sg.criterion,
  maxdepth,
  seedit
)

```

Arguments

frac.tau	Numeric. Fraction of tau for GRF horizon
n.min	Integer. Minimum subgroup size
dmin.grf	Numeric. Minimum difference in subgroup mean
RCT	Logical. Is the data from a randomized controlled trial?
sg.criterion	Character. Subgroup selection criterion
maxdepth	Integer. Maximum tree depth
seedit	Integer. Random seed

Details

Creates a configuration object to organize GRF parameters

Value

List with configuration parameters

Examples

```
cfg <- create_grf_config(frac.tau = 0.6, n.min = 60, dmin.grf = 6,
                        RCT = TRUE, sg.criterion = "mDiff",
                        maxdepth = 2, seedit = 42L)
str(cfg)
```

create_summary_table	<i>Create Enhanced Summary Table for Baseline Characteristics</i>
----------------------	---

Description

Generates a formatted summary table comparing baseline characteristics between treatment arms. Supports continuous, categorical, and binary variables with p-values, standardized mean differences (SMD), and missing data summaries.

Usage

```
create_summary_table(
  data,
  treat_var = "treat",
  vars_continuous = NULL,
  vars_categorical = NULL,
  vars_binary = NULL,
  var_labels = NULL,
  digits = 1,
  show_pvalue = TRUE,
  show_smd = TRUE,
```



```

    show_missing = TRUE,
    table_title = "Baseline Characteristics by Treatment Arm",
    table_subtitle = NULL,
    source_note = NULL,
    font_size = 12,
    header_font_size = 14,
    footnote_font_size = 10,
    use_alternating_rows = TRUE,
    stripe_color = "#f9f9f9",
    indent_size = 20,
    highlight_pval = 0.05,
    highlight_smd = 0.2,
    highlight_color = "#fff3cd",
    compact_mode = FALSE,
    column_width_var = 200,
    column_width_stats = 120,
    show_column_borders = FALSE,
    custom_css = NULL
  )

```

Arguments

<code>data</code>	Data frame containing the analysis data
<code>treat_var</code>	Character. Name of treatment variable (must have 2 levels)
<code>vars_continuous</code>	Character vector. Names of continuous variables
<code>vars_categorical</code>	Character vector. Names of categorical variables
<code>vars_binary</code>	Character vector. Names of binary (0/1) variables
<code>var_labels</code>	Named list. Custom labels for variables (optional)
<code>digits</code>	Integer. Number of decimal places for continuous variables
<code>show_pvalue</code>	Logical. Include p-values column
<code>show_smd</code>	Logical. Include SMD (effect size) column
<code>show_missing</code>	Logical. Include missing data rows
<code>table_title</code>	Character. Main title for the table
<code>table_subtitle</code>	Character. Subtitle for the table (optional)
<code>source_note</code>	Character. Source note at bottom (optional)
<code>font_size</code>	Numeric. Base font size in pixels (default: 12)
<code>header_font_size</code>	Numeric. Header font size in pixels (default: 14)
<code>footnote_font_size</code>	Numeric. Footnote font size in pixels (default: 10)
<code>use_alternating_rows</code>	Logical. Apply zebra striping (default: TRUE)

stripe_color	Character. Color for alternating rows (default: "#f9f9f9")
indent_size	Numeric. Indentation for sub-levels in pixels (default: 20)
highlight_pval	Numeric. Highlight p-values below this threshold (default: 0.05)
highlight_smd	Numeric. Highlight SMD values above this threshold (default: 0.2)
highlight_color	Character. Color for highlighting (default: "#ff3cd")
compact_mode	Logical. Reduce spacing for compact display (default: FALSE)
column_width_var	Numeric. Width for Variable column in pixels (default: 200)
column_width_stats	Numeric. Width for stat columns in pixels (default: 120)
show_column_borders	Logical. Show vertical column borders (default: FALSE)
custom_css	Character. Additional custom CSS styling (optional)

Details

Binary variables specified via `vars_binary` display a single row showing the count and proportion for the "1" level. Categorical variables specified via `vars_categorical` that happen to be binary-coded (i.e., have exactly two levels: 0 and 1) are automatically detected and displayed in the same compact single-row format, showing only the "1" proportion.

Value

A gt table object (or data frame if gt not available)

Examples

```
## Not run:
# Basic usage
create_summary_table(
  data = trial_data,
  treat_var = "treatment",
  vars_continuous = c("age", "bmi"),
  vars_categorical = c("sex", "stage")
)

# Customized appearance
create_summary_table(
  data = trial_data,
  treat_var = "treatment",
  vars_continuous = c("age", "bmi"),
  vars_categorical = c("sex", "stage"),
  font_size = 11,
  header_font_size = 13,
  use_alternating_rows = TRUE,
  highlight_pval = 0.05,
  compact_mode = TRUE
)
```

```
## End(Not run)
```

cv_compare_results	<i>Compare Multiple CV Results</i>
--------------------	------------------------------------

Description

Creates a comparison table from multiple cross-validation runs with different configurations.

Usage

```
cv_compare_results(
  cv_list,
  metrics = c("all", "finding", "agreement"),
  show_percentages = TRUE,
  digits = 1,
  use_gt = TRUE
)
```

Arguments

cv_list	Named list of cv_result objects from forestsearch_tenfold() or forestsearch_Kfold().
metrics	Character vector. Which metrics to include. Options: "finding", "agreement", "all". Default: "all".
show_percentages	Logical. Display as percentages. Default: TRUE.
digits	Integer. Decimal places. Default: 1.
use_gt	Logical. Return gt table if TRUE. Default: TRUE.

Value

A gt table or data.frame comparing CV results across configurations.

Examples

```
## Not run:
# Compare CV results from different configurations
cv_comparison <- cv_compare_results(
  cv_list = list(
    "maxk=1" = cv_maxk1,
    "maxk=2" = cv_maxk2,
    "maxk=3" = cv_maxk3
  )
)

## End(Not run)
```

cv_metrics_tables	Create Metrics Tables for Cross-Validation Results
-------------------	--

Description

Formats the `find_summary` and `sens_summary` outputs from `forestsearch_tenfold` or `forestsearch_Kfold` into publication-ready gt tables.

Usage

```
cv_metrics_tables(
  cv_result,
  sg_definition = NULL,
  title = "Cross-Validation Metrics",
  show_percentages = TRUE,
  digits = 1,
  include_raw = FALSE,
  table_style = c("combined", "separate", "minimal"),
  use_gt = TRUE
)
```

Arguments

<code>cv_result</code>	List. Result from <code>forestsearch_tenfold()</code> or <code>forestsearch_Kfold()</code> . Must contain <code>find_summary</code> and <code>sens_summary</code> elements.
<code>sg_definition</code>	Character vector. Subgroup factor definitions for labeling (optional). If <code>NULL</code> , extracted from <code>cv_result\$sg_analysis</code> .
<code>title</code>	Character. Main title for combined table. Default: "Cross-Validation Metrics".
<code>show_percentages</code>	Logical. Display metrics as percentages (0-100) instead of proportions (0-1). Default: <code>TRUE</code> .
<code>digits</code>	Integer. Decimal places for formatting. Default: 1.
<code>include_raw</code>	Logical. Include raw matrices (<code>sens_out</code> , <code>find_out</code>) in the output for detailed analysis. Default: <code>FALSE</code> .
<code>table_style</code>	Character. One of "combined", "separate", or "minimal". "combined": Single table with both agreement and finding metrics "separate": Two separate gt tables "minimal": Compact single-row summary Default: "combined".
<code>use_gt</code>	Logical. Return gt table(s) if <code>TRUE</code> , <code>data.frame(s)</code> if <code>FALSE</code> . Default: <code>TRUE</code> .

Value

Depending on table_style:

- "combined": A single gt table (or data.frame)
- "separate": A list with agreement_table and finding_table
- "minimal": A single-row gt table (or data.frame)

If include_raw = TRUE, also includes sens_out and find_out matrices in the returned list.

See Also

[cv_summary_tables](#) for formatting forestsearch_KfoldOut(outall=TRUE) results

Examples

```
## Not run:
# After running forestsearch_tenfold
tenfold_results <- forestsearch_tenfold(
  fs.est = fs_result,
  sims = 100,
  Kfolds = 10
)

# Create combined metrics table
cv_tables <- cv_metrics_tables(tenfold_results)
cv_tables

# Create separate tables
cv_tables <- cv_metrics_tables(tenfold_results, table_style = "separate")
cv_tables$agreement_table
cv_tables$finding_table

# Minimal one-row summary
cv_metrics_tables(tenfold_results, table_style = "minimal")

## End(Not run)
```

Description

Summarizes the match between cross-validation subgroups and analysis subgroups.

Usage

```
CV_sgs(sg1, sg2, confs, sg_analysis)
```

Arguments

sg1	Character vector. Subgroup 1 labels for each fold.
sg2	Character vector. Subgroup 2 labels for each fold.
confs	Character vector. Confounder names.
sg_analysis	Character vector. Subgroup analysis labels.

Value

List with indicators for any match, exact match, one match, and covariate-specific matches.

Examples

```
## Not run:
# CV_sgs() is called internally by forestsearch_KfoldOut().
# See forestsearch_Kfold() and forestsearch_KfoldOut() for the entry points.

## End(Not run)
```

cv_summary_tables	Create Summary Tables from forestsearch_KfoldOut Results
-------------------	--

Description

Formats the detailed output from `forestsearch_KfoldOut(outall=TRUE)` into publication-ready gt tables. This includes ITT estimates, original subgroup estimates, and K-fold subgroup estimates.

Usage

```
cv_summary_tables(
  kfold_out,
  title = "Cross-Validation Summary",
  subtitle = NULL,
  show_metrics = TRUE,
  digits = 3,
  font_size = 12,
  use_gt = TRUE
)
```

Arguments

kfold_out	List. Result from <code>forestsearch_KfoldOut(res, outall = TRUE)</code> . Must contain <code>itt_tab</code> , <code>SG_tab_original</code> , <code>SG_tab_Kfold</code> , and optionally <code>tab_all</code> .
title	Character. Main title for combined table. Default: "Cross-Validation Summary".
subtitle	Character. Subtitle for table. Default: NULL (auto-generated).
show_metrics	Logical. Include agreement and finding metrics in output. Default: TRUE.

digits	Integer. Decimal places for numeric formatting. Default: 3.
font_size	Integer. Font size in pixels. Default: 12.
use_gt	Logical. Return gt table if TRUE, data.frame if FALSE. Default: TRUE.

Value

If use_gt = TRUE, returns a list with gt table objects:

- combined_table: Combined ITT and subgroup estimates
- itt_table: ITT estimates only
- original_table: Original full-data subgroup estimates
- kfold_table: K-fold subgroup estimates
- metrics_table: Agreement and finding metrics (if show_metrics = TRUE)

If use_gt = FALSE, returns equivalent data.frames.

See Also

[cv_metrics_tables](#) for formatting forestsearch_tenfold() results

Examples

```
## Not run:
# Run K-fold CV
cv_results <- forestsearch_Kfold(fs.est = fs_result, Kfolds = 10)

# Get detailed output
kfold_out <- forestsearch_KfoldOut(cv_results, outall = TRUE)

# Create summary tables
cv_tables <- cv_summary_tables(kfold_out)
cv_tables$combined_table
cv_tables$metrics_table

## End(Not run)
```

cv_summary_text

Create Compact CV Summary Text

Description

Generates a compact text string summarizing CV results, suitable for annotations in plots or reports.

Usage

```
cv_summary_text(
  cv_result,
  est.scale = "hr",
  include_finding = TRUE,
  include_agreement = TRUE
)
```

Arguments

`cv_result` List. Result from `forestsearch_tenfold()` or `forestsearch_Kfold()`.

`est.scale` Character. "hr" or "1/hr" to determine label orientation. Default: "hr".

`include_finding` Logical. Include subgroup finding rate. Default: TRUE.

`include_agreement` Logical

Value

Character string with formatted CV metrics.

Examples

```
## Not run:
cv_text <- cv_summary_text(tenfold_results)
# Returns: "CV found = 95%, Agree(+,-) = 88%, 92%"

## End(Not run)
```

dina	<i>Estimate heterogeneous treatment effects via DINA (data-frame interface)</i>
------	---

Description

`dina()` is the high-level user-facing wrapper around `dina_fit()`. It accepts a data frame plus column-name arguments, matching the conventions used elsewhere in the `forestsearch` package, and extracts the covariate matrix, response, and treatment indicator before dispatching to `dina_fit()`.

Usage

```
dina(
  df,
  outcome,
  treatment,
  covariates,
  family = c("gaussian", "binomial", "poisson", "cox"),
  status = NULL,
  ...
)
```

Arguments

<code>df</code>	data frame (or tibble / <code>data.table</code>) containing the outcome, treatment, and covariate columns.
<code>outcome</code>	character(1); the name of the response column in <code>df</code> . For <code>family = "cox"</code> this is the event/censoring time column.
<code>treatment</code>	character(1); the name of the binary treatment indicator column in <code>df</code> . Values must be 0/1 or logical.
<code>covariates</code>	character vector of length ≥ 1 ; the names of the covariate columns in <code>df</code> . All must be numeric.
<code>family</code>	one of "gaussian", "binomial", "poisson", "cox".
<code>status</code>	character(1) or NULL; for <code>family = "cox"</code> only, the name of the event indicator column in <code>df</code> (1 = event, 0 = censored). Ignored for non-Cox families.
<code>...</code>	further arguments forwarded to <code>dina_fit()</code> .

Details

All other arguments are forwarded to `dina_fit()` via `...`, including `propensity_method`, `baseline_method`, `cross_fitting`, `n_folds`, `cens_type`, `cens_params`, `n_grid`, `eps`, and `seed`. See `dina_fit()` for their full documentation.

Covariate columns must be numeric (or integer). Convert factors to dummy variables (e.g., with `stats::model.matrix()`) before calling if needed. Rows with any NA value in the referenced columns are rejected with an error; use `stats::na.omit()` or impute beforehand.

Value

An object of class "dina", as returned by `dina_fit()`. The `$call` component is overwritten with the original `dina()` call so that `print()` and `summary()` show the high-level invocation rather than the internal matrix-based dispatch.

See Also

`dina_fit()` for the underlying matrix-based estimator; `dina_bagged()` for the bagged version with infinitesimal-jackknife variance.

Examples

```
set.seed(1)
n <- 400
df_demo <- data.frame(
  y      = NA,
  w      = stats::rbinom(n, 1, 0.5),
  x1     = stats::runif(n, -1, 1),
  x2     = stats::runif(n, -1, 1),
  x3     = stats::runif(n, -1, 1)
)
tau_x    <- 0.5 + 0.8 * df_demo$x1 - 0.3 * df_demo$x2
df_demo$y <- 1 + df_demo$x1 + df_demo$w * tau_x + stats::rnorm(n)

fit <- dina(
  df          = df_demo,
  outcome     = "y",
  treatment   = "w",
  covariates  = c("x1", "x2", "x3"),
  family      = "gaussian",
  seed        = 1L
)
coef(fit)
confint(fit)
```

dina_bagged

Bagged DINA with IJ variance (data-frame interface)

Description

`dina_bagged()` is the high-level user-facing wrapper around `dina_fit_bagged()`. It accepts a data frame plus column-name arguments and dispatches to `dina_fit_bagged()`, which fits the DINA estimator on `n_bags` bootstrap replicates and returns the infinitesimal-jackknife variance per Wager, Hastie and Efron (2014).

Usage

```
dina_bagged(
  df,
  outcome,
  treatment,
  covariates,
  family = c("gaussian", "binomial", "poisson", "cox"),
  status = NULL,
  ...
)
```

Arguments

df	data frame (or tibble / data.table) containing the outcome, treatment, and covariate columns.
outcome	character(1); the name of the response column in df. For family = "cox" this is the event/censoring time column.
treatment	character(1); the name of the binary treatment indicator column in df. Values must be 0/1 or logical.
covariates	character vector of length ≥ 1 ; the names of the covariate columns in df. All must be numeric.
family	one of "gaussian", "binomial", "poisson", "cox".
status	character(1) or NULL; for family = "cox" only, the name of the event indicator column in df (1 = event, 0 = censored). Ignored for non-Cox families.
...	further arguments forwarded to dina_fit_bagged() .

Details

All bagging-related arguments (n_bags, cross_fitting_per_bag, n_folds_per_bag, parallel, ij_finite_sample_correction, project_psd, verbose, ...) are forwarded via See [dina_fit_bagged\(\)](#) for full documentation.

For parallel = "bags", set `future::plan()` to a parallel backend *before* calling `dina_bagged()`. Restore with `future::plan("sequential")`; `gc()` afterwards.

Value

An object of class `c("dina_bagged", "dina")`, as returned by [dina_fit_bagged\(\)](#). The `$call` component is overwritten with the original `dina_bagged()` call.

See Also

[dina_fit_bagged\(\)](#) for the underlying matrix-based bagged estimator; [dina\(\)](#) for the single-pass DINA with sandwich variance.

Examples

```
set.seed(1)
n <- 400
df_demo <- data.frame(
  y      = NA,
  w      = stats::rbinom(n, 1, 0.5),
  x1     = stats::runif(n, -1, 1),
  x2     = stats::runif(n, -1, 1),
  x3     = stats::runif(n, -1, 1)
)
tau_x    <- 0.5 + 0.8 * df_demo$x1 - 0.3 * df_demo$x2
df_demo$y <- 1 + df_demo$x1 + df_demo$w * tau_x + stats::rnorm(n)

fit <- dina_bagged(
  df      = df_demo,
```

```

outcome = "y",
treatment = "w",
covariates = c("x1", "x2", "x3"),
family = "gaussian",
n_bags = 30L,
seed = 1L
)
coef(fit)
sqrt(diag(vcov(fit)))

```

dina_fit

Estimate heterogeneous treatment effects via DINA

Description

dina_fit() implements the two-step DINA (Difference in NATural parameters) estimator of Gao and Hastie (2025) for heterogeneous treatment effects (HTE) with general response types: Gaussian (mean difference), binomial (log odds ratio), Poisson (log mean ratio), and Cox (log hazard ratio). The treatment effect is assumed to be linear in the covariates,

$$\tau(x) = \beta_0 + x^\top \beta_{1:d},$$

following Proposition 1 of the paper. The procedure has two steps: (1) estimate nuisance functions $a(x)$ and $\nu(x)$ on a training fold, and (2) plug those estimators into a GLM (or Cox) likelihood on the held-out fold to estimate β . Cross-fitting averages the two estimators obtained by swapping the folds.

Usage

```

dina_fit(
  X,
  Y,
  W,
  family = c("gaussian", "binomial", "poisson", "cox"),
  propensity_method = "logistic",
  baseline_method = NULL,
  cross_fitting = TRUE,
  n_folds = 2L,
  cens_type = NULL,
  cens_params = list(),
  n_grid = 500L,
  eps = 1e-06,
  seed = NULL
)

```

Arguments

<code>X</code>	a numeric covariate matrix or data frame with n rows and d columns. All columns are treated as numeric predictors. Column names, when present, are propagated to the coefficient names of the fitted object.
<code>Y</code>	the response. For <code>family = "cox"</code> , a data frame (or matrix coercible to one) with columns <code>time</code> (event/censoring time) and <code>status</code> (1 = event, 0 = censored). For all other families, a numeric vector of length n .
<code>W</code>	binary treatment indicator (0 = control, 1 = treatment), length n .
<code>family</code>	character; one of "gaussian", "binomial", "poisson", "cox".
<code>propensity_method</code>	how to estimate the propensity score $e(x) = P(W = 1 \mid X = x)$. One of: - the string "logistic" (the default): a logistic GLM on X; - a numeric scalar in $(0, 1)$: a known constant propensity (e.g., 0.5 for a balanced randomized trial); - a function taking a covariate matrix and returning a numeric vector of fitted propensities, allowing the user to plug in random forests, boosting, etc.
<code>baseline_method</code>	how to estimate the natural-parameter functions $\eta_0(x)$, $\eta_1(x)$ (or the marginal mean for the Gaussian family). One of: - NULL (the default): use "glm" for non-Cox families and "cox" for the Cox family; "glm": a GLM with the natural link for the family; "cox": <code>survival::coxph()</code> (Cox family only); - a list with components <code>eta0</code> and <code>eta1</code>, each a function taking a covariate matrix and returning a vector of linear predictors. For Gaussian, a list with a single component <code>nu</code> may be supplied instead.
<code>cross_fitting</code>	logical; if TRUE (the default) use <code>n_folds</code> -fold cross-fitting to separate nuisance estimation from target-parameter estimation. If FALSE, the same data are used for both steps.
<code>n_folds</code>	positive integer giving the number of cross-fitting folds (default 2, matching the paper).
<code>cens_type</code>	for <code>family = "cox"</code> , the censoring mechanism. One of NULL (no censoring), "fixed", "uniform", "exponential", or "unknown" (estimate the probability of <i>not</i> being censored from X by logistic regression). Ignored for non-Cox families.
<code>cens_params</code>	named list of censoring parameters. Required entries depend on <code>cens_type</code> : "fixed", "uniform": <code>T0</code> (the administrative censoring time). "exponential": <code>rate_c</code> (the exponential rate). "unknown", NULL: ignored.
<code>n_grid</code>	positive integer giving the number of time-grid points used for numerical integration of the not-censoring probability for <code>cens_type %in% c("uniform", "exponential")</code> (default 500).

eps	small positive constant added to denominators for numerical stability (default 1e-6).
seed	optional integer seed for the random cross-fitting fold assignment. When supplied, the global RNG state is restored on exit.

Details

The Gaussian family uses the simplification of Section 3.2: $a(x) = e(x)$ and $\nu(x) = E[Y \mid X = x]$, so the conditional outcome means $\eta_0(x)$, $\eta_1(x)$ are not required separately. For the binomial and Poisson families, separate within-arm fits to $\eta_0(x)$ and $\eta_1(x)$ are used to construct $a(x)$ via the modified propensity in Eq. (12) of the paper. For the Cox family, $a(x)$ uses the not-censoring probabilities of Eq. (18); how those are estimated is controlled by `cens_type`.

Variance estimation follows the standard sandwich form: $\hat{V}(\hat{\beta}) = \hat{\Sigma}_1^{-1} \hat{\Sigma}_2 \hat{\Sigma}_1^{-1} / n$, where $\hat{\Sigma}_1$ and $\hat{\Sigma}_2$ are the bread (negative score derivative) and meat (outer product of scores) matrices. Under cross-fitting these are averaged across folds before the sandwich is formed. For the Cox step 2 we use the partial-likelihood information-matrix variance returned by `survival::coxph()`.

Value

An object of class "dina", a list with components:

coefficients named numeric vector of estimated DINA coefficients $\hat{\beta}$, with (Intercept) first followed by one entry per covariate.

vcov estimated variance-covariance matrix of $\hat{\beta}$.

family the response family, as supplied.

n, d sample size and number of covariates.

cross_fitting, n_folds cross-fitting settings.

call the matched call.

References

- Gao, Z. and Hastie, T. (2025). Estimating heterogeneous treatment effects for general responses. *Biometrics* 81(4): ujaf162. doi:[10.1093/biometc/ujaf162](https://doi.org/10.1093/biometc/ujaf162).
- Robinson, P. M. (1988). Root-n-consistent semiparametric regression. *Econometrica* 56(4): 931–954.
- Nie, X. and Wager, S. (2021). Quasi-oracle estimation of heterogeneous treatment effects. *Biometrika* 108(2): 299–319.
- Dandl, S., Bender, A. and Hothorn, T. (2024). Heterogeneous treatment effect estimation for observational data using model-based forests. *Statistical Methods in Medical Research* 33(3): 392–413.

Examples

```
set.seed(1)
n <- 400
d <- 3
X <- matrix(stats::runif(n * d, -1, 1), n, d)
colnames(X) <- paste0("x", seq_len(d))
```

```

W <- stats::rbinom(n, 1, stats::plogis(0.5 * X[, 1]))
tau_x <- 0.5 + 0.8 * X[, 1] - 0.3 * X[, 2]
Y <- 1 + X[, 1] + W * tau_x + stats::rnorm(n)
fit <- dina_fit(X, Y, W, family = "gaussian", seed = 1)
coef(fit)
confint(fit)

```

dina_fit_bagged

Bagged DINA with infinitesimal-jackknife variance

Description

`dina_fit_bagged()` fits the DINA estimator on `n_bags` bootstrap replicates of the data and aggregates the resulting per-bag coefficient vectors into a bagged estimator. The variance is estimated by the infinitesimal jackknife (IJ) of Wager, Hastie and Efron (2014), with the finite-sample-corrected form

$$\widehat{V}_{IJ-U} = \widehat{V}_{IJ} - (n/B)\widehat{V}_{boot},$$

where $\widehat{V}_{boot}$ is the empirical covariance of the per-bag coefficients. Negative eigenvalues introduced by the Monte-Carlo correction (an artifact of small `n_bags`) are optionally clipped to zero via PSD projection so that downstream `confint()` is always well-defined.

Usage

```

dina_fit_bagged(
  X,
  Y,
  W,
  family = c("gaussian", "binomial", "poisson", "cox"),
  propensity_method = "logistic",
  baseline_method = NULL,
  n_bags = 100L,
  cross_fitting_per_bag = TRUE,
  n_folds_per_bag = 2L,
  cens_type = NULL,
  cens_params = list(),
  n_grid = 500L,
  ij_finite_sample_correction = TRUE,
  project_psd = TRUE,
  eps = 1e-06,
  seed = NULL,
  parallel = c("none", "bags"),
  verbose = FALSE
)

```

Arguments

<code>X</code>	a numeric covariate matrix or data frame with n rows and d columns. All columns are treated as numeric predictors. Column names, when present, are propagated to the coefficient names of the fitted object.
<code>Y</code>	the response. For family = "cox", a data frame (or matrix coercible to one) with columns time (event/censoring time) and status (1 = event, 0 = censored). For all other families, a numeric vector of length n .
<code>W</code>	binary treatment indicator (0 = control, 1 = treatment), length n .
<code>family</code>	character; one of "gaussian", "binomial", "poisson", "cox".
<code>propensity_method</code>	how to estimate the propensity score $e(x) = P(W = 1 X = x)$. One of: - the string "logistic" (the default): a logistic GLM on X; - a numeric scalar in $(0, 1)$: a known constant propensity (e.g., 0.5 for a balanced randomized trial); - a function taking a covariate matrix and returning a numeric vector of fitted propensities, allowing the user to plug in random forests, boosting, etc.
<code>baseline_method</code>	how to estimate the natural-parameter functions $\eta_0(x)$, $\eta_1(x)$ (or the marginal mean for the Gaussian family). One of: - NULL (the default): use "glm" for non-Cox families and "cox" for the Cox family; "glm": a GLM with the natural link for the family; "cox": <code>survival::coxph()</code> (Cox family only); - a list with components <code>eta0</code> and <code>eta1</code>, each a function taking a covariate matrix and returning a vector of linear predictors. For Gaussian, a list with a single component <code>nu</code> may be supplied instead.
<code>n_bags</code>	positive integer giving the number of bootstrap bags (default 100L). For stable IJ variance estimates the literature recommends <code>n_bags</code> on the order of the sample size n ; smaller values are fine for diagnostic use but produce noisier variance.
<code>cross_fitting_per_bag</code>	logical; if TRUE (the default) each bag uses <code>n_folds_per_bag</code> -fold cross-fitting internally, matching the paper. If FALSE, each bag uses a single in-bag fit (faster but loses the Chernozhukov debiasing step).
<code>n_folds_per_bag</code>	positive integer giving the number of cross-fitting folds within each bag (default 2L). Ignored when <code>cross_fitting_per_bag</code> = FALSE.
<code>cens_type</code>	for family = "cox", the censoring mechanism. One of NULL (no censoring), "fixed", "uniform", "exponential", or "unknown" (estimate the probability of <i>not</i> being censored from X by logistic regression). Ignored for non-Cox families.
<code>cens_params</code>	named list of censoring parameters. Required entries depend on <code>cens_type</code> : "fixed", "uniform": <code>T0</code> (the administrative censoring time). "exponential": <code>rate_c</code> (the exponential rate).

	• "unknown", NULL: ignored.
n_grid	positive integer giving the number of time-grid points used for numerical integration of the not-censoring probability for cens_type %in% c("uniform", "exponential") (default 500).
ij_finite_sample_correction	logical; if TRUE (the default) the IJ variance includes the Monte-Carlo correction $(n/B)\hat{V}_{boot}$. If FALSE the raw IJ formula is returned, which tends to overstate variance for small n_bags.
project_psd	logical; if TRUE (the default) the corrected IJ variance matrix is projected to its nearest positive semi-definite matrix by clipping negative eigenvalues to zero. Strongly recommended unless you want to inspect raw variance values yourself.
eps	small positive constant added to denominators for numerical stability (default 1e-6).
seed	optional integer seed for the random cross-fitting fold assignment. When supplied, the global RNG state is restored on exit.
parallel	character, one of "none" (default) or "bags". When "bags", the per-bag loop is dispatched via <code>foreach::foreach() %dofuture% { ... }</code> , using whatever <code>future::plan()</code> is currently active. Call e.g. <code>future::plan(future::multisession, workers = N)</code> <i>before</i> this function to enable actual parallel execution; otherwise the parallel branch will execute serially with a warning.
verbose	logical; if TRUE, print a one-line progress message every 10 bags. Default FALSE. Per-bag progress messages are only emitted in parallel = "none" mode; in parallel = "bags" mode a single start-of-loop summary is printed if verbose = TRUE.

Details

The implementation calls `dina_fit()` once per bag, with cross-fitting optionally enabled within each bag. All other arguments (`propensity_method`, `baseline_method`, `cens_type`, `cens_params`, `n_grid`, `eps`) are passed through unchanged.

Value

An object of class `c("dina_bagged", "dina")`, a list containing all the components of a "dina" fit (so that the standard S3 methods work via inheritance) plus:

vcov the corrected, PSD-projected IJ variance matrix (or the raw IJ if `project_psd = FALSE`).

vcov_ij_raw the IJ variance without the finite-sample correction.

vcov_ij_corrected the IJ variance with the finite-sample correction, before PSD projection.

vcov_boot the empirical bootstrap covariance of the per-bag coefficients, $\hat{V}_{boot}$.

bag_coefficients `n_bags_used` by `(d + 1)` matrix of per-bag coefficient vectors.

bag_inclusion `n` by `n_bags_used` integer matrix of bag inclusion counts, N_{bi}^* .

n_bags, n_bags_used requested and successfully-fit bag counts.

n_folds_per_bag, cross_fitting_per_bag bag-level cross-fitting settings.

References

- Wager, S., Hastie, T. and Efron, B. (2014). Confidence intervals for random forests: the jackknife and the infinitesimal jackknife. *Journal of Machine Learning Research* 15: 1625–1651.
- Gao, Z. and Hastie, T. (2025). Estimating heterogeneous treatment effects for general responses. *Biometrics* 81(4): ujaf162. doi:[10.1093/biometc/ujaf162](https://doi.org/10.1093/biometc/ujaf162).

See Also

[dina_fit\(\)](#) for the single-pass DINA estimator with sandwich variance.

Examples

```
set.seed(1)
n <- 400
d <- 3
X <- matrix(stats::runif(n * d, -1, 1), n, d)
colnames(X) <- paste0("x", seq_len(d))
W <- stats::rbinom(n, 1, stats::plogis(0.5 * X[, 1]))
tau_x <- 0.5 + 0.8 * X[, 1] - 0.3 * X[, 2]
Y <- 1 + X[, 1] + W * tau_x + stats::rnorm(n)
fit <- dina_fit_bagged(X, Y, W, family = "gaussian",
                      n_bags = 30L, seed = 1L)

coef(fit)
sqrt(diag(vcov(fit)))
```

dina_frontier

Extract DINA per-covariate Pareto frontiers as forestsearch cuts

Description

Runs the same univariate (covariate, direction, threshold) search as [dina_subgroup\(\)](#), but instead of selecting one subgroup it returns a pool of **per-covariate Pareto-non-dominated** cuts. Each cut is rendered as a forestsearch cut expression ("x1 <= 0.5"), so the pool can be fed to [forestsearch\(\)](#) as screening-stage candidates in the same way GRF tree cuts are. This is the DINA analog of GRF candidate discovery.

Usage

```
dina_frontier(
  fit,
  df,
  covariates,
  scope = c("wide", "harm"),
  m_diff = NULL,
  n_min = 60L,
  direction = c("both", "left", "right"),
  max_per_covariate = Inf,
```

```

    max_subgroups = Inf,
    digits = 3L
)

```

Arguments

fit	a fitted DINA object (class "dina" or "dina_bagged").
df	data frame containing the covariate columns.
covariates	character vector of numeric covariate columns to search over.
scope	one of "wide" (default) or "harm". "wide" casts a broad net: every candidate with size $\geq n_{\min}$ is eligible, and the per-covariate frontiers span the whole effect-size boundary (mirroring GRF's wide-net role, leaving forestsearch's consistency stage as the gatekeeper). "harm" restricts the eligible set to candidates with mean tau-hat $\geq m_{\text{diff}}$ before taking the frontiers.
m_diff	scalar harm floor on the link (natural-parameter) scale. Required when scope = "harm"; ignored when scope = "wide". Default NULL.
n_min	positive integer minimum subgroup size. Default 60L.
direction	one of "both" (default), "left", "right".
max_per_covariate	positive integer, or Inf for no limit: the most cuts any single covariate may contribute to the returned table (its top cuts by effect). A report-trimming cap, not a search control. Default Inf – show everything. A finite value that actually trims warns, naming the kept and available counts.
max_subgroups	positive integer, or Inf for no limit: the cap on the total pooled cuts in the returned table . When the pool is larger it is trimmed round-robin by within-covariate effect rank. Default Inf; a finite value that actually trims warns, naming the kept and available counts. Both caps default to Inf for the same reason <code>forestsearch()</code> 's <code>max_subgroups_search</code> does: a finite pooled cap drops covariates whole and truncates the rank-2 pass, so the table stops being a faithful picture of the frontier. The two are still not the same knob – <code>max_subgroups_search</code> truncates the pool of candidate subgroups that forestsearch actually evaluates at the consistency stage, while <code>max_subgroups</code> trims the rows of the table this function returns . One changes what is computed; the other changes what is reported.
digits	significant figures for rounding the emitted threshold in <code>cut_expr</code> . Default 3L.

Details

Why per-covariate and not a single global frontier. The global Pareto frontier in (effect, N) space collapses onto whichever single covariate dominates the trade-off boundary, returning many micro-stepped cuts on that one covariate. That is faithful to the definition but the opposite of what forestsearch needs: its value is in *composing* cuts drawn from several covariates. So this function computes a separate frontier for each covariate, then pools them, mirroring how a GRF tree splits on multiple variables.

Two caps, each optional. Both caps act on **this function's return value** and nothing else: they trim the frontier table of **single cuts** that is returned here. They are not a cap on a selection family

– the rows they trim are univariate cuts, never conjunctions – and `dina_subgroup()` does not take them, so nothing they do reaches the subgroup selector. Either may be `Inf` to disable it:

- `max_per_covariate` bounds how many cuts any single covariate contributes (its top cuts by effect), limiting within-covariate redundancy.
- `max_subgroups` bounds the total pool size across covariates.

Under `forestsearch(subgroup_method = "dina")` neither cap has any effect: that path selects with `dina_subgroup()` and never consults a frontier table. They bite only where this function's output is used directly – including `forestsearch(use_dina = TRUE, dina_args = list(selected_only = FALSE))`, where `cut_expr` becomes the screening-stage candidate pool. When the pool exceeds `max_subgroups`, the global trim is round-robin on within-covariate effect rank – every covariate's best cut first (ordered by effect), then every covariate's second-best, and so on – so a finite budget preserves cross-covariate spread instead of refilling with one covariate's micro-steps. Setting `max_per_covariate = Inf` recovers a pure global budget; setting `max_subgroups = Inf` recovers per-covariate-only limits. Both default to `Inf`, and a finite cap that actually removes rows warns rather than trimming the display silently.

Cut form. The expression is always the canonical "`<covariate> <= <threshold>`". `forestsearch`'s factor machinery exposes both the cut and its complement, so one expression covers both of DINA's left/right directions – the direction column is retained for reference only. Thresholds are rounded to digits significant figures so they deduplicate cleanly against existing GRF / median cuts and read well; raw values remain in threshold.

These cuts are **candidates**, not forced selections: `forestsearch` composes (up to `maxk`) and consistency-gatekeeps them, exactly as it does GRF cuts. To use them without disturbing any user-supplied forced cuts, append rather than overwrite, e.g. `forestsearch(..., conf_force = c(my_forced_cuts, fr$cut_expr))`.

Value

A data frame (one row per cut, after the caps and threshold-rounding deduplication) with columns:

covariate, direction, threshold the subgroup definition (raw threshold).

n_subgroup subgroup size.

mean_tau_hat subgroup-mean tau-hat on the link scale.

effect the same on the natural scale (exponentiated for ratio families binomial/poisson/cox), used for the frontiers and the effect ranking.

cut_expr the canonical `forestsearch` cut expression.

The result carries attributes `n_frontier` (total per-covariate frontier size, deduped, before the caps), `n_qualifying`, and `scope`. Returns a 0-row data frame (same columns) when no candidate qualifies.

See Also

`dina_subgroup()` for the single-subgroup selector; `forestsearch()` for the screening stage that consumes the cuts; `compute_pareto_frontier()` for the post-hoc frontier reporting on a forest-search result.

Examples

```

set.seed(1)
n <- 400
df_demo <- data.frame(
  w = stats::rbinom(n, 1, 0.5),
  x1 = stats::runif(n, -1, 1),
  x2 = stats::runif(n, -1, 1),
  x3 = stats::runif(n, -1, 1)
)
tau_x <- 0.3 + 1.2 * df_demo$x1 - 0.4 * df_demo$x2
df_demo$y <- 0.5 * df_demo$x1 + df_demo$w * tau_x + stats::rnorm(n)

fit <- dina(df_demo, outcome = "y", treatment = "w",
  covariates = c("x1", "x2", "x3"),
  family = "gaussian", seed = 1L)

fr <- dina_frontier(fit, df_demo, covariates = c("x1", "x2", "x3"),
  n_min = 60L)

fr
fr$cut_expr # ready to append to forestsearch(conf_force = ...)

```

dina_subgroup	<i>Identify a harm subgroup from a DINA fit via univariate threshold search</i>
---------------	---

Description

Given a fitted DINA object ("dina" or "dina_bagged") and a data frame containing the covariates the model was fit on, search over (covariate, threshold, direction) candidates for subgroups whose mean per-patient HTE estimate meets the harm threshold `m_diff`, then select one according to `sg_focus` (default "maxSG": the largest qualifying subgroup). This is the simplest bridge between DINA's continuous estimator $\hat{\tau}(x) = \hat{\beta}_0 + x'\hat{\beta}$ and forestsearch's interpretable signature framework.

Usage

```

dina_subgroup(
  fit,
  df,
  covariates,
  m_diff,
  n_min = 60L,
  n_min.frac = 0.1,
  direction = c("both", "left", "right"),
  max_depth = 2L,
  grid_probs = seq(0.1, 0.9, by = 0.1),
  sg_focus = "maxSG",
  selection_rule = "neighborhood",

```

```

    effect_neighborhood = 0.1,
    alpha = 0.05,
    tau_sign = 1
)

```

Arguments

fit	a fitted DINA object of class "dina" or "dina_bagged", as returned by <code>dina_fit()</code> , <code>dina_fit_bagged()</code> , <code>dina()</code> , or <code>dina_bagged()</code> .
df	data frame containing the covariate columns referenced by covariates. Typically the same data frame used to fit fit.
covariates	character vector of column names in df to search over. All must be numeric.
m_diff	scalar harm threshold on the natural-parameter scale of fit\$family. Subgroups are kept only if their mean tau-hat meets or exceeds this value. It is a candidacy floor, not a ranking key. It decides which candidates enter the qualifying family; sg_focus then orders that family. Every focus ranks only over survivors of this floor, so widening or tightening m_diff changes the selection by changing the family, not by changing the sort. Scale. The link (tau) scale of fit\$family, which is not the scale the effect is usually reported on: log(HR) for cox, log(OR) for binomial, log(IRR) for poisson, and the raw mean difference for gaussian. A harm threshold of HR 1.25 is therefore m_diff = log(1.25), not 1.25. Under <code>forestsearch()</code> with subgroup_method = "dina" this argument is derived, not passed through. It is computed as log(hr.threshold) (identity for gaussian) from the resolved effect.threshold, and an m_diff supplied in dina_args – a dina_frontier() key – is ignored on that path. Set the floor with effect.threshold / hr.threshold instead. Under dina_select_statistic = "effect" the floor still defines the qualifying family on the tau-hat scale, while selection re-ranks that family on the refit inferential effect. The two scales therefore do different jobs in one selection: m_diff shapes the family, the refit effect orders it. Since multiplier resampling de-biases over the realized selection event, m_diff is what fixes the family MR de-biases over – changing it changes the inferential target, not just the winner.
n_min	positive integer, or NULL; minimum subgroup size. Default 60L. Supplying a value (or omitting it) uses that fixed floor. Passing n_min = NULL opts into a sample-size-adaptive floor $\max(60, \text{ceiling}(n_{\min.\text{frac}} * n))$, where $n = \text{nrow}(\text{df})$ – at least 60, and at least n_min.frac of n. The resolved integer is stored on the returned object (\$n_min), so <code>dina_subgroup_bootstrap()</code> and <code>dina_subgroup_refit()</code> inherit it.
n_min.frac	numeric in (0, 1); fraction of nrow(df) used for the adaptive n_min floor when n_min = NULL. Default 0.10. Ignored when n_min is supplied. Default 60L, matching the convention in GRF and forestsearch().
direction	one of "both" (default), "left", or "right". For each covariate x_j controls whether to search subgroups of the form $x_j \leq q$ (left), $x_j \geq q$ (right), or both.

max_depth	integer, 1L or 2L (default 2L, aligning with the consistency search maxk = 2 and GRF grf_depth = 2). 1L searches single-covariate cuts only (the legacy behaviour). 2L additionally searches AND-conjunctions of two distinct covariates and ranks them jointly with the single-covariate cuts; see the <i>Depth-2 conjunctions</i> section.
grid_probs	numeric vector of probabilities in $[0, 1]$ giving the per-covariate quantile grid used to generate depth-2 pair thresholds. Only consulted when max_depth = 2L. Default interior deciles seq(0.1, 0.9, 0.1). A finer grid recovers more of the exhaustive pair space at higher cost; covariates with fewer unique values than length(grid_probs) are searched at their unique values instead.
sg_focus	character; subgroup selection criterion applied to the qualifying candidates. One of "maxSG" (default), "minSG", "eff", "effMaxSG", "effMinSG", "maxeff", "maxeffCons". The canonical forms "hr", "hrMaxSG", "hrMinSG" (used internally by <code>forestsearch()</code>) and the alias "maxcons" for "eff" are also accepted. See Description for the semantics of each. Five spellings, one rule. On this path "eff", "hr", "maxcons", "maxeff" and "maxeffCons" all resolve to the same ordering, <code>order(-eff)</code>. Two separate collapses produce that: "eff" and "maxcons" are aliases of "hr" everywhere (see <code>.normalize_sg_focus()</code>), and "maxeff" / "maxeffCons" are behavioural synonyms of "hr" here specifically. Use <code>fs_focus_tag()</code> to label output by the rule that ran; a hand-written tag reading "maxeffCons" on this path names a rule that did not execute. <code>forestsearch()</code> announces the collapse at run time, gated on its quiet argument; <code>dina_subgroup()</code> called directly does not, having no quiet of its own. "maxeff" and "maxeffCons" are synonyms here, both ranking the qualifying candidates by effect (ties broken by insertion order) exactly as "eff" / "hr" does. The Cons suffix names the consistency floor, and DINA has none – it computes no Pcons, and no ordering key below contains one – so there is nothing for the qualifier to bind to. This is engine-specific: under <code>forestsearch()</code> with <code>subgroup_method = "consistency"</code> the two are genuinely distinct, since "maxeffCons" retains both the consistency and effect floors while "maxeff" disables both. The synonymy disables no floor – candidates are qualified before this ordering is applied – and raises no condition, by design.
selection_rule	character; rule defining the candidate inclusion band for "effMaxSG" / "effMinSG". One of "neighborhood" (default; 1-D effect band), "pareto" (2-D non-dominated frontier in (effect, N)), or "both" (intersection). Must be "neighborhood" for the other foci. Forwarded unchanged to the shared band computation, so semantics match <code>forestsearch()</code> .
effect_neighborhood	numeric in $[0, 1]$; relative tolerance for the effect band. A candidate is in the neighborhood iff its (natural-scale) effect is at least $(1 - \text{effect_neighborhood}) * \max(\text{effect})$ over the qualifying set. Default 0.10 (within 10% of the strongest effect). Consulted by <code>selection_rule = "neighborhood"</code> and "both"; ignored by "pareto" (and must be left at its default there) and for "maxSG", "minSG", "eff".
alpha	confidence level for the Wald interval on the subgroup-mean tau-hat. Default 0.05.

tau_sign 1 (default) or -1; the orientation applied to the per-patient tau-hat before the `m_diff` floor, the `sg_focus` ranking and the reported mean and interval. 1 uses the fit's own scale. -1 negates it, for a fit whose harm direction is the negative effect: the floor then reads "harm of at least `m_diff`". `forestsearch()` passes -1 for continuous and binary outcomes with `adverse_outcome = FALSE`, the same outcomes its effect estimator flips ($-Y$, $1 - Y$), so DINA's proposal floor and the admission floor point the same way. With -1, `mean_tau_hat`, `ci` and `candidates$tau_hat` are on the negated scale. The recorded call carries `tau_sign` only when it is not 1.

Details

For each covariate `x_j` in `covariates`, the search visits the sorted unique values of `df[[x_j]]` as candidate thresholds `q` and for each threshold considers the left-tail subgroup of patients with $x_{j,i} \leq q$ (and optionally also the right-tail subgroup with $x_{j,i} \geq q$). A candidate subgroup must satisfy:

- subgroup size $|S| \geq n_{\min}$, AND
- mean per-patient tau-hat over $S \geq m_{\text{diff}}$.

Among the qualifying candidates, `sg_focus` selects the returned subgroup, mirroring `forestsearch()`'s selection vocabulary:

- "maxSG" (default) – largest qualifying subgroup (legacy behaviour; size ties broken by the most extreme mean tau-hat).
- "minSG" – smallest qualifying subgroup.
- "eff" – most extreme effect, ignoring size.
- "effMaxSG" – largest subgroup in the candidate inclusion band (see `selection_rule`).
- "effMinSG" – smallest subgroup in the inclusion band. The "eff*" band foci reuse the inclusion-band logic shared with `sort_subgroups()`; for ratio families (binomial, poisson, cox) the band is applied on the natural (exponentiated) effect scale. `selection_rule` defines the band, exactly as in `forestsearch()`: "neighborhood" (1-D effect band within effect_neighborhood of the maximum qualifying effect), "pareto" (the 2-D non-dominated frontier in (effect, N) space), or "both" (their intersection). DINA's univariate (covariate, threshold, direction) candidates populate the (effect, N) plane just as `forestsearch`'s signatures do, so the frontier is defined analogously.

Value

An object of class "dina_subgroup", a list with components:

found logical; TRUE if any candidate satisfied `m_diff` and `n_min`, otherwise FALSE. When FALSE, the subgroup-specific components below are NULL.

covariate, direction, threshold the chosen cut(s). Length 1 for a single-covariate subgroup; length 2 (one entry per covariate) when `max_depth = 2L` selects a conjunction.

labels character vector of the chosen {var op q} factor label(s), in the form consumed by `get_dfpred()` (AND-composed). Length matches `covariate`.

label a single combined display string, e.g. "{nodes >= 10 & age >= 60}".

depth integer; number of covariates in the selected subgroup (1 or 2).
n_subgroup integer size of the chosen subgroup.
mean_tau_hat scalar subgroup-mean tau-hat (in the tau_sign orientation).
se_mean_tau_hat Wald standard error, computed as $\sqrt{a_S^T \text{vcov}(\text{fit}) a_S}$ (CONDITIONAL on the chosen subgroup – not selection-adjusted).
ci named length-2 vector (lower, upper) giving the Wald 1 - alpha CI.
mask logical vector of length nrow(df) marking which rows are in the chosen subgroup.
m_diff, n_min, sg_focus, selection_rule, effect_neighborhood, alpha, family, max_depth the inputs / fit family, echoed for reproducibility.
n_total nrow(df).
n_candidates_searched total number of (j, dir, q) triples evaluated.
n_candidates_qualifying number of those triples that met both the size and m_diff constraints.
call the matched call.

Depth-2 conjunctions

With max_depth = 2 the candidate set is extended to AND-conjunctions of two thresholds on *distinct* covariates, e.g. nodes >= q1 & age >= q2. Because tau-hat is linear in X, the mean over the intersection mask is scored exactly like a depth-1 cut, so the m_diff, n_min, and sg_focus machinery is unchanged. To stay tractable, depth-2 pairs are generated over a per-covariate quantile grid (grid_probs) rather than every unique value, while depth-1 singletons retain full resolution. Singletons and pairs are ranked *jointly* under sg_focus, so a depth-2 conjunction is returned only when it outranks every depth-1 candidate; otherwise the depth-1 selection is recovered exactly. When the selected subgroup is a conjunction, covariate, direction, and threshold are length-2 vectors and labels holds the two {var op q} factors in the form consumed by [get_dfpred\(\)](#).

The variance of the subgroup-mean tau-hat is computed as

$$\widehat{\text{Var}}(\tilde{\tau}_S) = a_S^T \widehat{\text{Var}}(\hat{\beta}) a_S,$$

with $a_S = (1, \bar{x}_S^T)$ and $\widehat{\text{Var}}(\hat{\beta}) = \text{vcov}(\text{fit})$. The variance source therefore tracks the fit class: sandwich for "dina" and infinitesimal jackknife for "dina_bagged".

Inference caveat

The returned Wald confidence interval is *conditional on the selected (covariate, threshold) being treated as pre-specified*. It does not adjust for the selection across the candidate set and will under-cover when the search space is large. Use this CI for descriptive reporting; for hypothesis testing against m_diff, a selection-adjusted interval via bootstrap of the full search procedure is required. The undercoverage is more pronounced with max_depth = 2L, whose candidate space is substantially larger.

See Also

[dina_fit\(\)](#) / [dina\(\)](#) for the underlying estimator; [dina_fit_bagged\(\)](#) / [dina_bagged\(\)](#) for the bagged variant whose IJ variance propagates through the subgroup-mean SE here; [sort_subgroups\(\)](#) and [forestsearch\(\)](#) for the selection vocabulary mirrored by sg_focus.

Examples

```

set.seed(1)
n <- 400
df_demo <- data.frame(
  w = stats::rbinom(n, 1, 0.5),
  x1 = stats::runif(n, -1, 1),
  x2 = stats::runif(n, -1, 1),
  x3 = stats::runif(n, -1, 1)
)
tau_x <- 0.3 + 1.2 * df_demo$x1 - 0.4 * df_demo$x2
df_demo$y <- 0.5 * df_demo$x1 + df_demo$w * tau_x + stats::rnorm(n)

fit <- dina(df_demo, outcome = "y", treatment = "w",
  covariates = c("x1", "x2", "x3"),
  family = "gaussian", seed = 1L)

sg <- dina_subgroup(fit, df_demo,
  covariates = c("x1", "x2", "x3"),
  m_diff = 0.5, n_min = 60L)

sg

# Tighter subgroup via the effect-neighborhood band:
dina_subgroup(fit, df_demo, covariates = c("x1", "x2", "x3"),
  m_diff = 0.5, n_min = 60L, sg_focus = "effMaxSG")

# Frontier-aware selection (2-D non-dominated set in (effect, N)):
dina_subgroup(fit, df_demo, covariates = c("x1", "x2", "x3"),
  m_diff = 0.5, n_min = 60L, sg_focus = "effMaxSG",
  selection_rule = "both")

# Depth-2: allow AND-conjunctions of two covariates.
dina_subgroup(fit, df_demo, covariates = c("x1", "x2", "x3"),
  m_diff = 0.5, n_min = 60L, max_depth = 2L)

```

dina_subgroup_bootstrap

Selection-adjusted bootstrap CI for dina_subgroup()

Description

Performs selection-adjusted inference for the harm subgroup discovered by `dina_subgroup()`. For each of `n_boot` bootstrap resamples of the input data frame, the function refits DINA via `dina()` and re-runs the subgroup search via `dina_subgroup()`, then reports three classes of output:

Usage

```

dina_subgroup_bootstrap(
  df,

```

```

outcome,
treatment,
covariates,
family = c("gaussian", "binomial", "poisson", "cox"),
status = NULL,
m_diff,
n_min = 60L,
n_min.frac = 0.1,
direction = c("both", "left", "right"),
sg_focus = "maxSG",
effect_neighborhood = 0.1,
alpha = 0.05,
n_boot = 200L,
parallel = c("none", "boots"),
seed = NULL,
verbose = FALSE,
fit = NULL,
sg = NULL,
refit = TRUE,
refit_strata = NULL,
refit_confounders = "none",
...
)

```

Arguments

<code>df</code>	data frame containing the outcome, treatment, and covariate columns.
<code>outcome</code>	character(1); name of the response column in <code>df</code> . For <code>family = "cox"</code> this is the event/censoring time column.
<code>treatment</code>	character(1); name of the binary treatment indicator column.
<code>covariates</code>	character vector; names of the covariate columns to search over. All must be numeric.
<code>family</code>	one of "gaussian", "binomial", "poisson", "cox".
<code>status</code>	character(1) or NULL; for <code>family = "cox"</code> only, the event indicator column name.
<code>m_diff</code>	scalar harm threshold on the natural-parameter scale.
<code>n_min</code>	positive integer, or NULL; minimum subgroup size. Default 60L. Passing <code>n_min = NULL</code> opts into the sample-size-adaptive floor <code>max(60, ceiling(n_min.frac * nrow(df)))</code> , matching dina_subgroup() . When supplying a pre-computed <code>sg</code> , use the same <code>n_min</code> (or <code>n_min.frac</code>) it was found with, since the resolved value must match <code>sg\$n_min</code> .
<code>n_min.frac</code>	numeric in (0, 1); fraction of <code>nrow(df)</code> used for the adaptive <code>n_min</code> floor when <code>n_min = NULL</code> . Default 0.10. Ignored when <code>n_min</code> is supplied.
<code>direction</code>	one of "both" (default), "left", "right".
<code>sg_focus</code>	character; subgroup selection criterion, forwarded to <code>dina_subgroup()</code> for both the original-data point estimate and every bootstrap iteration. Accepts the same set as dina_subgroup() : one of "maxSG" (default), "minSG", "eff", "effMaxSG",

"effMinSG", "maxeff", "maxeffCons" (canonical "hr", "hrMaxSG", "hrMinSG" and the alias "maxcons" also accepted). "maxeff" and "maxeffCons" were previously rejected here even though [dina_subgroup\(\)](#) and [forestsearch\(\)](#) both accept them. On this path all five of "eff", "hr", "maxcons", "maxeff" and "maxeffCons" rank identically – see [dina_subgroup\(\)](#) and [fs_focus_tag\(\)](#).

effect_neighborhood	numeric in $[0, 1]$; relative tolerance for the "effMaxSG" / "effMinSG" effect band, forwarded to dina_subgroup() . Default 0.10. Ignored for the non-band foci.
alpha	confidence level for the percentile CI. Default 0.05 (percentile endpoints at the $\alpha/2$ and $1 - \alpha/2$ quantiles).
n_boot	positive integer; number of bootstrap iterations. Default 200L. Warns if $n_boot < 100$.
parallel	one of "none" (default) or "boots". When "boots", bootstrap iterations dispatch via <code>foreach::foreach() %dofuture%</code> against the active future: <code>plan()</code> .
seed	optional integer seed for reproducibility of the bootstrap indices. Default NULL.
verbose	logical; if TRUE, print a progress message every 10 iterations (serial mode) or a start summary (parallel mode). Default FALSE.
fit	optional precomputed DINA fit ("dina" or "dina_bagged") on the original df. When supplied (and sg is NULL), the function reuses it for the original-data point estimate instead of refitting, then runs only the subgroup search. Its family must match family. Default NULL.
sg	optional precomputed "dina_subgroup" result on the original df. When supplied, it is used directly as the point estimate with no refit and no re-search; fit is then ignored. Its stored m_diff, n_min, alpha, family, and row count must match the corresponding arguments / df, or an error is raised. Supplying the upstream sg is the recommended way to guarantee the bootstrap's point estimate is identical to a standalone dina_subgroup() result. Default NULL.
refit	logical; if TRUE (default), also fit the <i>standard</i> within-subgroup treatment-effect model (Cox for survival, GLM otherwise) via dina_subgroup_refit on the original data, and bootstrap it within the fixed discovered signature. Reported alongside the DINA effect. Skipped (with a warning) if the original-data subgroup is not found or the within-subgroup fit errors.
refit_strata	NULL (default) or a character vector of column names entered as Cox <code>strata()</code> terms in the within-subgroup model. Cox family only.
refit_confounders	adjustment set for the within-subgroup model: "none" (default, unadjusted), NULL (automatic: the DINA covariates with the subgroup-defining covariate omitted), or a character vector of column names. See dina_subgroup_refit .
...	further arguments forwarded to dina() for each per-iteration DINA fit (e.g., <code>n_folds</code> , <code>cens_type</code> , <code>cens_params</code> , <code>propensity_method</code> , <code>baseline_method</code>).

Details

- An **effect CI** on $a^* \wedge T$ $\beta_{a,b}$, where $a^* = (1, \bar{x}_S)$ is held FIXED from the original-data subgroup S^* and $\beta_{a,b}$ is each iteration's DINA coefficient vector. This is the bootstrap

analog of the Wald CI returned by `dina_subgroup()` – conditional on the discovered subgroup definition, with width reflecting sampling variance of $\hat{\beta}$.

- **Stability CIs** on the bootstrap-selected `n_subgroup` and `threshold`, restricted to iterations that selected the same (covariate, direction) as the original-data subgroup.
- A **selection-frequency table** counting how often each (covariate, direction) was chosen. The covariate-selection stability diagnostic.

Each bootstrap iteration calls `dina()` (single-pass, sandwich variance) under the hood, not `dina_bagged()`.

If you have already run `dina()` and `dina_subgroup()` on the original data, pass those objects via `fit` and/or `sg` so the original-data point estimate reuses them rather than being recomputed here. This guarantees the reported point estimate matches your standalone result exactly; otherwise the internal refit uses its own cross-fitting fold assignment, which at a boundary-case `m_diff` can disagree with the upstream fit on whether a subgroup qualifies.

Value

An object of class "dina_subgroup_bootstrap", a list with components:

point the "dina_subgroup" object computed on the original (unbootstrapped) data.

effect_ci named length-2 numeric vector (lower, upper) giving the bootstrap percentile CI on the fixed-subgroup linear functional $a^T \beta_b$, where $a = (1, \bar{x}_S)$ is the covariate-mean vector of the original-data subgroup and β_b is each bootstrap iteration's DINA coefficient vector. This is the meaningful effect-uncertainty CI; width reflects sampling variance of $\hat{\beta}$.

effect_dist numeric vector of length `n_boot` containing $a^T \beta_b$ for each iteration (NA for iterations whose DINA fit failed).

refit the "dina_subgroup_refit" object (standard within-subgroup model on the original data), or NULL if `refit = FALSE`, no subgroup was found, or the fit failed.

refit_effect_ci named length-2 numeric vector (lower, upper); bootstrap percentile CI on the within-subgroup standard-model treatment effect, refit within the FIXED discovered signature on each resample. Conditional on the signature; comparable to `effect_ci`. (NA, NA) if refit was not performed.

refit_effect_dist numeric vector of length `n_boot` of per-iteration within-signature standard-model treatment effects (NA where not computed or the fit failed).

n_subgroup_ci, threshold_ci named length-2 numeric vectors giving percentile CIs for the structural quantities of bootstrap-selected subgroups, restricted to iterations that selected the same (covariate, direction) as the original-data subgroup. Subgroup-size and boundary-location stability diagnostics.

n_modal_match integer; number of bootstrap iterations whose selection matched the original-data (covariate, direction). Denominator for the stability CIs.

selection_frequency a table of (covariate, direction) selection counts across found iterations. Covariate-selection stability diagnostic.

boot_results data frame with one row per bootstrap iteration; columns `found`, `covariate`, `direction`, `threshold`, `n_subgroup`, `mean_tau_hat`, `failed`. The `mean_tau_hat` column records the bootstrap-selected subgroup's boundary value (pinned near `m_diff` by the search rule) and is retained for diagnostic transparency rather than as a meaningful effect estimate.

n_boot, n_boot_found, n_boot_failed convergence diagnostics.

alpha, m_diff, n_min, sg_focus, effect_neighborhood, family echoed inputs.

call the matched call.

Why the bootstrap target is the *fixed-subgroup* effect

A naive bootstrap of `dina_subgroup()`'s `mean_tau_hat` is uninformative. Under the default `sg_focus = "maxSG"` the search selects the *largest* subgroup whose mean tau-hat exceeds `m_diff`, so the chosen subgroup lies at the boundary by construction: adding one more patient would push the mean below `m_diff`. Each bootstrap iteration's selected subgroup is similarly pinned, so the bootstrap distribution of `mean_tau_hat` clusters tightly above `m_diff` and the resulting CI quantifies threshold-grid granularity rather than effect uncertainty. (Under the band foci `"effMaxSG"` / `"effMinSG"` the per-iteration subgroup is instead anchored to the maximum effect rather than the `m_diff` boundary, but the same caveat applies: `mean_tau_hat` is a selection-pinned quantity, not an effect estimate.) The fixed-subgroup linear functional $a^T \beta_b$ instead targets the clinically meaningful estimand "treatment effect for patients whose covariates lie in the discovered subgroup". Because DINA's `beta_hat` is cross-fit (and hence asymptotically unbiased even for data-driven contrasts), the percentile distribution of $a^T \beta_b$ gives a non-parametric CI that complements the sandwich Wald CI of `dina_subgroup()`.

Two complementary effect estimates

When `refit = TRUE` (default), the function reports two treatment effects for the discovered subgroup, on the same natural-parameter scale:

- the **DINA effect** (`effect_ci`): the BLP-analog $a^T \beta$, a projection of the cross-fitted CATE surface averaged over the subgroup; and
- the **within-subgroup standard-model effect** (`refit_effect_ci`): the treatment coefficient from a plain Cox model (or GLM) fit on the subgroup, refit within the FIXED discovered signature on each resample. This is the contrast a clinical-trial reader usually expects.

The two coincide only under correct linearity; reporting both is informative. Crucially, *both* CIs are conditional on the discovered signature treated as pre-specified – neither corrects for the data-driven selection of the signature. A selection-adjusted (de-biased) interval requires the more involved bias-corrected bootstrap of Leon et al. (2024, Section 3.2) and is not provided here; selection *stability* is instead summarized by `selection_frequency`.

Performance

With `parallel = "boots"`, each bootstrap iteration runs in a worker via `foreach::foreach() %dofuture%`. Set `future::plan()` to a parallel backend (e.g., `future::multisession`) before calling, and reset to `"sequential"` afterwards. Workers run from the installed package; `devtools::install()` is required after code edits before the parallel branch will see them.

See Also

`dina_subgroup()` for the underlying threshold search; `dina()` for the per-iteration DINA fit; [dina_subgroup_refit](#) for the standard within-subgroup model reported when `refit = TRUE`.

Examples

```
## Not run:
set.seed(1)
n <- 400
df_demo <- data.frame(
  w = stats::rbinom(n, 1, 0.5),
  x1 = stats::runif(n, -1, 1),
  x2 = stats::runif(n, -1, 1)
)
tau_x <- 0.4 + 1.2 * df_demo$x1
df_demo$y <- 0.5 * df_demo$x1 + df_demo$w * tau_x + stats::rnorm(n)

bs <- dina_subgroup_bootstrap(
  df = df_demo,
  outcome = "y",
  treatment = "w",
  covariates = c("x1", "x2"),
  family = "gaussian",
  m_diff = 0.5,
  n_boot = 50L,
  seed = 1L
)
print(bs)

## End(Not run)
```

dina_subgroup_refit	<i>Standard within-subgroup treatment-effect model for a DINA subgroup</i>
---------------------	--

Description

Given a subgroup discovered by [dina_subgroup](#), fit the *standard* treatment-effect model on the patients in that subgroup: a Cox proportional-hazards model when `sg$family == "cox"`, or a generalized linear model otherwise. The reported effect is the treatment coefficient on the natural-parameter scale (log-HR for Cox, log-OR for binomial, log-IRR for Poisson, mean difference for Gaussian), directly comparable to the DINA effect returned by `dina_subgroup()` but obtained from a conventional within-subgroup regression rather than from the cross-fitted CATE surface.

Usage

```
dina_subgroup_refit(
  sg,
  df,
  treatment,
  outcome,
  covariates,
```

```

    status = NULL,
    strata = NULL,
    confounders = "none",
    alpha = 0.05
  )

```

Arguments

<code>sg</code>	a "dina_subgroup" object from dina_subgroup with <code>sg\$found == TRUE</code> .
<code>df</code>	data frame the subgroup was identified on; must contain the outcome, treatment, covariate, and any strata columns.
<code>treatment</code>	character(1); name of the binary treatment column.
<code>outcome</code>	character(1); name of the response column. For family = "cox" this is the event/censoring time column.
<code>covariates</code>	character vector of the DINA covariate names. Used only to construct the automatic adjustment set when confounders = NULL; ignored otherwise.
<code>status</code>	character(1) or NULL; for Cox only, the event indicator column name. Required when <code>sg\$family == "cox"</code> .
<code>strata</code>	NULL (default) or a character vector of column names entered as Cox <code>strata()</code> terms. Only valid for family = "cox".
<code>confounders</code>	"none" (default, unadjusted), NULL (automatic; see Details), or a character vector of column names.
<code>alpha</code>	confidence level for the Wald interval. Default 0.05 for a 95% CI.

Details

The model is unadjusted by default ($\sim$ treatment); optional covariate adjustment and stratification are available via `confounders` and `strata`.

Value

An object of class "dina_subgroup_refit", a list with components:

- effect** treatment coefficient on the natural-parameter scale (the point estimate).
- se** Wald standard error of effect.
- ci** named length-2 numeric vector (lower, upper); Wald 1 - alpha CI on the natural-parameter scale.
- effect_scale** character label for the scale: one of "log-HR", "log-OR", "log-IRR", "MD".
- ratio_scale** logical; TRUE when effect is a log-ratio (Cox/binomial/Poisson) so that `exp()` maps it to HR/OR/IRR.
- signature** character; the subgroup rule, e.g. "nodes >= 12".
- formula** the fitted model formula.
- confounders_used, strata_used** the resolved adjustment set and stratification columns actually used.
- n_subgroup, n_treated, n_control** subgroup size and per-arm counts.

n_events, n_events_treated, n_events_control event counts within the subgroup (Cox only; NA otherwise).

family, alpha echoed inputs.

model the fitted coxph / glm object.

call the matched call.

Adjustment set (the confounders argument)

Three modes:

- "none" (default) – unadjusted: $\sim$ treatment.
- NULL – automatic: the DINA covariates supplied in `covariates`, with the subgroup-defining covariate (`sg$covariate`) omitted. Omitting it avoids adjusting on a variable whose range is restricted by the subgroup definition.
- a character vector – exactly those columns. A warning is issued if it includes `sg$covariate`.

Inference caveat

The Wald confidence interval treats the subgroup definition as *pre-specified*. It does **not** adjust for the data-driven selection of the signature; it is a within-signature (conditional) interval, not a selection-adjusted one. A selection-adjusted interval requires a bias-corrected bootstrap of the full search procedure (Leon et al. 2024, Section 3.2) and is not provided here.

See Also

[dina_subgroup](#) for the subgroup search and its DINA (BLP-analog) effect; [dina_subgroup_bootstrap](#) for bootstrap inference that reports this within-subgroup model alongside the DINA effect.

Examples

```
## Not run:
fit <- dina(df, outcome = "time", treatment = "trt",
           covariates = c("age", "nodes", "er"),
           family = "cox", status = "status", seed = 1L)
sg <- dina_subgroup(fit, df, covariates = c("age", "nodes", "er"),
                   m_diff = 0, n_min = 60L)

# Unadjusted within-subgroup Cox model (default):
dina_subgroup_refit(sg, df, treatment = "trt", outcome = "time",
                   covariates = c("age", "nodes", "er"),
                   status = "status")

# Adjusted for DINA covariates except the subgroup-defining one:
dina_subgroup_refit(sg, df, treatment = "trt", outcome = "time",
                   covariates = c("age", "nodes", "er"),
                   status = "status", confounders = NULL)

## End(Not run)
```

dummy_encode	<i>Dummy-code a data frame (numeric pass-through, factors expanded)</i>
--------------	---

Description

Dummy-code a data frame (numeric pass-through, factors expanded)

Usage

```
dummy_encode(df)
```

Arguments

df Data frame with numeric and/or factor columns.

Value

Data frame with numeric columns unchanged and factor columns expanded via [acm.disjctif](#).

Examples

```
df <- data.frame(age = c(40, 55, 70), grade = factor(c("1", "2", "3")))
dummy_encode(df)
```

d_eff_binary	<i>Effective Information for Binary Outcomes (Logistic)</i>
--------------	---

Description

Computes $d_{\text{eff}} = n_H \bar{p}(1 - \bar{p})$ from the Fisher information of the logistic regression treatment coefficient.

Usage

```
d_eff_binary(n_sg, p_event)
```

Arguments

n_sg Integer. Subgroup sample size.
p_event Numeric. Event probability (0–1), averaged across arms.

Details

For the model $\text{logit}(p_i) = \beta_0 + \beta V_i$ with balanced arms ($n_1 = n_0 = n_H/2$):

$$\text{Var}(\hat{\beta}) = \frac{4}{n_H \bar{p}(1 - \bar{p})}$$

Value

Numeric. Effective information.

Examples

```
d_eff_binary(n_sg = 100, p_event = 0.30)
# 21
```

d_eff_continuous	<i>Effective Information for Continuous Outcomes (Gaussian)</i>
------------------	---

Description

Computes $d_{\text{eff}} = n_H / \sigma_Y^2$ from the Fisher information of the Gaussian linear model treatment coefficient.

Usage

```
d_eff_continuous(n_sg, sigma_y)
```

Arguments

n_sg	Integer. Subgroup sample size.
sigma_y	Numeric. Residual standard deviation of the outcome.

Details

For the model $Y_i = \beta_0 + \beta V_i + \epsilon_i$, $\epsilon_i \sim N(0, \sigma^2)$, with balanced arms:

$$\text{Var}(\hat{\beta}) = \frac{4\sigma^2}{n_H}$$

Value

Numeric. Effective information.

Examples

```
d_eff_continuous(n_sg = 100, sigma_y = 1.5)
# 44.4
```

d_eff_count

*Effective Information for Count Outcomes (Poisson)***Description**

Computes $d_{\text{eff}} = D/\phi$ where D is the total expected events and ϕ is the dispersion parameter.

Usage

```
d_eff_count(
  n_sg = NULL,
  event_rate = NULL,
  total_events = NULL,
  dispersion = 1
)
```

Arguments

n_sg	Integer. Subgroup sample size. Required if total_events is not provided.
event_rate	Numeric. Mean events per patient. Required if total_events is not provided.
total_events	Numeric. Total observed events D . Overrides $n_sg * event_rate$ if provided.
dispersion	Numeric. Overdispersion parameter (1.0 for Poisson, greater than 1 for quasi-Poisson). Default: 1.0.

Details

For Poisson regression with log-offset:

$$\text{Var}(\hat{\beta}) = \phi \left(\frac{1}{D_0} + \frac{1}{D_1} \right) \approx \frac{4\phi}{D}$$

Under proportional hazards, $D = d$ (Cox events), so $d_{\text{eff}}^{\text{Poisson}} = d_{\text{eff}}^{\text{Cox}}$ (Laird and Olivier, 1981).

Value

Numeric. Effective information.

Examples

```
# From total events
d_eff_count(total_events = 55)

# From sample size and rate
d_eff_count(n_sg = 100, event_rate = 0.8)

# Quasi-Poisson with overdispersion
d_eff_count(total_events = 55, dispersion = 1.5)
```

d_eff_survival	<i>Effective Information for Survival (Cox PH)</i>
----------------	--

Description

Computes $d_{\text{eff}} = n_H(1 - p_c)$ where p_c is the censoring proportion.

Usage

```
d_eff_survival(n_sg, prop_cens)
```

Arguments

n_sg	Integer. Subgroup sample size.
prop_cens	Numeric. Proportion censored (0–1).

Value

Numeric. Effective information (expected events).

Examples

```
d_eff_survival(n_sg = 60, prop_cens = 0.45)
# 33
```

early_stop_decision	<i>Early Stopping Decision</i>
---------------------	--------------------------------

Description

Evaluates whether enough evidence exists to stop early based on confidence interval for consistency proportion.

Usage

```
early_stop_decision(
  n_success,
  n_total,
  threshold,
  conf.level = 0.95,
  min_samples = 20
)
```

Arguments

n_success	Integer. Number of splits meeting consistency.
n_total	Integer. Total number of valid splits.
threshold	Numeric. Target consistency threshold.
conf.level	Numeric. Confidence level for decision (default 0.95).
min_samples	Integer. Minimum samples before allowing early stop.

Value

Character. One of "continue", "pass", or "fail".

Examples

```
early_stop_decision(95, 100, threshold = 0.90)
early_stop_decision(60, 100, threshold = 0.90)
early_stop_decision(10, 15, threshold = 0.90) # below min_samples
```

estimate_propensity_scores

Estimate Propensity Scores

Description

Computes $P(W = 1 \mid X)$ using the requested method, then constructs both stabilized IPTW weights and the Bang & Robins (2005) inverse propensity score (IPS) covariate. The data frame is returned with three new columns: ps_hat (raw PS), sw (stabilized IPTW weights), and ips_covar (IPS covariate).

Usage

```
estimate_propensity_scores(
  data,
  treat.name,
  confounders.name,
  method = c("grf", "lasso", "logistic", "none"),
  grf_forest = NULL,
  seed = 8316951L,
  trim = c(0.025, 0.975)
)
```

Arguments

data	Data frame.
treat.name	Character. Name of binary treatment column (0/1).
confounders.name	Character vector. Covariate names for the propensity model.

method	Character. One of "grf" (default), "lasso", "logistic", or "none".
grf_forest	Optional. A fitted causal_forest object from which to extract W.hat.
seed	Integer. Random seed for GRF.
trim	Numeric vector of length 2. Quantile bounds for PS trimming. Default: c(0.025, 0.975).

Value

A list with:

data	Data frame with ps_hat, sw, and ips_covar columns appended.
ps_hat	Numeric vector of estimated propensity scores.
sw	Numeric vector of stabilized IPTW weights.
ips_covar	Numeric vector of inverse PS covariates (Bang & Robins 2005).
method	Character. Method actually used.
trimmed	Integer. Number of observations trimmed.

References

Bang H, Robins JM (2005). "Doubly Robust Estimation in Missing Data and Causal Inference Models." *Biometrics* 61(4): 962–973.

Hernan MA, Robins JM, Brumback B (2000). "Marginal Structural Models and Causal Inference in Epidemiology." *Epidemiology* 11(5): 550–560.

Examples

```
## Not run:
df <- survival::gbsg
df$grade3 <- as.integer(df$grade == "3")
ps_result <- estimate_propensity_scores(
  data = df,
  treat.name = "hormon",
  confounders.name = c("age", "meno", "size", "grade3", "nodes",
    "pgr", "er"),
  method = "grf"
)
summary(ps_result$ps_hat)
summary(ps_result$sw)

## End(Not run)
```

evaluate_comparison	<i>Evaluate a Comparison Expression Without eval(parse())</i>
---------------------	---

Description

Parses a string of the form "var op value" and evaluates it directly against a data frame column using operator dispatch. Falls back to column-name lookup for bare names.

Usage

```
evaluate_comparison(expr, df)
```

Arguments

expr	Character. An expression like "er <= 0", "size > 35", "grade3 == 1", or a bare column name like "q3.1".
df	Data frame whose columns are referenced by expr.

Details

Supported operators (matched longest-first to avoid partial-match ambiguity): <=, >=, !=, ==, <, >.

If no operator is found, expr is treated as a column name and the result is df[[expr]] == 1.

The value on the right-hand side is coerced to numeric when possible, otherwise kept as character for string comparisons.

Value

Logical vector of length nrow(df).

Examples

```
## Not run:
df <- data.frame(er = c(-1, 0, 1, 2), size = c(10, 20, 30, 40))
evaluate_comparison("er <= 0", df)
# [1] TRUE TRUE FALSE FALSE

evaluate_comparison("size > 25", df)
# [1] FALSE FALSE TRUE TRUE

## End(Not run)
```

evaluate_consistency_twostage

Evaluate Consistency (Two-Stage Algorithm)

Description

Evaluates a single subgroup for consistency using a two-stage approach: Stage 1 screens with fewer splits, Stage 2 uses sequential batched evaluation with early stopping for efficient evaluation.

Usage

```
evaluate_consistency_twostage(
  m,
  index.Z,
  names.Z,
  df,
  found.hrs,
  hr.consistency,
  pconsistency.threshold,
  pconsistency.digits = 2,
  maxk,
  confs_labels,
  details = FALSE,
  n.splits.screen = 30,
  screen.threshold = NULL,
  n.splits.max = 400,
  batch.size = 20,
  conf.level = 0.95,
  min.valid.screen = 10,
  estimator_fn = NULL,
  consistency_threshold = NULL,
  adjust_covariates = NULL,
  consistency_method = "split",
  glm_resample_spec = NULL
)
```

Arguments

<code>m</code>	Integer. Index of subgroup to evaluate.
<code>index.Z</code>	data.table or matrix. Factor indicators for all subgroups.
<code>names.Z</code>	Character vector. Names of factor columns.
<code>df</code>	data.frame. Original data with Y, Event, Treat, id columns.
<code>found.hrs</code>	data.table. Subgroup hazard ratio results.
<code>hr.consistency</code>	Numeric. c2, an effect threshold – the minimum effect required in <i>each</i> half of a split for that split to count as consistent. Not a proportion and not the consistency rate. <code>forestsearch()</code> calls this <code>consistency.threshold/hr.consistency</code> .

<code>pconsistency.threshold</code>	Numeric. p^* , a proportion in $[0, 1]$ – the fraction of splits that must be consistent, where one split’s consistency is decided by <code>hr.consistency (c2)</code> . <code>c2</code> sets the bar; p^* counts how often it is met.
<code>pconsistency.digits</code>	Integer. Number of decimal places to which the consistency proportion is rounded before it is compared with <code>pconsistency.threshold</code> . The rounded value decides admission and is the value the selection sort orders on, so this setting affects which subgroups are admitted, not only what is displayed. At the default of 2, a threshold of p^* admits any candidate whose unrounded proportion is at or above the largest 0.01 grid point strictly below p^* plus half a step (0.005) – so $p^* = 0.99$ admits from about 0.985. Raising the setting makes admission finer-grained and produces fewer exact ties for the selection rule to break. Under <code>consistency_method = "resample"</code> the proportion is a continuous closed-form quantity, so the rounding is a deliberate coarsening rather than a formatting step. Default: 2
<code>maxk</code>	Integer. Maximum number of factors in a subgroup.
<code>confs_labels</code>	Character vector. Labels for confounders.
<code>details</code>	Logical. Print progress details.
<code>n.splits.screen</code>	Integer. Number of splits for Stage 1 (default 30).
<code>screen.threshold</code>	Numeric. Screening threshold for Stage 1 (default auto-calculated).
<code>n.splits.max</code>	Integer. Maximum total splits (default 400).
<code>batch.size</code>	Integer. Splits per batch in Stage 2 (default 20).
<code>conf.level</code>	Numeric. Confidence level for early stopping (default 0.95).
<code>min.valid.screen</code>	Integer. Minimum valid splits in Stage 1 (default 10).
<code>estimator_fn</code>	Closure or NULL. Effect-estimator closure from make_effect_estimator for GLM outcomes. Default NULL (uses Cox model).
<code>consistency_threshold</code>	Numeric or NULL. Threshold for GLM consistency evaluation. Default NULL (uses <code>hr.consistency</code>).
<code>adjust_covariates</code>	Character vector or NULL. Additional Cox model terms (e.g. <code>"strata(x1)"</code>) used when scoring survival subgroups. Referenced columns must be present in <code>df</code> . Ignored on the GLM path. Default NULL (unadjusted).
<code>consistency_method</code>	Character. <code>"split"</code> (default) or <code>"resample"</code> ; see subgroup.consistency . When <code>"resample"</code> engages it bypasses the two-stage split screening (the rate is computed from a single fit); an unsupported/non-convergent GLM rate falls through to the Stage 1/2 splitting.
<code>glm_resample_spec</code>	List or NULL. GLM resampling specification threaded from forestsearch ; see subgroup.consistency . NULL on the survival path.

Value

Named numeric vector with consistency results, or NULL if not met.

Examples

```
## Not run:
# evaluate_consistency_twostage() is called internally by forestsearch()
# when use_twostage = TRUE. Use forestsearch() as the entry point:
fs <- forestsearch(gbsg,
  confounders.name = c("age", "meno", "size", "grade3", "nodes", "pgr", "er"),
  outcome.name = "rfstime", treat.name = "hormon", event.name = "status",
  use_twostage = TRUE)

## End(Not run)
```

evaluate_cuts_once	<i>Cache and validate cut expressions efficiently</i>
--------------------	---

Description

Evaluates all cut expressions once and caches results to avoid redundant evaluation. Much faster than evaluating repeatedly.

Usage

```
evaluate_cuts_once(confs, df, details = FALSE)
```

Arguments

confs	Character vector of cut expressions.
df	Data frame to evaluate expressions against.
details	Logical. Print details during execution.

Details

This replaces multiple `eval(parse())` calls scattered throughout `get_FSdata`. By caching results, we avoid:

1. Repeated parsing of expressions
2. Repeated evaluation on dataframe
3. Redundant uniqueness checks

Value

List with:

- `evaluations`: List of evaluated vectors (logical TRUE/FALSE) for each cut
- `is_valid`: Logical vector indicating which cuts produced >1 unique value
- `has_error`: Logical vector indicating which cuts failed to evaluate

Examples

```
df <- data.frame(age = c(40, 55, 70), size = c(20, 30, 25))
confs <- c("age <= 50", "size > 25")
evaluate_cuts_once(confs, df)
```

evaluate_subgroup_consistency

Evaluate Subgroup Consistency (Single-Stage)

Description

Evaluates a single candidate subgroup for treatment-effect consistency using repeated random data splits (the standard, non-two-stage algorithm). For each split the subgroup effect is scored and compared against the harm threshold; the consistency rate is the proportion of splits meeting it. Called internally by [forestsearch](#) (via [subgroup_consistency](#)); [evaluate_consistency_twostage](#) is the screened, early-stopping counterpart used when `use_twostage = TRUE`.

Usage

```
evaluate_subgroup_consistency(
  m,
  index.Z,
  names.Z,
  df,
  found.hrs,
  n.splits,
  hr.consistency,
  pconsistency.threshold,
  pconsistency.digits = 2,
  maxk,
  confs_labels,
  details = FALSE,
  estimator_fn = NULL,
  consistency_threshold = NULL,
  adjust_covariates = NULL,
  consistency_method = "split",
  glm_resample_spec = NULL,
  skip_consistency = FALSE
)
```

Arguments

<code>m</code>	Integer. Index of subgroup to evaluate.
<code>index.Z</code>	data.table or matrix. Factor indicators for all subgroups.
<code>names.Z</code>	Character vector. Names of factor columns.
<code>df</code>	data.frame. Original data with Y, Event, Treat, id columns.

found.hrs	data.table. Subgroup hazard ratio results.
n.splits	Integer. Number of random splits used to evaluate consistency.
hr.consistency	Numeric. c2, an effect threshold – the minimum effect required in <i>each</i> half of a split for that split to count as consistent. Not a proportion and not the consistency rate. forestsearch() calls this consistency.threshold/hr.consistency.
pconsistency.threshold	Numeric. p*, a proportion in $[0, 1]$ – the fraction of splits that must be consistent, where one split's consistency is decided by hr.consistency (c2). c2 sets the bar; p* counts how often it is met.
pconsistency.digits	Integer. Number of decimal places to which the consistency proportion is rounded before it is compared with pconsistency.threshold. The rounded value decides admission and is the value the selection sort orders on, so this setting affects which subgroups are admitted, not only what is displayed. At the default of 2, a threshold of p* admits any candidate whose unrounded proportion is at or above the largest 0.01 grid point strictly below p* plus half a step (0.005) – so p* = 0.99 admits from about 0.985. Raising the setting makes admission finer-grained and produces fewer exact ties for the selection rule to break. Under consistency_method = "resample" the proportion is a continuous closed-form quantity, so the rounding is a deliberate coarsening rather than a formatting step. Default: 2
maxk	Integer. Maximum number of factors in a subgroup.
confs_labels	Character vector. Labels for confounders.
details	Logical. Print progress details.
estimator_fn	Closure or NULL. Effect-estimator closure from make_effect_estimator for GLM outcomes. Default NULL (uses Cox model).
consistency_threshold	Numeric or NULL. Threshold for GLM consistency evaluation. Default NULL (uses hr.consistency).
adjust_covariates	Character vector or NULL. Additional Cox model terms (e.g. "strata(x1)") used when scoring survival subgroups. Referenced columns must be present in df. Ignored on the GLM path. Default NULL (unadjusted).
consistency_method	Character. "split" (default) or "resample"; see subgroup.consistency . When "resample" engages it bypasses the split evaluation (the rate is computed from a single fit); an unsupported/non-convergent GLM rate falls through to the split evaluation.
glm_resample_spec	List or NULL. GLM resampling specification threaded from forestsearch ; see subgroup.consistency . NULL on the survival path.
skip_consistency	Logical. When TRUE, skip the consistency estimation entirely and return the effect-carry result row (effect from found.hrs, Pcons = NA) without forming the subgroup subset or refitting. Used by the sg_focus = "maxeff" fast path, which needs effects for the argmax but consistency for the selected winner only. Default FALSE.

Value

Named numeric vector with consistency results, or NULL if the subgroup does not meet the criteria.

Examples

```
## Not run:
# evaluate_subgroup_consistency() is called internally by forestsearch().
# Use forestsearch() as the entry point:
fs <- forestsearch(gbsg,
  confounders.name = c("age", "meno", "size", "grade3", "nodes", "pgr", "er"),
  outcome.name = "rfstime", treat.name = "hormon", event.name = "status")

## End(Not run)
```

expand_covariates	<i>Expand a seed covariate frame to n_super rows via a Gaussian copula.</i>
-------------------	---

Description

Expand a seed covariate frame to n_super rows via a Gaussian copula.

Usage

```
expand_covariates(
  seed_df,
  vars,
  n_super,
  seed = 20260903L,
  integer_vars = NULL,
  method = c("copula", "bootstrap")
)
```

Arguments

seed_df	data.frame of the seed trial (only vars are used)
vars	covariates to expand (numeric or 0/1 binary)
n_super	rows to generate
seed	RNG seed
integer_vars	variables to round back to integers (auto-detected if NULL)
method	"copula" (default) or "bootstrap" (row resampling, as the package does by default – no novel covariate patterns)

Value

data.frame with n_super rows and columns vars

 explain_pareto_selection

Describe the Pareto-Frontier Selection in Words

Description

Wraps `pareto_frontier_table` with a verbal account of the selection: why the chosen subgroup satisfies the configured `sg_focus` + `selection_rule` criterion, and, for each non-dominated alternative, on which lexicographic axis it lost to the winner.

Usage

```
explain_pareto_selection(
  fs,
  ci_table = NULL,
  effect_label = NULL,
  digits = 3L,
  verbosity = c("medium", "compact", "detailed"),
  format = c("markdown", "character", "list")
)
```

Arguments

<code>fs</code>	A forestsearch result object.
<code>ci_table</code>	Optional data.table from <code>compute_frontier_cis</code> . When supplied, the Naive 95\ Loser paragraphs do not reference CIs by design – the selection rule does not consider CIs.
<code>effect_label</code>	Optional character override for the effect- measure label in prose, e.g. "Hazard ratio". Defaults to a sensible string for the outcome type.
<code>digits</code>	Integer. Decimal places for effect estimates and bound values in the prose. Default 3L.
<code>verbosity</code>	One of "compact", "medium" (default), or "detailed".
<code>format</code>	One of "markdown" (default), "character", or "list".

Details

The wrapper is **descriptive**, not algorithmic: it reads the frontier and the configuration off the returned `fs` object and renders prose. It does not re-run the selection. A future change to the lexicographic key in `forestsearch` would require updating this wrapper to remain accurate.

Value

Depending on format:

"markdown" A single string with embedded line breaks and bullet markers, ready for `cat()` in a Quarto chunk with `results = "asis"`.

"character" A character vector with one element per logical paragraph or bullet (no markdown markers).

"list" A structured list with named elements criterion, selected, losers, closing, and meta.

Verbosity

"compact" Criterion sentence + selected line + a one-line bullet per loser.

"medium" (**default**) Criterion sentence + selected paragraph (including the Naive CI if ci_table is given) + one short paragraph per loser, each naming the lex-key axis on which the loser was beaten.

"detailed" Medium + a closing paragraph that reproduces the formal lex key, the resolved indicator values, and a note about any ties broken downstream.

Loser ordering

Losers are sorted by "closeness-to-winning", which is operationalised as: **later loss-axis comes first**, then **smaller gap on that axis**. Specifically, a loser that lost on K (the last tiebreaker) is reported before one that lost on N (an early axis), and a loser that beat the winner on every axis until the final one is reported before a loser that fell off the band entirely. This puts the boundary cases first, which is the most informative order for understanding how the rule made its choice.

See Also

[pareto_frontier_table](#), [plot_pareto_frontier](#), [compute_frontier_cis](#).

Examples

```
## Not run:
ci_tab <- compute_frontier_cis(fs, n_splits = 1000, seed = 1)

# Inline in a Quarto chunk
cat(explain_pareto_selection(fs, ci_table = ci_tab))

# Or get the structured object
explain_pareto_selection(fs, ci_table = ci_tab, format = "list")

## End(Not run)
```

extract_grf_args

Recover the GRF arguments used inside a forestsearch run

Description

Returns the exact list of arguments that forestsearch() passed to its inner GRF subgroup-identification call (grf.subg.harm.survival() for survival outcomes, grf.subg.harm.glm() for binary, continuous, and count outcomes). The result is suitable for do.call(grf.subg.harm.glm, args) or do.call(grf.subg.harm.survival, args) to reproduce the standalone GRF run with identical settings.

Usage

```
extract_grf_args(fs)
```

Arguments

`fs` A fitted forestsearch object.

Details

Typical uses:

- Sensitivity analyses: modify one or two arguments and re-fit GRF without re-running the full forestsearch pipeline, e.g. `do.call(grf.subg.harm.glm, modifyList(extract_grf_args(fs), list(maxdepth = 3L)))`.
- Reproducibility audits: confirm what the inner GRF saw.
- Independent variable-importance or policy-tree extraction from a forest fit with the same configuration.

This function reads from `fs$args_call_all` (populated by `forestsearch()` via `mget()` of its `formals`) and from `fs$outcome_type`. It is the user-facing complement to the internal builders that the `forestsearch()` call sites use.

Value

A named list of arguments. For survival outcomes the list targets `grf.subg.harm.survival`; for binary, continuous, or count outcomes it targets `grf.subg.harm.glm`. The list carries an attribute `"grf_function"` holding the target function name as a character string.

See Also

[forestsearch](#), [grf.subg.harm.glm](#), [grf.subg.harm.survival](#).

Examples

```
## Not run:
# Reproduce the inner GRF run
args <- extract_grf_args(fs)
fn   <- match.fun(attr(args, "grf_function"))
grf_standalone <- do.call(fn, args)

# Sensitivity: deeper trees, everything else unchanged
args_deep <- modifyList(args, list(maxdepth = 3L))
grf_deep  <- do.call(fn, args_deep)

## End(Not run)
```

extract_selected_subgroups

Tidy summary of selected subgroups across a comparison run

Description

Reduces a forestsearch_comparison object to a tidy data.frame with one row per combo, summarizing each configuration's selected subgroup. Intended for post-hoc aggregation across analyses (e.g.\ side-by-side summary of a continuous-outcome comparison and a binary-outcome comparison in a single document).

Usage

```
extract_selected_subgroups(x, analysis_label = NULL)
```

Arguments

x A forestsearch_comparison object as returned by [compare_selection_rules](#).
analysis_label Optional character scalar identifying the parent analysis (e.g.\ "continuous" or "binary"). When supplied, it appears as the first column. Useful when row-binding tidy summaries from multiple analyses.

Value

A data.frame with columns:

analysis If analysis_label is supplied; otherwise omitted.
combo Integer index, 1..n_combos.
combo_label Pretty label (e.g.\ "effMaxSG / pareto").
sg_focus, selection_rule The two comparison axes.
outcome_type, effect_measure Recorded per row from fs\$outcome_type / fs\$effect_measure, so mixed-effect-measure runs remain comparable.
subgroup Selected subgroup definition – NA_character_ if no subgroup was identified; otherwise paste(fs\$sg.harm, collapse = " & ").
N_H, N_Hc Sample sizes in the harm subgroup H (treat.recommend == 0) and its complement Hc.
effect The winner's effect estimate on the natural scale (exp applied to the stored hr column for ratio-scale measures OR/RR/IRR; passed through otherwise).
Pcons The winner's consistency probability.
K Number of factors in the winner's subgroup definition.
n_passed Number of candidates that passed consistency (= nrow(out_sg\$result)).
on_frontier Logical – did the selected subgroup lie on the Pareto frontier? NA when the frontier is unavailable.
error The error message for failed combos (NA_character_ on success).

Examples

```
## Not run:
cmp_cont <- compare_selection_rules(...) # continuous analysis
cmp_bin  <- compare_selection_rules(...) # binary analysis

tidy_cont <- extract_selected_subgroups(cmp_cont, analysis_label = "continuous")
tidy_bin  <- extract_selected_subgroups(cmp_bin,  analysis_label = "binary")

all_selected <- rbind(tidy_cont, tidy_bin)
saveRDS(all_selected, "selected_subgroups_all.rds")

## End(Not run)
```

figure_note

Generate Figure Note for Quarto/RMarkdown

Description

Formats the figure note from `plot_sg_weighted_km()` output for use in Quarto or RMarkdown documents.

Usage

```
figure_note(
  x,
  prefix = "*Note*: ",
  include_definition = TRUE,
  include_hr_explanation = TRUE,
  custom_text = NULL
)
```

Arguments

<code>x</code>	Output from <code>plot_sg_weighted_km()</code>
<code>prefix</code>	Character. Prefix for the note. Default uses italic Note.
<code>include_definition</code>	Logical. Include subgroup definition. Default: TRUE
<code>include_hr_explanation</code>	Logical. Include HR(bc) explanation. Default: TRUE
<code>custom_text</code>	Character. Additional custom text to append. Default: NULL

Value

Character string formatted as a figure note, or NULL if no content

Examples

```
## Not run:
km_result <- plot_sg_weighted_km(fs.est = fs)
cat(figure_note(km_result))

## End(Not run)
```

filter_by_lassokeep	<i>Filter a vector by LASSO-selected variables</i>
---------------------	--

Description

Returns elements of `x` that are in `lassokeep`.

Usage

```
filter_by_lassokeep(x, lassokeep)
```

Arguments

<code>x</code>	Character vector.
<code>lassokeep</code>	Character vector of selected variables.

Value

Filtered character vector or `NULL`.

Examples

```
filter_by_lassokeep(c("age", "size", "nodes"), lassokeep = c("age", "nodes"))
filter_by_lassokeep(c("pgr", "er"), lassokeep = c("age", "nodes")) # returns NULL
```

filter_call_args	<i>Filter and merge arguments for function calls</i>
------------------	--

Description

Simplifies the common pattern of filtering arguments from a source list to match a target function's formal parameters, then adding/overriding specific arguments.

Usage

```
filter_call_args(source_args, target_func, override_args = NULL)
```

Arguments

source_args	List of all arguments (typically from <code>mget()</code> or a stored args list).
target_func	Function whose formals define the filter criteria.
override_args	List of arguments to add or override (optional).

Details

This function:

1. Extracts formal parameter names from `target_func`
2. Keeps only arguments from `source_args` that match those names
3. Adds or overrides with any `override_args` provided

Reduces boilerplate and improves readability across the codebase.

Value

List of filtered arguments ready for `do.call()`.

Examples

```
## Not run:
# Instead of:
args_FS <- names(formals(get_FSdata))
args_FS_filtered <- args_call_all[names(args_call_all) %in% args_FS]
args_FS_filtered$df.analysis <- df.analysis
args_FS_filtered$grf_cuts <- grf_cuts
FSdata <- do.call(get_FSdata, args_FS_filtered)

# You now write:
FSdata <- do.call(get_FSdata,
  filter_call_args(args_call_all, get_FSdata,
    list(df.analysis = df.analysis, grf_cuts = grf_cuts)))

## End(Not run)
```

find_k_inter_for_target_hr

Find k_inter Value to Achieve Target Harm Subgroup Hazard Ratio

Description

Uses numerical root-finding to determine the interaction parameter (`k_inter`) that achieves a specified target hazard ratio in the harm subgroup. This is the most efficient method for single target calibration.

Usage

```
find_k_inter_for_target_hr(
  target_hr_harm,
  data,
  continuous_vars,
  factor_vars,
  outcome_var,
  event_var,
  treatment_var,
  subgroup_vars,
  subgroup_cuts,
  k_treat = 1,
  k_inter_range = c(-10, 10),
  tol = 0.001,
  tol_rel = 2.5,
  n_super = 5000,
  verbose = TRUE
)
```

Arguments

<code>target_hr_harm</code>	Numeric value specifying the target hazard ratio for the harm subgroup. Must be positive.
<code>data</code>	A <code>data.frame</code> containing the dataset to use for model fitting.
<code>continuous_vars</code>	Character vector of continuous variable names to be standardized and included as covariates.
<code>factor_vars</code>	Character vector of factor/categorical variable names to be converted to dummy variables.
<code>outcome_var</code>	Character string specifying the name of the outcome/time variable.
<code>event_var</code>	Character string specifying the name of the event/status variable (1 = event, 0 = censored).
<code>treatment_var</code>	Character string specifying the name of the treatment variable.
<code>subgroup_vars</code>	Character vector of variable names defining the subgroup.
<code>subgroup_cuts</code>	Named list of cutpoint specifications for subgroup variables. See generate_aft_dgm_flex for details on flexible specifications.
<code>k_treat</code>	Numeric value for treatment effect modifier. Default is 1 (no modification).
<code>k_inter_range</code>	Numeric vector of length 2 specifying the search range for <code>k_inter</code> . Default is <code>c(-10, 10)</code> .
<code>tol</code>	Numeric value specifying tolerance for root finding convergence. Default is 0.001.
<code>tol_rel</code>	Numeric. Maximum tolerated relative error, in percent; the call stops if the achieved HR is not within <code>tol_rel</code> percent of the target. Default 2.5.
<code>n_super</code>	Integer specifying size of super population for hazard ratio calculation. Default is 5000.
<code>verbose</code>	Logical indicating whether to print progress information. Default is TRUE.

Details

This function uses the uniroot algorithm to solve the equation:

$$HR_{harm}(k_{inter}) - HR_{target} = 0$$

The algorithm typically converges within 5-10 iterations and achieves high precision (within the specified tolerance). If the root-finding fails, the function evaluates the boundaries and provides diagnostic information.

Value

A list of class "k_inter_result" containing:

k_inter Numeric value of optimal k_inter parameter

achieved_hr_harm Numeric value of achieved hazard ratio in harm subgroup

target_hr_harm Numeric value of target hazard ratio (for reference)

error Numeric value of absolute error between achieved and target HR

dgm Object of class "aft_dgm_flex" containing the final DGM

convergence Integer number of iterations to convergence

method Character string "root-finding" indicating method used

See Also

[sensitivity_analysis_k_inter](#) for sensitivity analysis [generate_aft_dgm_flex](#) for DGM generation

Examples

```
## Not run:
gbsg <- survival::gbsg

# Find k_inter for target HR = 2.0 in harm subgroup
result <- find_k_inter_for_target_hr(
  target_hr_harm = 2.0,
  data = gbsg,
  continuous_vars = c("age", "er", "pgr"),
  factor_vars = c("meno", "grade"),
  outcome_var = "rfstime",
  event_var = "status",
  treatment_var = "hormon",
  subgroup_vars = c("er", "meno"),
  subgroup_cuts = list(
    er = list(type = "quantile", value = 0.25),
    meno = 0
  ),
  k_treat = 1.0,
  verbose = TRUE
)
```

```
cat("Optimal k_inter:", result$k_inter, "\n")
cat("Achieved HR:", result$achieved_hr_harm, "\n")

## End(Not run)
```

find_leaf_split

Find the split that leads to a specific leaf node (deprecated)

Description

Deprecated.

Returns only the single split immediately above leaf_node, rendered as "var <= value". This is **not** a correct subgroup definition for a leaf below depth 1: it drops the rest of the root-to-leaf path and mis-renders right-turns (which are >, not <=). Use the path-based `.grf_build_subgroup_definition()` instead, which `grf.subg.harm.survival()` now calls internally. Retained (exported) only for backward compatibility; it is no longer used inside the package and will be removed in a future release.

Usage

```
find_leaf_split(tree, leaf_node)
```

Arguments

tree	Policy tree object
leaf_node	Integer. Leaf node identifier

Value

Character string with a single split expression, or NULL.

See Also

[grf.subg.harm.survival](#) for the standard entry point.

Examples

```
## Not run:
# Deprecated; grf.subg.harm.survival() builds the full path internally.

## End(Not run)
```

find_required_sample_size

Find Minimum Sample Size for Target Detection Power

Description

Determines the minimum subgroup sample size needed to achieve a target detection probability for a given true hazard ratio.

Usage

```
find_required_sample_size(
  theta,
  target_power = 0.8,
  prop_cens = 0.3,
  hr_threshold = 1.25,
  hr_consistency = 1,
  n_range = c(20L, 500L),
  tol = 1,
  verbose = TRUE
)
```

Arguments

theta	Numeric. True hazard ratio in subgroup.
target_power	Numeric. Target detection probability (0-1). Default: 0.80
prop_cens	Numeric. Proportion censored. Default: 0.3
hr_threshold	Numeric. HR threshold. Default: 1.25
hr_consistency	Numeric. HR consistency threshold. Default: 1.0
n_range	Integer vector of length 2. Range of sample sizes to search. Default: c(20, 500)
tol	Numeric. Tolerance for bisection search. Default: 1
verbose	Logical. Print progress. Default: TRUE

Value

A list with:

n_sg_required	Minimum sample size (rounded up)
achieved_power	Actual detection probability at n_sg_required
theta	Input hazard ratio
target_power	Input target power

Examples

```
## Not run:
# Find sample size for 80% power to detect HR = 1.5
result <- find_required_sample_size(
  theta = 1.5,
  target_power = 0.80,
  prop_cens = 0.2
)
print(result)

## End(Not run)
```

fit_causal_forest	<i>Fit causal survival forest</i>
-------------------	-----------------------------------

Description

Wrapper function to fit GRF causal survival forest with appropriate settings

Usage

```
fit_causal_forest(X, Y, W, D, tau.rmst, RCT, seedit, tune_grf = FALSE)
```

Arguments

X	Matrix. Covariate matrix
Y	Numeric vector. Outcome variable
W	Numeric vector. Treatment indicator
D	Numeric vector. Event indicator
tau.rmst	Numeric. Time horizon for RMST
RCT	Logical. Is this RCT data?
seedit	Integer. Random seed
tune_grf	Logical. If TRUE, enables cross-validated hyperparameter tuning via <code>tune.parameters</code> = "all". Default FALSE.

Value

Causal survival forest object

Examples

```
library(survival)
library(grf)
df <- survival::gbsg
X <- as.matrix(df[, c("age", "meno", "size", "nodes", "pgr", "er")])
tau <- quantile(df$rfstime[df$status == 1], 0.6)
cs <- fit_causal_forest(X = X, Y = df$rfstime, W = df$hormon,
                       D = df$status, tau.rmst = tau,
                       RCT = FALSE, seedit = 42L)
```

fit_cox_models	<i>Fit Cox Models for Subgroups</i>
----------------	-------------------------------------

Description

Fits Cox models for two subgroups defined by treatment recommendation.

Usage

```
fit_cox_models(df, formula, treat.name = NULL)
```

Arguments

df	Data frame.
formula	Cox model formula.
treat.name	Character or NULL. Passed to get_Cox_sg to extract the treatment coefficient by name when formula is adjusted for covariates. NULL (default) preserves the treatment-only behaviour.

Value

List with HR and SE for each subgroup.

Examples

```
## Not run:
library(survival)
df <- data.frame(
  tte      = gbsg$rfstime / 30.4375,
  event    = gbsg$status,
  treat    = gbsg$hormon,
  treat.recommend = as.integer(gbsg$er > 0)
)
formula <- build_cox_formula("tte", "event", "treat")
fit_cox_models(df, formula)

## End(Not run)
```

fit_policy_trees	<i>Fit policy trees up to specified depth</i>
------------------	---

Description

Fits policy trees of depths 1 through maxdepth and computes metrics

Usage

```
fit_policy_trees(X, data, dr.scores, maxdepth, n.min)
```

Arguments

X	Matrix. Covariate matrix
data	Data frame. Original data
dr.scores	Matrix. Doubly robust scores
maxdepth	Integer. Maximum tree depth (1-3)
n.min	Integer. Minimum subgroup size

Value

List with trees and combined values

Examples

```
## Not run:  
# fit_policy_trees() is called internally by grf.subg.harm.survival().  
# See grf.subg.harm.survival() for the standard entry point.  
  
## End(Not run)
```

forestsearch	<i>ForestSearch: Exploratory Subgroup Identification</i>
--------------	--

Description

Implements advanced statistical methods for exploratory subgroup identification in clinical trials. Provides tools for identifying patient subgroups with differential treatment effects using machine learning approaches including Generalized Random Forests ('GRF'), LASSO regularization, and exhaustive combinatorial search algorithms. Supports survival endpoints (Cox proportional hazards), binary outcomes (log odds ratio, log relative risk, risk difference), continuous outcomes (mean difference), and count / rate outcomes (log incidence rate ratio via Poisson, quasi-Poisson, or negative-binomial GLMs with optional person-time offset). Features bootstrap bias correction using infinitesimal jackknife methods to address selection bias in post-hoc analyses. Designed for clinical researchers conducting exploratory subgroup analyses in randomized controlled trials, particularly for multi-regional clinical trials ('MRCT') requiring regional consistency evaluation. Methods are described in Leon et al. (2024) [doi:10.1002/sim.10163](https://doi.org/10.1002/sim.10163).

Usage

```
forestsearch(  
  df.analysis,  
  outcome.name = "tte",  
  event.name = "event",  
  treat.name = "treat",  
  id.name = "id",  
  potentialOutcome.name = NULL,  
  flag_harm.name = NULL,  
  confounders.name = NULL,  
  parallel_args = list(plan = "multisession", workers = .default_parallel_workers(),  
    show_message = TRUE),  
  df.predict = NULL,  
  df.test = NULL,  
  is.RCT = TRUE,  
  seedit = 8316951,  
  est.scale = "hr",  
  use_lasso = FALSE,  
  use_grf = FALSE,  
  grf_res = NULL,  
  grf_cuts = NULL,  
  use_dina = FALSE,  
  dina_res = NULL,  
  dina_cuts = NULL,  
  dina_args = list(),  
  dina_select_statistic = c("effect", "dina"),  
  subgroup_method = c("consistency", "dina", "grf"),  
  max_n_confounders = 1000,  
  grf_depth = 2,  
  grf_selection = c("frontier", "tree"),  
  grf_select_statistic = c("effect", "dr"),  
  dmin.grf = 0,  
  frac.tau = 0.8,  
  return_selected_cuts_only = TRUE,  
  conf_force = NULL,  
  defaultcut_names = NULL,  
  cut_type = "default",  
  exclude_cuts = NULL,  
  replace_med_grf = FALSE,  
  cont.cutoff = 4,  
  conf.cont_medians = NULL,  
  conf.cont_medians_force = NULL,  
  conf.cont_jcuts = NULL,  
  collapse_cuts = TRUE,  
  collapse_cuts_args = list(),  
  n.min = 60,  
  n.min.frac = 0.1,  
  effect.threshold = NULL,
```

```

consistency.threshold = NULL,
hr.threshold = 1.25,
hr.consistency = 1,
sg_focus = "hr",
selection_rule = "neighborhood",
effect_neighborhood = 0.1,
fs.splits = 1000,
m1.threshold = Inf,
pconsistency.threshold = 0.9,
stop_threshold = pconsistency.threshold,
pconsistency.digits = 2,
show_candidate_summary = FALSE,
max_print = 10,
d0.min = 10,
d1.min = 10,
max.minutes = 3,
minp = 0.025,
details = FALSE,
quiet = FALSE,
maxk = 2,
by.risk = 12,
plot.sg = FALSE,
plot.grf = FALSE,
max_subgroups_search = Inf,
vi.grf.min = NULL,
use_twostage = TRUE,
twostage_args = list(),
consistency_method = c("resample", "split"),
outcome_type = c("survival", "binary", "continuous", "count"),
effect_measure = NULL,
offset.name = NULL,
adverse_outcome = NULL,
overdispersion = c("none", "quasi", "negbin"),
grf_count_transform = c("log", "identity"),
tune_grf = FALSE,
adjust_covariates = NULL,
ps_method = NULL,
ps_adjust_method = c("none", "iptw", "dr_gcomp"),
ps_hat = NULL,
mr_inference = FALSE,
mr_inference_args = list()
)

```

Arguments

<code>df.analysis</code>	Data frame. Analysis dataset with required columns.
<code>outcome.name</code>	Character. Name of time-to-event outcome variable. Default "tte".
<code>event.name</code>	Character. Name of event indicator (1=event, 0=censored). Default "event".

<code>treat.name</code>	Character. Name of treatment variable (1=treatment, 0=control). Default "treat".
<code>id.name</code>	Character. Name of subject ID variable. Default "id".
<code>potentialOutcome.name</code>	Character. Name of potential outcome variable (optional).
<code>flag_harm.name</code>	Character. Name of true harm flag for simulations (optional).
<code>confounders.name</code>	Character vector. Names of candidate subgroup-defining variables.
<code>parallel_args</code>	List. Parallel processing configuration: plan Character. One of "multisession" (default; cross-platform, reuses workers across calls), "multicore" (Linux/macOS only, fastest startup but fails in RStudio interactive sessions), "callr" (slower per-call, fully isolated R processes), or "sequential" (no parallelism). workers Integer. Number of parallel workers. Default: $\text{floor}(0.80 * N)$ where N is the number of physical (non-hyperthreaded) cores, capped to 2 when <code>_R_CHECK_LIMIT_CORES_</code> is set (e.g., during R CMD check). On a 4-core laptop the default is 3 workers; on a 16-core workstation it is 12. show_message Logical. Show parallel setup messages. batch_size Integer. Number of <i>candidate subgroups</i> dispatched to the workers per batch during consistency evaluation. Default: chosen from the worker count and pool size. This is a performance knob only – it does not affect the selected subgroup. Smaller batches let an early stop take effect sooner (relevant only for <code>sg_focus = "maxeffCons"</code>, the one focus that early-stops); larger batches amortise dispatch overhead. Not to be confused with <code>twostage_args\$batch.size</code>, which batches <i>splits</i> within Stage 2 of the two-stage consistency algorithm – a different quantity in different units. To disable parallelism entirely, pass <code>parallel_args = list(plan = "sequential")</code>.
<code>df.predict</code>	Data frame. Prediction dataset (optional).
<code>df.test</code>	Data frame. Test dataset (optional).
<code>is.RCT</code>	Logical. Is this a randomized controlled trial? Default TRUE.
<code>seedit</code>	Integer. Random seed. Default 8316951.
<code>est.scale</code>	Character. Estimation scale ("hr" or "rmst"). Default "hr".
<code>use_lasso</code>	Logical. Use LASSO for variable selection. Default FALSE. The default changed from TRUE in an earlier release: with <code>subgroup_method = "consistency"</code> a model-based front end makes the candidate family data-dependent, which multiplier resampling (<code>mr_inference = TRUE</code>) requires be fixed. Set TRUE to restore the prognostic-lasso prefilter used in Leon et al. (2024).
<code>use_grf</code>	Logical. Generate additional candidate cuts from a GRF fit: tree-derived cut-points are extracted via <code>grf.subg.harm.survival()</code> / <code>grf.subg.harm.glm()</code> (or taken from <code>grf_res</code> / <code>grf_cuts</code> when supplied) and enter the candidate family through get_FSdata alongside the quartile (and any DINA) cuts. This flag governs candidate-cut generation only: GRF variable-importance ordering of the cut columns is a separate step controlled solely by <code>vi.grf.min</code> and <code>runs</code> , or is skipped, regardless of <code>use_grf</code> . Default FALSE. The default changed from TRUE in an earlier release, for the same fixed-candidate-family reason given under <code>use_lasso</code> .

grf_res	GRF results object (optional, for reuse).
grf_cuts	List. Custom GRF cut points (optional).
use_dina	Logical. Generate additional screening-stage candidate cuts from a DINA per-covariate Pareto frontier, fed into get_FSdata alongside the GRF cuts. Default FALSE (opt-in); existing behaviour is unchanged when FALSE.
dina_res	A fitted DINA object (optional, for reuse). When NULL and use_dina = TRUE, a DINA model is fit at the screening stage.
dina_cuts	Character vector of pre-supplied DINA cut expressions (optional). When supplied, the DINA fit / frontier step is skipped and these are used directly.
dina_args	Named list of DINA tuning options, all optional. Recognised keys: family (defaults to the mapping from outcome_type: survival -> cox, binary -> binomial, continuous -> gaussian, count -> poisson), seed (defaults to seedit), and the fit-only keys n_folds, cens_type, cens_params (passed to dina only when set); plus the frontier keys forwarded to dina_frontier: scope (default "wide"), m_diff, n_min (defaults to the n.min of this call), direction, max_per_covariate, max_subgroups, and digits; plus the screening-behavior keys selected_only (default TRUE, matching GRF's return_selected_cuts_only), max_depth (2L default – aligning DINA with maxk = 2 and grf_depth = 2 so the selected cut may be an AND-conjunction of two covariates; set 1L for single-covariate cuts only), and grid_probs (the per-covariate quantile grid for depth-2 pair thresholds; default interior deciles). max_depth / grid_probs are forwarded to dina_subgroup and apply to both the selected-cut screening and subgroup_method = "dina". (The selector statistic formerly accepted here as select_statistic is now the top-level dina_select_statistic argument; a value supplied inside dina_args is deprecated and overridden with a warning.) When selected_only = TRUE (the default), use_dina screening contributes the cut(s) chosen by dina_subgroup – using this call's sg_focus / selection_rule / effect_neighborhood / n.min / hr.threshold, i.e. the same cut subgroup_method = "dina" would select. Set FALSE to contribute the full frontier candidate set instead. m_diff is the candidacy floor, on the link scale, and is derived rather than taken from this list. It is the harm floor a candidate must clear to enter the qualifying family that sg_focus then ranks – log(HR) for cox, log(OR) for binomial, log(IRR) for poisson, and the raw mean difference for gaussian. Under subgroup_method = "dina" (and under selected_only = TRUE screening) it is computed as log(hr.threshold) – identity for gaussian – from the resolved effect.threshold, and an m_diff supplied here is ignored. Set the floor with effect.threshold / hr.threshold. The key remains meaningful only where the raw dina_frontier candidate set is used directly. The same holds for every frontier key, not only m_diff. All seven – scope, m_diff, n_min, direction, max_per_covariate, max_subgroups and digits – are inert under subgroup_method = "dina", which selects with dina_subgroup and never consults a frontier table. Under use_dina = TRUE screening with selected_only = TRUE (the default) a frontier is computed and then discarded in favour of the selected cut, so six of the seven change nothing; only digits still acts, rounding the threshold in the emitted cut expression. The seven keys govern the candidate pool only under selected_only = FALSE, where dina_frontier's

cut_expr column is the pool. Supplying any of the seven under subgroup_method = "dina" now raises a single warning() naming every offending key.

Under dina_select_statistic = "effect" the floor still defines the qualifying family on the tau-hat scale while selection re-ranks that family on the refit inferential effect; because multiplier resampling de-biases over the realized selection event, m_diff fixes the family MR de-biases over. See [dina_subgroup](#). Unknown keys raise an error.

dina_select_statistic

Character, one of "effect" (default) or "dina", used only when subgroup_method = "dina". The DINA analogue of grf_select_statistic. "dina" ranks DINA's qualifying family on its native subgroup-mean tau-hat (unchanged legacy behaviour). "effect" re-ranks the same family on the inferential effect measure (Cox HR for survival; the resolved GLM effect otherwise), scored with the same per-candidate estimator multiplier resampling (MR) uses, so the realized selection is the exact event MR de-biases (and MR's native-family restriction follows the effect rather than tau-hat). "effect" is the default because only it ranks on the inferential coefficient; "dina" is rejected under mr_inference = TRUE.

subgroup_method

Character, one of "consistency" (default), "dina", or "grf". "consistency" is the standard ForestSearch pipeline (GRF/LASSO screening then the consistency search). "dina" delegates subgroup selection to [dina_subgroup](#) and bypasses GRF, LASSO and the consistency search: a DINA model is fit, a single subgroup is selected using this call's sg_focus / selection_rule / effect_neighborhood / n.min / hr.threshold, and that subgroup is returned as sg.harm. dina_args supplies only the DINA fit tuning in this mode (frontier keys are ignored). "grf" is the GRF analogue: it delegates selection to the GRF subgroup-identification routine ([grf.subg.harm.survival](#) for survival, [grf.subg.harm.glm](#) for GLM outcomes), again bypassing LASSO, DINA and the consistency search, and returns the policy-tree harm leaf as sg.harm. GRF-selection mode is configured from this call's existing GRF arguments (frac.tau, dmin.grf, grf_depth, is.RCT, seedit, and the GLM extras) – there is no separate argument list – so it uses the same settings GRF screening would. For both "dina" and "grf", the existing bootstrap de-biasing applies unchanged, re-fitting and re-selecting on each replicate. The selected subgroup may be a single covariate or, where the method's depth allows (DINA max_depth = 2; GRF grf_depth >= 2), an AND-conjunction.

max_n_confounders

Integer. Maximum number of candidate cut columns retained after GRF variable-importance screening, keeping the highest-importance columns. Default 1000. **Applied only when vi.grf.min is numeric:** the truncation lives inside the variable-importance block, so with vi.grf.min = NULL (the default) this argument has no effect, and a finite value below the number of cut columns triggers a warning saying so.

grf_depth

Integer. GRF tree depth. Default 2.

grf_selection

Character, one of "frontier" (default) or "tree", used only when subgroup_method = "grf". "tree" selects via the GRF policy tree. "frontier" selects from the Pareto frontier of the doubly-robust scores (see [grf.subg.harm.survival](#));

the frontier rule is taken from `sg_focus` – “eff”, “effMaxSG”, “effMinSG”, “maxSG” and “minSG” each map to the aligned frontier rule, as in `dina_subgroup()` – and the relative band from `effect_neighborhood`. “maxeff” and “maxeffCons” map **explicitly** to the plain effect-argmax rule (“eff”), the same rule “hr” takes: GRF computes no Pcons, so the Cons qualifier has nothing to bind to. The mapping is exhaustive over the canonical foci and raises an error rather than falling through, so no focus can silently select under a rule the caller did not request. (Earlier versions had no branch for these two and defaulted them to “effMaxSG” silently; pairing them with `grf_selection = “frontier”` is now well defined.) “frontier” is the default because it enumerates a family that can be ranked on the inferential effect; the policy tree exposes no such family, so “tree” is rejected under `mr_inference = TRUE`.

`grf_select_statistic`

Character, one of “effect” (default) or “dr”, used only when `subgroup_method = “grf”` and `grf_selection = “frontier”`. “dr” selects the frontier winner on GRF’s native doubly-robust score (legacy behaviour). “effect” re-ranks the same DR-candidate frontier on the inferential effect measure (Cox HR for survival; the resolved GLM effect otherwise), scored with the same per-candidate estimator multiplier resampling (MR) uses, so the realized selection is the exact event MR de-biases (and MR’s native-family restriction follows the effect rather than the DR score). “effect” is the default because only it ranks on the inferential coefficient; “dr” is rejected under `mr_inference = TRUE`. Ignored for `grf_selection = “tree”` (the policy-tree leaf is the selection, with no enumerated family to rank).

`dmin.grf`

Numeric. Minimum events for GRF. Default 0.0.

`frac.tau`

Numeric in (0, 1]. Multiplier on the GRF time horizon passed to `grf::causal_survival_forest()`. The effective horizon is `frac.tau * min(max(Y[treated, event]), max(Y[control, event]))`, i.e. `frac.tau` times the smaller of the two arm-specific maximum event times. Values < 1 truncate the horizon inward of that maximum (the conservative choice when events are sparse near the tail). Applies to both the GRF variable-importance pre-screen inside `forestsearch()` and the downstream call to [grf.subg.harm.survival](#). Used only when `outcome_type = “survival”`; ignored for GLM outcomes (binary, continuous, count). Default 0.8.

`return_selected_cuts_only`

Logical. If TRUE (default), GRF returns only cuts from the tree depth that identified the selected subgroup meeting `dmin.grf`. If FALSE returns all cuts from all fitted trees (depths 1 through `grf_depth`). See [grf.subg.harm.survival](#) for details.

`conf_force`

Character vector of forced cut expressions (optional), e.g. `c(“er <= 0”, “pgr <= 10”)`. Literal cut locations: unlike the `conf.cont_*` arguments these do not move with a resample.

`defaultcut_names`

Character vector. Default cut variable names (optional).

`cut_type`

Character. Cut strategy: “default” or “median”. Default “default”. These are the only permitted values – `get_FSdata()` validates against them and errors otherwise.

<code>exclude_cuts</code>	Character vector. Variables to exclude from cutting (optional).
<code>replace_med_grf</code>	Logical. Replace median with GRF cuts. Default FALSE.
<code>cont.cutoff</code>	Integer. Cutoff for continuous vs categorical. Default 4.
<code>conf.cont_medians</code>	Character vector of continuous confounders to cut at their median (optional). Names, not values – the cut location is the median computed from the supplied data, so it is not fixed by this argument and moves with a resample. To pin a cut location, pass a literal cut expression via <code>conf_force</code> .
<code>conf.cont_medians_force</code>	Character vector of additional continuous confounders to force a median cut on (optional). Names, not values, with the same consequence as <code>conf.cont_medians</code> above.
<code>conf.cont_jcuts</code>	Named integer list (each value ≥ 1). Per-variable J-quantile cut override for continuous covariates: for an entry $X = J$, the default cut set for X is replaced by J binary cut points at the $(k/(J+1))$ -th quantiles of X ($k = 1, \dots, J$), defining $J+1$ non-overlapping intervals. Names must be in <code>confounders.name</code> and must not overlap with <code>defaultcut_names</code> or <code>conf.cont_medians_force</code> . J-cut variables bypass LASSO filtering, matching <code>defaultcut_names</code> semantics. Variables not listed retain default behaviour. Forwarded to <code>get_FSdata</code> via <code>filter_call_args()</code> ; see there for full semantics.
<code>collapse_cuts</code>	Logical. If TRUE, collapse near-redundant continuous candidate cuts before the search. Continuous covariates can generate many thresholds that are practically redundant (e.g. "age ≤ 35.7 " and "age ≤ 35 "); when enabled, cuts on the same variable and operator within a per-variable standard-error band are merged to a single rounded-centroid threshold, subject to a membership safety check. Forwarded to <code>get_FSdata</code> ; see <code>collapse_redundant_cuts</code> for the algorithm. Default TRUE. Pass <code>collapse_cuts = FALSE</code> to recover the un-coarsened candidate pool used by earlier package versions.
<code>collapse_cuts_args</code>	List of overrides merged onto the defaults <code>list(c = 1.0, tol = 0.05, digits = 0L)</code> : band multiplier c ($\text{band} = c * \text{sd}(x) / \sqrt{n}$), membership safety tolerance tol (fraction of n when < 1 , absolute count when ≥ 1), and representative rounding digits. Ignored when <code>collapse_cuts = FALSE</code> .
<code>n.min</code>	Integer or NULL. Minimum subgroup size. Default 60. Supplying a value (or omitting the argument) uses that fixed floor, as in prior versions. Passing <code>n.min = NULL</code> opts into a sample-size-adaptive floor $\max(60, \text{ceiling}(n.\text{min.frac} * N))$, where N is the analysis sample size – at least 60, and at least <code>n.min.frac</code> of N . Growing the minimum subgroup size with N curbs false discovery from very small subgroups in large samples. The resolved integer is used everywhere downstream (screening, DINA, GRF, the consistency search, and the bootstrap).
<code>n.min.frac</code>	Numeric in $(0, 1)$. Fraction of the analysis sample size used for the adaptive <code>n.min</code> floor when <code>n.min = NULL</code> . Default 0.10. Ignored when <code>n.min</code> is supplied.
<code>effect.threshold</code>	Numeric or NULL. c_1 , the screening threshold on a candidate subgroup's own effect , estimated on the full sample. A candidate must clear it to enter

the consistency stage. For ratio-scale measures (OR, RR, IRR, HR): on the ratio scale (e.g., 1.5 means $OR \geq 1.5$). For identity-scale measures (RD, IRD, MD): on the identity scale (e.g., 0.07 means $RD \geq 7$ pct points). If NULL (default), falls back to `hr.threshold`. Distinct from `consistency.threshold` (`c2`), which is applied within each split half, and from `pconsistency.threshold` (`p*`), which is a proportion rather than an effect. On the consistency path with a ratio estimand (survival HR, binary "OR"), supplying `c1` and **not** `c2` derives $c2 = 0.80 * c1$ on the ratio scale, with a message; see the *Threshold vocabulary and resolved defaults* section for that rule and for the value this resolves to under each estimand.

`consistency.threshold`

Numeric or NULL. `c2`, **an effect threshold** – **not** a proportion, and **not** the consistency rate. It is applied to the subgroup's effect estimated *within each split half*; a split counts as consistent when both of its halves clear `c2`. Same scale conventions as `effect.threshold`. If NULL (default), falls back to `hr.consistency`. Its companion is `pconsistency.threshold` (`p*`), the *fraction* of splits that must clear `c2`: this argument sets the bar, that one counts how often the bar is met. Both names contain the word "consistency" and they are different quantities. On the consistency path with a ratio estimand (survival HR, binary "OR") `c2` must be **less than or equal to** `c1`; $c2 > c1$ is an error. Left unsupplied there, it is derived as $0.80 * c1$ rather than left at its default. See the *Threshold vocabulary and resolved defaults* section for both rules and for the value this resolves to under each estimand.

`hr.threshold`

Numeric. Legacy name for `effect.threshold` – `c1`, the screening threshold on a candidate subgroup's own effect. Retained for backward compatibility. Default 1.25. When `effect.threshold` is provided, `hr.threshold` is ignored; supplying both at disagreeing values is an error. Carries the same `c2` derivation as `effect.threshold`.

`hr.consistency`

Numeric. Legacy name for `consistency.threshold` – `c2`, the per-split **effect** threshold, not a proportion and not the consistency rate. Retained for backward compatibility. Default 1.0. When `consistency.threshold` is provided, `hr.consistency` is ignored; supplying both at disagreeing values is an error. Carries the same $c2 \leq c1$ requirement as `consistency.threshold`.

`sg_focus`

Character. Subgroup selection focus – *what* is selected, not merely how candidates are sorted. Except for "maxeff", every focus selects among **qualifiers**: candidates whose consistency rate reaches `pconsistency.threshold`. One of:

"hr" (or "eff", "maxcons") Pick the qualifier with the highest consistency (`Pcons`); ties broken by largest effect. **This is consistency-primary with effect only as a tiebreak** – despite the name, it does not maximise the effect. Use "maxeffCons" for that. "maxcons" is the recommended spelling, since it says what the rule does.

"maxeffCons" Pick the qualifier with the largest effect; ties broken by fewest cuts (`K`). The effect maximiser *subject to* the consistency floor – as against "maxeff", which applies no floor. This is the only focus for which early stopping is sound; see `stop_threshold`.

"maxSG" Pick the qualifier with the largest sample size; ties broken by consistency. Not a spelling of "hrMaxSG": it applies no effect band.

"minSG" Pick the qualifier with the smallest sample size; ties broken by consistency. Not a spelling of "hrMinSG".

"hrMaxSG" (or "effMaxSG") Among candidates with effect size within effect_neighborhood of the maximum, pick the one with the *largest* sample size.

"hrMinSG" (or "effMinSG") Among candidates with effect size within effect_neighborhood of the maximum, pick the one with the *smallest* sample size.

"maxeff" The effect maximiser over the enumerated family, with no consistency threshold, effect threshold, early stopping, two-stage screening or candidate-pool truncation. Implements the argmax primitive of Guo and He (2021). Size/event minima (n.min, d0.min, d1.min) still apply as estimability constraints; prevalence and redundancy pruning are relaxed to their most permissive setting (minp -> 0, rmin -> 0) and duplicate removal collapses identical-membership candidates to their fewest-cut representative, so only degenerate or exactly-duplicated candidates are dropped and the argmax is invariant. Because consistency (Pcons) never enters the argmax, it is computed for the SELECTED subgroup only (reported alongside it) rather than for every candidate, avoiding the per-candidate consistency re-fits. Contrast "maxeffCons", which applies the same argmax but only among candidates clearing pconsistency.threshold, and "hr", which ranks by consistency first and uses effect only as a tiebreaker.

The descriptions above are for subgroup_method = "consistency". There, "maxeffCons" retains *both* the consistency floor (pconsistency.threshold) and the effect floor, while "maxeff" disables both – the distinction the Cons suffix names.

For subgroup_method = "dina" and "grf", "maxeff" and "maxeffCons" are **synonyms**, both selecting the effect maximiser over the identifier's qualifiers with the harm floor active. Neither engine has a consistency floor – neither computes Pcons, and no DINA or GRF sort key contains one – so the Cons qualifier has nothing to bind to. The two could in principle have been separated by the effect floor alone; they deliberately are not, because the manuscript keeps the harm threshold in a common form across all three identifiers, and "maxeff" exists for "consistency" as a comparison against the argmax primitive of Guo and He (2021), which has no DINA or GRF counterpart. No floor is disabled by this on any path, and the collapse raises no condition: it is intended behaviour, and a warning would fire on every such run.

Per-engine resolution. Every accepted spelling, and the rule that actually runs for it on each engine:

sg_focus	consistency	dina / grf
eff, hr, maxcons	consistency argmax (Pcons-primary, effect breaks ties)	effect argmax, order(-eff)
maxeff	effect argmax, no floors	effect argmax, order(-eff)
maxeffCons	effect argmax with the consistency floor	effect argmax, order(-eff)
effMaxSG, hrMaxSG	largest N within the effect band, per selection_rule	as documented in dina_subgroup
effMinSG, hrMinSG	smallest N within the effect band, per selection_rule	as documented in dina_subgroup
maxSG	largest N among qualifiers	largest N
minSG	smallest N among qualifiers	smallest N

The collapse sets, stated explicitly. On "consistency", {eff, hr, maxcons} is one rule, and maxeff and maxeffCons are two further *distinct* rules. On "dina" and "grf", {eff, hr, maxcons, maxeff, maxeffCons} is **one** rule – five spellings, one order(-eff).

forestsearch() announces both collapses at run time: when the canonical rule differs from the spelling passed, it emits one message() (suppressed by quiet) naming the rule that will actually run. `fs_focus_tag` returns the stem tag for each cell of the table above, and is the supported way to label output files by the rule that ran rather than by the spelling requested.

Default "hr". The "eff*" forms are aliases for the "hr*" forms and read more naturally in GLM contexts (continuous MD, binary OR/RR/RD, count IRR) where there is no hazard ratio. Both vocabularies produce identical results; pick whichever fits the outcome type. See `sort_subgroups` for full sort keys and tiebreakers.

Scope of the band arguments. `selection_rule` and `effect_neighborhood` configure the effect band, which only "hrMaxSG" / "hrMinSG" (equivalently "effMaxSG" / "effMinSG") consult. They are inert for every other focus, and supplying `selection_rule` other than "neighborhood" with a non-band focus is an error rather than a silent no-op.

Choosing between the three effect-oriented rules. "maxeff" answers "what is the most extreme subgroup in this family?"; "maxeffCons" answers "what is the most extreme subgroup that also replicates across splits?"; "hr"/"maxcons" answers "which subgroup replicates most reliably?". They can and do select different subgroups on the same data.

<code>selection_rule</code>	Character. Rule defining the candidate inclusion set for "hrMaxSG" / "hrMinSG". One of "neighborhood" (default; current behaviour), "pareto" (2D Pareto-non-dominated set in (effect, N)), or "both" (intersection of the two). Forwarded to <code>sort_subgroups</code> via <code>filter_call_args()</code> ; see there for full semantics. Must be "neighborhood" for other <code>sg_focus</code> values.
<code>effect_neighborhood</code>	Numeric in $[0, 1)$. Relative tolerance for "hrMaxSG" and "hrMinSG" selection. A candidate is in the <i>effect-size neighborhood</i> iff its (natural-scale) effect is at least $(1 - \text{effect_neighborhood}) * \max(\text{effect})$. Default 0.10 (within 10\ <code>effect_neighborhood</code> = 0 reduces "hrMaxSG" / "hrMinSG" to a strict max-effect filter (only candidates tied at the maximum qualify). For GLM ratio measures (OR, IRR), the neighborhood test is applied on the natural scale. Ignored when <code>sg_focus</code> is "hr", "maxSG", or "minSG".
<code>fs.splits</code>	Integer. Number of splits for consistency evaluation (or maximum splits when <code>use_twostage</code> = TRUE). Default 1000.
<code>m1.threshold</code>	Numeric. Maximum median survival threshold. Default Inf.
<code>pconsistency.threshold</code>	Numeric. p^* , a proportion in $[0, 1]$ – the fraction of random splits that must be consistent, where a split is consistent when both of its halves clear <code>consistency.threshold</code> (c2). It is a rate, not an effect threshold: it never enters an effect comparison, and unlike <code>c1</code> and <code>c2</code> it is not remapped per estimand – the same number means the same thing for every <code>outcome_type</code> . Default 0.90. Overridden to 0 under <code>sg_focus</code> = "maxeff" (no consistency filter).

stop_threshold Numeric in $[0, 1]$ or NULL. Early stopping threshold for consistency evaluation. When a candidate subgroup's consistency probability (Pcons) meets or exceeds this threshold, evaluation stops early – remaining candidates are skipped. Set to NULL to disable early stopping and force evaluation of all candidates up to `max_subgroups_search`. Defaults to `pconsistency.threshold`, so it tracks the consistency floor automatically if that is changed.

Applies to `sg_focus = "maxeffCons"` **only**. Early stopping halts the scan and then applies the focus's selection key to the resulting prefix, so it is sound only when that key is fully determined by the enumeration order. Pcons is not – it is unknown until a candidate is evaluated – so any focus whose key contains a Pcons term can be beaten by a candidate the scan never reached. "maxeffCons" is the only focus with no such term: its key $(-hr, K)$ is exactly the enumeration order $(-HR, K)$, ties included, so the first qualifying candidate is provably the winner.

For every other focus `stop_threshold` is reset to NULL, with a warning if it was supplied explicitly:

- "hr" ("eff", "maxcons") selects on Pcons directly;
- "maxSG" / "minSG" use Pcons to break ties in subgroup size, which a truncated scan can miss;
- "hrMaxSG" / "hrMinSG" additionally need the whole pool to compute the effect neighborhood;
- "maxeff" evaluates only its winner, so early stopping has nothing to skip.

Note: Values > 1.0 are not permitted. To disable early stopping, use `stop_threshold = NULL`, not a value above 1.

pconsistency.digits

Integer. Number of decimal places to which the consistency proportion is rounded **before** it is compared with `pconsistency.threshold`. The rounded value decides admission and is the value the selection sort orders on, so this setting affects which subgroups are admitted, not only what is displayed. At the default of 2, a threshold of p^* admits any candidate whose unrounded proportion is at or above the largest 0.01 grid point strictly below p^* plus half a step (0.005) – so $p^* = 0.99$ admits from about 0.985. Raising the setting makes admission finer-grained and produces fewer exact ties for the selection rule to break. Under `consistency_method = "resample"` the proportion is a continuous closed-form quantity, so the rounding is a deliberate coarsening rather than a formatting step. Passed to [subgroup.consistency](#). Default 2.

show_candidate_summary

Logical. If TRUE, prints a post-consistency summary table of all passing candidates with Frontier/InBand/Selected flags. See [subgroup.consistency](#) for details. Default FALSE.

max_print

Integer. Maximum number of candidate subgroups to print in the pre- and post-consistency candidate summaries when `show_candidate_summary = TRUE`. Candidates are printed in sorted order with the selected subgroup first, so `max_print = 10` shows the selected subgroup and the next nine, followed by a "... N more not shown" line. Only the per-candidate rows are capped; the banner, header, legend, and summary footer are always printed. Default 10 (the selected subgroup and the next nine); use Inf to print all candidates.

d0.min	Integer. Minimum per-arm filter for candidate subgroups. For "survival" and "binary" outcomes, this is the minimum number of events in the control arm (for binary, events are $Y = 1$). Ignored for "continuous" outcomes (only n.min applies). Default 10.
d1.min	Integer. Same as d0.min for the treatment arm. Default 10.
max.minutes	Numeric. Currently inert; scheduled for deprecation in v0.3.0. Previously intended as a wall-clock time budget for the combination search, this argument is no longer enforced in the parallelized search path and has no effect on behavior. Search scope is governed by maxk, the candidate-factor front ends (use_grf, use_lasso, conf_force), and the number of parallel workers. The default of 3 is retained only for signature compatibility.
minp	Numeric. Minimum prevalence threshold. Default 0.025.
details	Logical. Print progress details. Default FALSE.
quiet	Logical. If TRUE, suppress the configuration summary message printed at startup. Useful when forestsearch() is called repeatedly inside bootstrap or cross-validation loops. Default FALSE.
maxk	Integer. Maximum number of factors per subgroup. Default 2.
by.risk	Integer. Risk table interval. Default 12.
plot.sg	Logical. Plot subgroup survival curves. Default FALSE.
plot.grf	Logical. Plot GRF results. Default FALSE.
max_subgroups_search	Numeric. Maximum subgroups to evaluate. Default Inf – no truncation. The cap has no counterpart in the method: the candidate family is every conjunction of at most two conditions, less empty intersections and size/event failures, and selection ranges over that family. A finite value truncates the pool under a <i>pre-view</i> ordering, so the subgroup that is optimal under the actual criterion can be discarded before it is ever evaluated – most acutely for the size foci "maxSG" / "minSG", where small high-effect subgroups sort last, and for the band foci "effMaxSG" / "effMinSG". When a finite value does truncate the pool a warning naming both counts is emitted. Set a finite value only to bound runtime, knowing it can change which subgroup is selected. The default was 200 in earlier versions.
vi.grf.min	Numeric or NULL. GRF variable-importance screening of the candidate cut columns. NULL (the default since 0.3.0) skips the screening entirely: no causal forest is fitted and the cut columns are searched in the order get_FSdata() builds them. A numeric value fits a causal forest (grf::causal_forest() for GLM outcomes, grf::causal_survival_forest() for survival), orders the cut columns by variable importance, and retains those with $vi_ratio > vi.grf.min$, where vi_ratio is each column's importance divided by the maximum. At values ≤ 0 nothing is filtered out – $grf::variable_importance()$ is non-negative and the block is guarded by $vi_max > 0$ – so the effect is ordering only. That ordering sets the candidate matrix's column order; it does not reach candidate selection, because the candidate table is re-sorted by $(-HR, K)$ before the consistency loop. Measured effect on n_candidates_evaluated and early_stop_candidate: zero; the selected subgroup's subject membership is unchanged and only the

clause order of `sg.harm` may differ. Versions before 0.3.0 defaulted to `-0.2`; pass `vi.grf.min = -0.2` explicitly to reproduce their output. `max_n_confounders` is applied **only when this screening runs** – see that argument.

<code>use_twostage</code>	Logical. Use two-stage sequential consistency algorithm for improved performance. Default FALSE for backward compatibility. When TRUE, <code>fs.splits</code> becomes the maximum number of splits, and early stopping is enabled. See Details. Overridden to FALSE under <code>sg_focus = "maxeff"</code> (no two-stage screening).
<code>twostage_args</code>	List. Parameters for two-stage algorithm (only used when <code>use_twostage = TRUE</code>): <code>n.splits.screen</code> Integer. Splits for Stage 1 screening. Default 30. <code>screen.threshold</code> Numeric. Consistency threshold for Stage 1. Default is automatically calculated to provide ~2.5 SE margin. <code>batch.size</code> Integer. Splits per batch in Stage 2. Default 20. <code>conf.level</code> Numeric. Confidence level for early stopping. Default 0.95. <code>min.valid.screen</code> Integer. Minimum valid Stage 1 splits. Default 10.
<code>consistency_method</code>	Character. "resample" (default) uses the multiplier (influence-function / dfbeta) approximation (consistency_resample), returning the rate from a single subgroup fit; "split" is retained as the fallback and runs the literal repeated 50/50 split-and-refit calculation, which costs <code>fs.splits * 2</code> refits per candidate evaluation. Despite the name, "resample" performs no numerical resampling at evaluation time – the rate is a closed-form expression. The name reflects the method's derivation: the repeated-split consistency rate admits an equivalent multiplier-resampling (MR) representation, and taking that representation to its analytic limit yields the closed form implemented here. The "resample" path applies to the Cox (survival) outcome and to GLM outcomes whose effect is a single model coefficient (OR, RR, RD, MD, IRR); configurations it cannot represent that way (IRD, propensity-adjusted effects, or a non-convergent fit on a given candidate) fall back to "split" automatically. When <code>use_twostage = TRUE</code> , "resample" bypasses the two-stage split screening entirely (the rate is computed directly).
<code>outcome_type</code>	Character. One of "survival" (default), "binary", "continuous", or "count".
<code>effect_measure</code>	Character or NULL. Effect measure for GLM outcomes. For binary: "OR" (default), "RD", "RR", "IRR", "IRD". For continuous: "MD" (default). For count: "IRR" (default), "IRD". NULL resolves to the per-outcome default shown here; the resolved measure then fixes the scale on which <code>effect.threshold</code> and <code>consistency.threshold</code> are compared – log for the ratio measures ("OR", "RR", "IRR"), identity for "RD", "IRD" and "MD" – so an unset binary call screens on $\log(1.25)$ rather than 0.05. See the <i>Threshold vocabulary and resolved defaults</i> section. Ignored for <code>outcome_type = "survival"</code> , which always uses the hazard ratio.
<code>offset.name</code>	Character or NULL. Name of the follow-up time column for rate-based measures (IRR, IRD).

adverse_outcome	Logical or NULL. If TRUE, higher outcome values indicate harm (default for binary and count). If FALSE, higher values indicate benefit (default for continuous).
overdispersion	Character. Overdispersion correction for outcome_type = "count". One of "none" (default, standard Poisson), "quasi" (quasi-Poisson), or "negbin" (negative binomial via MASS::glm.nb). Ignored for other outcome types.
grf_count_transform	Character. Transformation applied to the count outcome before passing to grf::causal_forest() for factor screening. "log" (default) applies $\log(Y + 0.5)$; "identity" passes raw counts. Ignored for non-count outcomes.
tune_grf	Logical. If TRUE, enables cross-validated hyperparameter tuning for GRF's causal_forest() via tune.parameters = "all". If FALSE (default), all GRF parameters remain at their defaults, preserving existing behavior. Applies to both internal GRF pre-screening and standalone GRF analysis when passed through run_simulation_analysis().
adjust_covariates	Character vector or NULL. Additional terms used to adjust the Cox model that scores survival subgroups during consistency evaluation. Terms are appended verbatim to the model right-hand side after the treatment term, so survival modelling helpers may be used directly: adjust_covariates = "strata(x1)" fits a model stratified by x1, whereas adjust_covariates = "x1" includes x1 as a linear covariate. The referenced raw columns must be present in df.analysis; they are carried into the internal scoring frame automatically. Applies to the survival (Cox) consistency path and to GLM outcomes (the effect-estimator closure adds the terms to the outcome model, e.g. logistic for binary/OR). The candidate-search ranking is adjusted on both paths. Mutually exclusive with ps_method != "none". Default NULL (treatment-only, unadjusted – the previous behaviour).
ps_method	Character or NULL. Propensity score estimation method: "grf", "lasso", "logistic", or "none". Default: "none" for RCT, "grf" for observational. GLM outcome types only ("binary", "continuous", "count"). Propensity adjustment is not implemented for outcome_type = "survival": no Cox estimation path reads the resulting weights, so any value other than "none" is an error there rather than a silent no-op. For confounding control on a survival outcome use adjust_covariates. Note the default resolves to "grf" when is.RCT = FALSE, so an observational survival analysis must pass ps_method = "none" explicitly.
ps_adjust_method	Character. PS adjustment method: "none" (default), "iptw" (stabilized inverse probability of treatment weights), or "dr_gcomp" (IPS covariate). GLM outcome types only, and consumed only when ps_method is not "none" – it selects how the estimator closure is rebuilt, and that rebuild is GLM-gated. It has no effect on a survival outcome, which is refused at ps_method.
ps_hat	Numeric vector or NULL. User-supplied propensity scores. Must have length equal to nrow(df.analysis), and is applied positionally to df.analysis.

Not carried into bootstrap replicates or cross-validation folds: a resample has the same row count but different subjects, so a supplied score would be attached to the wrong people. Both re-estimate their own score per replicate/fold under `ps_method`.

- `mr_inference` Logical. Whether to follow the search with multiplier resampling (MR): a refit-free de-biased estimate of the selected subgroup's treatment effect, an interval for it, and a harm-confirmation flag. **Default** FALSE, so by default this function performs *identification only* – no de-biasing, no harm confirmation, and `mr_inference` / `mr_harm_confirmed` come back NULL / NA. Setting it TRUE adds a step that runs *after* the subgroup has been chosen and cannot change that choice. Supported on the GLM resample-consistency path (`consistency_method` = "resample" with a GLM effect) and on the survival/Cox path (`outcome_type` = "survival"), where it runs under the default split-consistency search. MR approximates the full bootstrap ([forestsearch_bootstrap_dofuture](#)) to leading order and does not replace it. See the vocabulary section.
- `mr_inference_args` List of optional MR controls; used only when `mr_inference` = TRUE.
- `confirm_rule` Which harm-confirmation rule to apply to the de-biased estimate: "point" (default) compares the de-biased point estimate to `t_confirm`; "ci" compares the one-sided 95% selection-adjusted lower bound, and is therefore the stricter requirement.
- `t_confirm` The threshold `confirm_rule` compares against, on the **effect** scale (not the working log scale). NULL (default) resolves to the null effect: 1 for ratio measures, 0 for differences. Set near the null rather than at the screening threshold, because the correction over-shrinks true effects.
- `reselection` Which selection rule to replay on each perturbed draw when forming the bias term. Defaults to the rule `sg_focus` implies, so it normally needs no setting.
- `draws` Number of multiplier draws. Default 2000.
- `multiplier` Multiplier distribution, default "poisson" (centred Poisson(1) weights, the multiplier-bootstrap analogue of nonparametric-bootstrap multiplicities); also "gaussian", "rademacher".
- `ci_method` "field" (default) additionally runs the field-calibrated block, whose one-sided bounds are the primary reported products; the de-biased element, its infinitesimal-jackknife SE and its IJ intervals are returned in every call regardless, so nothing is removed – only a per-fit Monte Carlo cost is added. "ij" restores the previous default: the de-biased interval from the infinitesimal-jackknife variance, the leading-order analogue of the full-bootstrap interval, with the field block omitted. "wald" uses the subgroup robust SE and also omits the field block. The naive interval always uses the robust SE.
- `include_complement` Also de-bias the complement subgroup. Passed as TRUE by every internal caller.
- `seed` Seed for the multiplier draws; defaults to `seedit`.
- `return_reselection` Logical, default FALSE. When TRUE the gate's return gains the reselection element (per-draw winners and the re-selection frequen-

cies p_{hat}); add-only pass-through, the default reproduces prior output exactly.

field_M_cap Optional override of the uniform (kappa) sweep's mass-carrying cap; NULL (default) keeps `fs_mr_field_uniform()`'s default. Consulted only when `field_uniform = TRUE`.

field_complement Logical, default FALSE. Under `ci_method = "field"` with `include_complement = TRUE`, also evaluate the field block for the complement subgroup, attached as `field$complement` with the one-sided 95% bound as its primary product (the benefit-claim orientation; see [fs_mr_inference](#)). Add-only pass-through: the default reproduces prior output exactly.

ij_residual "two_term" (default), "winner" or "winner_floor": which infinitesimal-jackknife residual populates the reported de-biased SE and interval (both blocks); all three SEs are returned side by side regardless (see [fs_mr_inference](#)). Add-only pass-through: the default reproduces prior output exactly.

keep_declaration_field, keep_field_matrix Logical, both default FALSE. Retain the standardized perturbation field that [fs_declaration_calibration](#) reads, as `mr_inference$declaration_field`; `keep_field_matrix` additionally keeps the full draws-by-candidates matrix (needed for `family = "reduced"`). Add-only pass-through: unset, nothing is computed or stored and the default reproduces prior output exactly.

declaration_c0 NULL (default) or a numeric vector of pre-specified protected null levels c_0 on the natural scale of `hr.consistency` ($c_0 \leq c_2$); consulted only with `keep_declaration_field = TRUE`. The capture then also stores the shifted family maxima `Mstar_c0` that [fs_declaration_calibration\(fit, c0 = \)](#) reads (see [fs_mr_inference](#)). Add-only pass-through: unset, nothing is computed or stored.

Details

Algorithm Overview:

1. **Variable Selection:** GRF identifies variables with treatment effect heterogeneity; LASSO selects most predictive
2. **Subgroup Discovery:** Exhaustive search over factor combinations up to `maxk`
3. **Consistency Validation:** Split-sample validation ensures reproducibility
4. **Selection:** Choose subgroup based on `sg_focus` criterion

Two-Stage Consistency Algorithm: When `use_twostage = TRUE`, the consistency evaluation uses an optimized algorithm that can provide 3-10x speedup:

- **Stage 1:** Quick screening with `n.splits.screen` splits eliminates clearly non-viable candidates
- **Stage 2:** Sequential batched evaluation with early stopping for candidates passing Stage 1

The two-stage algorithm is recommended for:

- Exploratory analyses with many candidate subgroups
- Large `fs.splits` values (>200)

- Iterative model development

For final regulatory submissions, `use_twostage = FALSE` may be preferred for exact reproducibility.

Value

A list of class "forestsearch" containing:

grp.consistency Consistency evaluation results including:

- `out_sg`: Selected subgroup based on `sg_focus`
- `sg_focus`: Focus criterion used
- `df_flag`: Treatment recommendations
- `sg.harm`: Subgroup definition labels (character vector of cut names) – same as the top-level `sg.harm`
- `sg.harm.id`: per-subject 0/1 membership indicator (*not* a character vector of cut expressions; see the *Field naming collision* section below)
- `algorithm`: "twostage" or "fixed"
- `n_candidates_evaluated`: Number evaluated
- `n_passed`: Number passing threshold

find.grps Subgroup search results

confounders.candidate Candidate confounders considered

confounders.evaluated Confounders after variable selection

df.est Analysis data with treatment recommendations

df.predict Prediction data with recommendations (if provided)

df.test Test data with recommendations (if provided)

minutes_all Total computation time

grf_res GRF results object

sg_focus Subgroup focus criterion used

sg.harm Selected subgroup definition – character vector of factor-level cut names. **Recommended accessor** for displaying the identified subgroup (e.g., `paste(obj$sg.harm, collapse = " & ")`). NULL if no subgroup was identified.

grf_cuts GRF cut points used

prop_maxk Proportion of max combinations searched

max_sg_est Maximum subgroup HR estimate

grf_plot GRF plot object (if `plot.grf = TRUE`)

args_call_all All arguments for reproducibility

threshold_config The resolved threshold configuration: `outcome_type`, `effect_measure`, `screening`, `consistency`, `screening_natural`, `consistency_natural`, `scale`, `pconsistency` and two description strings. **Read `$scale` before reading `$screening`.** The phrase "the screening threshold" names two different numbers depending on which object is inspected:

- On the **survival** path `$screening` is `log(hr.threshold)` – the scale the admission set and multiplier resampling work on – while the candidate search itself compares the fitted hazard ratio against `hr.threshold` on the **natural** scale. The two are the same threshold on two scales, and `$screening_natural` holds the search's value.

- On the **GLM** paths `$screening` is the value the search compares against directly: the link scale for ratio measures (log of the supplied ratio) and the identity scale for RD, IRD and MD.

`$consistency` follows the same convention, and `$pconsistency` is the rate `pconsistency.threshold`, which has no scale.

family_status Character scalar recording whether the candidate family the identifier ranked over is fixed – the condition multiplier resampling requires (see `mr_inference`). "no-front-end": `subgroup_method = "consistency"` with `use_lasso`, `use_grf` and `use_dina` all FALSE. Named for what it checks: no fitted model shapes the family on the observed data. This is weaker than the manuscript's Section 2.1 fixed family, which additionally requires resample-invariant cut locations – quantile-derived cuts are not. "conditional-removable": the same method with at least one front end on – data-dependent, but fixed by turning those off. "conditional-inherent": `subgroup_method` of "dina" or "grf", where the family is generated by fitting a model to the same data and cannot be made fixed. Descriptive only; it never raises a condition.

mr_inference Multiplier-resampling (MR) result, or NULL when MR did not run. See [fs_mr_inference](#) for fields (naive, debiased, selection_bias, settings, harm_flag, timing_seconds). MR approximates the full bootstrap (FB) to leading order; its interval is the infinitesimal-jackknife analogue of the FB interval under the default `ci_method = "field"` and under "ij", and the subgroup robust-SE interval under "wald". Under the default the return also carries the `field` element.

mr_harm_confirmed Logical, three-valued. TRUE when MR ran and the de-biased estimate still indicates harm under `confirm_rule`; FALSE when MR ran and it does not; NA when MR did **not** run or could not be computed. NA is therefore *absence of evidence, not evidence against harm* – test it with `is.na()` before `isTRUE()`, which collapses NA to FALSE and so reports "harm not confirmed" for an analysis that never ran. See the vocabulary section.

Threshold vocabulary and resolved defaults

Three arguments carry thresholds, and two of them have the word "consistency" in the name. They are different quantities:

- `c1 – effect.threshold` (**legacy** `hr.threshold`) The **screening** threshold on a candidate subgroup's own effect, estimated on the full sample. A candidate must clear it to enter the consistency stage.
- `c2 – consistency.threshold` (**legacy** `hr.consistency`) An **effect** threshold, applied to the subgroup's effect estimated within each split half. **Not** a proportion, and **not** the consistency rate.
- `p* – pconsistency.threshold` A **proportion** in $[0, 1]$: the fraction of splits that must clear `c2`. **Not** an effect threshold.

In one line: `c2` sets the bar a split half must clear, and `p*` counts how often that bar is met.

Resolved defaults, per estimand. Reading `hr.consistency`'s default as "1.0" is misleading on the GLM paths: when neither threshold is supplied, the value a run actually uses is resolved from the estimand. The resolved pairs are:

resolved estimand	c1	c2
HR (survival)	1.25, compared by the search on the natural HR scale	1.0
OR, RR, IRR	$\log(1.25)$	$\log(1.0)$, i.e. 0
RD	0.05	0.0
IRD	0.01	0.0
MD	0.0	0.0

Supplying `effect.threshold` or `consistency.threshold` (or the legacy names) suppresses the corresponding remap and the supplied value is used on the scale shown above.

The pair rule, on the consistency path with a ratio estimand. For `subgroup_method = "consistency"` with survival (HR) or binary `effect_measure = "OR"` – the two estimands whose `c1` and `c2` sit on one comparable ratio scale – the two thresholds are treated as a pair:

- `c2 > c1` is an **error**. It is degenerate: the consistency-stage entry condition re-screens the admitted family on `c2`, so candidates the screen let through are discarded at the stage boundary, and when none survives the consistency stage does not run at all and the fit reads as a finding of no subgroup. `c2 = c1` is allowed.
- `c1` supplied **without** `c2` derives `c2 = 0.80 * c1` on the **ratio** scale (on the log scale an additive shift of $\log(0.80)$, never $0.80 * \log(c1)$): 1.25 -> 1.00, 1.00 -> 0.80, 0.90 -> 0.72. Announced with a message, and carried into every bootstrap replicate and CV fold. The default pair (1.25, 1.0) is the rule's fixed point, so a call that sets neither threshold is unaffected. An explicitly supplied `c2`, in either spelling, is never overridden.
- Supplying both spellings of one threshold at disagreeing values is an error naming both.

Neither rule applies under `subgroup_method = "dina"` or `"grf"` (which return before the consistency stage and never consult `c2`), nor to the identity-scale estimands RD, IRD, MD, nor to IRR.

`pconsistency.threshold` is **not** remapped. It is a rate, and it is identical for every `outcome_type` and every estimand.

Field naming collision with GRF results

The top-level `sg.harm` on this object and the `grp.consistency$sg.harm` nested field both hold a character vector of cut expressions. The nested `grp.consistency$sg.harm.id` field holds a *per-subject 0/1 membership indicator*, not a cut vector – unlike `sg.harm.id` on GRF result objects, which *does* hold cut expressions. See [subgroup.consistency](#) and [grf.subg.harm.glm](#) for the full discussion of this naming collision. Code that must handle both object types should prefer the top-level `sg.harm` accessor on forestsearch results.

Vocabulary FB MR and harm confirmation

Four different things in this package used to share the word "gate". They are distinct, they happen at different stages, and only the last one is a pass/fail decision:

Admissibility criteria Which candidate subgroups are even eligible to be considered: `hr.threshold`, `pconsistency.threshold`, `n.min`, `d0.min`, `d1.min`, `maxk`. Applied during the search.

Selection rule Which admissible candidate is chosen as the estimated subgroup: `sg_focus`. Applied during the search.

Post-selection inference Correcting the chosen subgroup's treatment effect for the optimism of having chosen it, and putting an interval around the corrected value. This is FB and MR below. Applied *after* the search.

Harm confirmation The one pass/fail: does the corrected estimate still indicate harm? This is `confirm_rule` against `t_confirm`, reported as `mr_harm_confirmed`.

Full bootstrap (FB) — `forestsearch_bootstrap_dofuture`. Resamples *subjects*, and re-runs the entire subgroup search inside every replicate, so the replicate may select a different subgroup than the original analysis did. That is what lets it correct the selection-induced bias in the reported treatment effect, and supply an infinitesimal-jackknife interval. It is the reference method, and it is expensive: cost scales with `nb_boots` times the cost of one search.

Multiplier resampling (MR) — `fs_mr_inference`, turned on by `mr_inference = TRUE`. Resamples nothing; it perturbs the per-subject influence contributions (`dfbeta`) of a *single* set of fits with random mean-zero multipliers, and re-applies the selection rule on each perturbed draw. It corrects the same selection bias FB corrects and produces the same kind of interval, to leading order, at a small fraction of the cost — no refits. MR is **post-selection inference on a completed analysis**: it runs after the subgroup has been chosen and **cannot change which subgroup is identified**. Running with `mr_inference = TRUE` and `FALSE` gives identical `sg.harm`, `df.est` and `max_sg.est`. MR approximates FB; it does not replace it.

Harm confirmation. Once MR has produced a de-biased estimate, `mr_inference_args$confirm_rule` decides whether that estimate still indicates harm:

"point" (**default**) Compares the de-biased *point estimate* to `t_confirm`.

"ci" Compares the one-sided 95% selection-adjusted *lower bound* to `t_confirm`. Strictly the stronger requirement, so anything it confirms "point" also confirms.

`t_confirm` is the threshold those rules compare against, on the **effect** scale (HR, OR, RR, IRR; or RD, MD for differences) — not the working log scale used internally. Left NULL it resolves to the null effect: 1 for ratio measures, 0 for differences. It is deliberately set near the null rather than at `hr.threshold`, because the de-biasing correction over-shrinks genuine effects, so re-applying the screening threshold to a corrected estimate would discard true findings.

`mr_harm_confirmed` carries the answer, and is three-valued: TRUE = MR ran and harm is confirmed; FALSE = MR ran and harm is not confirmed; NA = MR did not run, or could not be computed. NA **is not evidence against harm**. It arises when `mr_inference = FALSE` (the default), when no subgroup was identified, or when the MR step was skipped or failed for the outcome/consistency combination in use. Because `isTRUE(NA)` is FALSE, code that reads this field with `isTRUE()` alone silently reports "harm not confirmed" for an analysis that was never performed; branch on `is.na()` first.

References

- FDA Guidance for Industry: Enrichment Strategies for Clinical Trials
- Athey & Imbens (2016). Recursive partitioning for heterogeneous causal effects. PNAS.
- Wager & Athey (2018). Estimation and inference of heterogeneous treatment effects using random forests. JASA.

See Also

[subgroup.consistency](#) for consistency evaluation details [forestsearch_bootstrap_dofuture](#) for the full bootstrap (FB) [fs_mr_inference](#) for multiplier resampling (MR), the `mr_inference = TRUE` step [mr_estimates_table](#) to render the MR estimates [fs_fdr_report](#) to sweep harm-confirmation thresholds (`c_confirm`) and measure what confirmation buys [forestsearch_Kfold](#) for cross-validation

Package website: <https://larry-leon.github.io/forestsearch/>

Source code: <https://github.com/larry-leon/forestsearch>

Examples

```
## Not run:
# Example 1: Standard analysis (backward compatible)
result <- forestsearch(
  df.analysis = trial_data,
  sg_focus = "hr",
  hr.threshold = 1.25,
  pconsistency.threshold = 0.90,
  fs.splits = 400,
  details = TRUE
)

# Example 2: Fast exploratory analysis with two-stage
result_fast <- forestsearch(
  df.analysis = trial_data,
  sg_focus = "maxSG",
  hr.threshold = 1.15,
  pconsistency.threshold = 0.85,
  fs.splits = 500,
  use_twostage = TRUE,
  details = TRUE
)

# Example 3: Two-stage with custom parameters
result_custom <- forestsearch(
  df.analysis = trial_data,
  sg_focus = "hr",
  hr.threshold = 1.3,
  pconsistency.threshold = 0.95,
  fs.splits = 600,
  use_twostage = TRUE,
  twostage_args = list(
    n.splits.screen = 50,
    batch.size = 25,
    conf.level = 0.99
  ),
  parallel_args = list(plan = "multisession", workers = 4),
  details = TRUE
)

# Example 4: Binary outcome with OR thresholds
```

```

result_binary <- forestsearch(
  df.analysis = trial_data,
  outcome_type = "binary",
  effect_measure = "OR",
  effect.threshold = 1.5,
  consistency.threshold = 1.3,
  pconsistency.threshold = 0.90
)

# Example 5: Binary outcome with RD thresholds
result_rd <- forestsearch(
  df.analysis = trial_data,
  outcome_type = "binary",
  effect_measure = "RD",
  effect.threshold = 0.07,      # 7 pct-point harm
  consistency.threshold = 0.03 # 3 pct-point per split
)

## End(Not run)

```

forestsearch_bootstrap_dofuture

ForestSearch Bootstrap with doFuture Parallelization

Description

Orchestrates bootstrap analysis for ForestSearch using doFuture parallelization. Implements bias correction methods to adjust for optimism in subgroup selection.

Usage

```

forestsearch_bootstrap_dofuture(
  fs.est,
  nb_boots,
  seed = 8316951L,
  details = FALSE,
  show_three = FALSE,
  parallel_args = list(),
  digits = 4,
  mr_in_replicates = FALSE
)

```

Arguments

fs.est	List. ForestSearch results object from forestsearch . Must contain df.est (data frame) and args_call_all (list of arguments).
nb_boots	Integer. Number of bootstrap samples (recommend 500-1000).

seed	Integer. Random seed for reproducibility of bootstrap sample generation. Default 8316951L. The value is passed to both <code>bootstrap_ystar</code> (which constructs the $n \times B$ bootstrap index matrix) and <code>bootstrap_results</code> (which re-runs the ForestSearch algorithm on each replicate); both calls must use the same seed to ensure bootstrap index alignment. Set to NULL for a non-reproducible run.
details	Logical. If TRUE, prints detailed progress information. Default: FALSE.
show_three	Logical. If TRUE, shows verbose output for first 3 bootstrap iterations for debugging. Default: FALSE.
parallel_args	List. Parallelization configuration with elements: • plan: Character. One of "multisession", "multicore", "callr", or "sequential" • workers: Integer. Number of parallel workers • show_message: Logical. Show parallel setup messages If empty list, inherits settings from original forestsearch call.
digits	Integer. Decimal places for construction-time numeric formatting in FSsg_tab (arm summaries, Diff, effect-estimate CI, bias-corrected CI). Default 4 to provide round-down headroom for downstream display reformatting (e.g., <code>summarize_bootstrap_results = ...</code>), which can round down but cannot recover precision lost at construction). Threaded through <code>format_CI</code> and <code>SG_tab_estimates_glm</code> .
mr_in_replicates	Logical. Whether multiplier resampling (MR) runs <i>inside</i> each bootstrap replicate. Default FALSE. <code>mr_inference</code> reaches each replicate through the reconstructed call in <code>fs.est\$args_call_all</code>, so an analysis fitted with <code>mr_inference = TRUE</code> would otherwise run MR in every replicate – and nothing consumes the result. Under the default the flag is stripped from the replicate call, so that work is not done at all. This does <i>not</i> affect the top-level analysis: <code>fs.est</code> keeps whatever de-biased estimate it was fitted with. TRUE lets <code>mr_inference</code> propagate unchanged into each replicate <i>and</i> retains the result, returned as <code>\$mr_replicates</code> – a list of length <code>nb_boots</code> whose <code>b</code>-th element is that replicate's <code>mr_inference</code> object, or NULL where the replicate failed or identified no subgroup. MR therefore runs in the replicates only if the original analysis set <code>mr_inference = TRUE</code>; running it and discarding the result is never a reachable combination. The retained objects are not aggregable into an estimate for the original analysis. Each replicate re-runs the search, selects its own subgroup, and debiases <i>that</i> subgroup – so the collection describes MR's sampling behaviour across resampled selections, not a better estimate of the original quantity. Averaging them, or pooling their intervals, is an error. The bias-corrected estimate for the original analysis is the one this function already returns in <code>SG_CIs</code>, and the original subgroup's own MR estimate is on <code>fs.est</code>.

Value

An `fs_bootstrap` object (a list with class `c("fs_bootstrap", "list")`) containing:

results Data.table with bias-corrected estimates for each bootstrap iteration

SG_CIs List of confidence intervals for H and Hc (raw and bias-corrected)

FSsg_tab Formatted table of subgroup estimates

Ystar_mat Matrix (nb_boots x n) of bootstrap sample indicators

H_estimates Detailed estimates for subgroup H

Hc_estimates Detailed estimates for subgroup Hc

mr_replicates NULL under the default `mr_in_replicates = FALSE`. Otherwise a list of length `nb_boots`, element `b` holding replicate `b`'s `mr_inference` object (or NULL for a failed or no-subgroup replicate), each computed against *that replicate's* candidate family and selected subgroup. See the `mr_in_replicates` parameter for why these must not be averaged.

summary (If `create_summary=TRUE`) Enhanced summary with tables and diagnostics

nb_boots Integer. Number of bootstrap iterations requested. Used by `print.fs_bootstrap` to compute identification percentages without re-inspecting results. Added in v0.2.0.

original_sg Character vector. The subgroup identified by the primary `forestsearch` call (`fs.est$sg.harm`), carried forward so that `print.fs_bootstrap` can report exact- and partial-match rates without retaining a reference to `fs.est`. Added in v0.2.0.

outcome_type Character. Outcome type of the underlying analysis: one of "survival", "binary", "continuous", or "count". Added in v0.2.0.

effect_measure Character. Effect measure label ("HR" for survival; "OR", "RR", "RD", "IRR", "IRD", or "MD" for GLM outcomes). Added in v0.2.0.

est.scale Character. Estimation scale for confidence intervals ("hr" or "1/hr"). Added in v0.2.0.

Bias Correction Methods

Two bias correction approaches are implemented:

1. Method 1 (Simple Optimism):

$$H_{adj1} = H_{obs} - (H_*^* - H_{obs}^*)$$

where H_*^* is the new subgroup HR on bootstrap data and H_{obs}^* is the new subgroup HR on original data.

2. Method 2 (Double Bootstrap):

$$H_{adj2} = 2 \times H_{obs} - (H_* + H_*^* - H_{obs}^*)$$

where H_* is the original subgroup HR on bootstrap data.

Variable Naming Convention

- H: Original subgroup (harm/questionable, `treat.recommend == 0`)
- Hc: Complement subgroup (`recommend, treat.recommend == 1`)
- `_obs`: Estimate from original data
- `_star`: Estimate from bootstrap data
- `_biasadj_1`: Bias correction method 1
- `_biasadj_2`: Bias correction method 2

Valid for every identifier configuration

Unlike multiplier resampling, the full bootstrap places no alignment requirement on the identifier. It replays the entire pipeline on each resample – refitting the front ends, re-enumerating the candidate family and re-running selection – rather than linearizing the selection event, so neither of MR’s two conditions (selection ranking on the inferential coefficient, and a fixed candidate family) bears on its validity. In the equivalence result those conditions state, the bootstrap is the reference standard, not a party to it.

It is therefore valid for all six identifier configurations, including the three that rank on a native statistic: DINA under `dina_select_statistic = "dina"`, GRF under `grf_select_statistic = "dr"`, and GRF under `grf_selection = "tree"`. For those, it correctly estimates the selection bias that ranking on the native statistic induces on the reported effect $\hat{\beta}$. That is a coherent quantity, and the estimate of it is valid – it simply sits outside the framework the manuscript develops, which is about identifiers whose selection map and reported effect are the same functional.

The one place an alignment check does apply here is `mr_in_replicates = TRUE`, which asks for MR *inside* each replicate; see that parameter.

Performance

Typical runtime: 1-5 seconds per bootstrap iteration. For 1000 bootstraps with 6 workers, expect 3-10 minutes total. Memory usage scales with dataset size and number of workers.

Requirements

- Original `fs.est` must have identified a valid subgroup
- Requires packages: `data.table`, `foreach`, `doFuture`, `survival`
- For plots: requires `ggplot2`

See Also

[forestsearch](#) for initial subgroup identification, and its vocabulary section for how the full bootstrap (FB) implemented here relates to multiplier resampling (MR) [fs_mr_inference](#) for MR, the fast refit-free approximation to this function [mr_estimates_table](#) to render MR estimates in this function’s table layout [fs_fdr_report](#) for harm-confirmation threshold sweeps [bootstrap_results](#) for the core bootstrap worker function [build_cox_formula](#) for Cox formula construction [fit_cox_models](#) for Cox model fitting

Examples

```
## Not run:
# Run ForestSearch
fs_result <- forestsearch(
  df.analysis = mydata,
  outcome.name = "time",
  event.name = "status",
  treat.name = "treatment",
  confounders.name = c("age", "sex", "stage")
)
```

```

# Run bootstrap with bias correction
boot_results <- forestsearch_bootstrap_dofuture(
  fs.est = fs_result,
  nb_boots = 1000,
  parallel_args = list(
    plan = "multisession",
    workers = 6,
    show_message = TRUE
  ),
  create_summary = TRUE,
  create_plots = TRUE
)

# View results
print(boot_results$FSsg_tab)
print(boot_results$summary$table)

# Check success rate
mean(!is.na(boot_results$results$H_biasadj_2))

## End(Not run)

```

forestsearch_Kfold	<i>ForestSearch K-Fold Cross-Validation</i>
--------------------	---

Description

This function assesses the stability and reproducibility of ForestSearch subgroup identification through cross-validation. For each fold:

1. Train ForestSearch on (K-1) folds
2. Apply the identified subgroup to the held-out fold
3. Compare predictions to the original full-data analysis

Usage

```

forestsearch_Kfold(
  fs.est,
  Kfolds = .fs_cv_nobs(fs.est),
  seedit = 8316951L,
  parallel_args = list(plan = "multisession", workers = 6, show_message = TRUE),
  sg0.name = "Not recommend",
  sg1.name = "Recommend",
  details = FALSE,
  mr_in_replicates = FALSE
)

```

Arguments

fs.est	List. ForestSearch results object from forestsearch . Must contain df.est (data frame) and args_call_all (list of arguments).
Kfolds	Integer. Number of folds (default: nrow(fs.est\$df.est) for LOO).
seedit	Integer. Random seed for fold assignment (default: 8316951).
parallel_args	List. Parallelization configuration with elements: • plan: Character. One of "multisession", "multicore", "sequential" • workers: Integer. Number of parallel workers • show_message: Logical. Show parallel setup messages parallel_args controls fold-level parallelism only: the fold loop is distributed across workers from this argument, while each fold's inner forestsearch refit is pinned to plan = "sequential" (deliberate nested-parallelism avoidance, so fold-level workers are not oversubscribed by inner-search workers). The plan is taken from parallel_args, not from any ambient future::plan() set by the caller (which is saved and restored on exit). For example, to run the folds on four workers: parallel_args = list(plan = "multisession", workers = 4).
sg0.name	Character. Label for subgroup 0 (default: "Not recommend").
sg1.name	Character. Label for subgroup 1 (default: "Recommend").
details	Logical. Print progress details (default: FALSE).
mr_in_replicates	Logical. Whether multiplier resampling (MR) runs <i>inside</i> each CV fold. Default FALSE. mr_inference reaches each fold through the reconstructed call in fs.est\$args_call_all, so an analysis fitted with mr_inference = TRUE would otherwise run MR in every fold – and nothing consumes the result. Under the default the flag is stripped from the per-fold call, so that work is not done at all. The top-level analysis on fs.est is unaffected. TRUE lets mr_inference propagate unchanged into each fold <i>and</i> retains the result, returned as \$mr_replicates – a list of length Kfolds whose k-th element is fold k's mr_inference object, or NULL where that fold identified no subgroup. MR therefore runs in the folds only if the original analysis set mr_inference = TRUE. The retained objects are not aggregable into an estimate for the original analysis. Each fold re-runs the search on its training split, selects its own subgroup, and de-biases <i>that</i> subgroup – so the collection describes MR's behaviour across fold-specific selections, not a better estimate of the original quantity. Averaging them, or pooling their intervals, is an error.

Details

Performs K-fold cross-validation for ForestSearch, evaluating subgroup identification and agreement between training and test sets.

Value

List with components:

resCV Data frame with CV predictions for each observation

cv_args Arguments used for CV ForestSearch calls

timing_minutes Execution time in minutes

prop_SG_found Percentage of folds where a subgroup was found

sg_analysis Original subgroup definition from full-data analysis

sg0.name, sg1.name Subgroup labels

Kfolds Number of folds used

sens_summary Named vector of sensitivity metrics (sens_H, sens_Hc, ppv_H, ppv_Hc)

find_summary Named vector of subgroup-finding metrics (Any, Exact, etc.)

mr_replicates NULL under the default `mr_in_replicates = FALSE`. Otherwise a list of length `Kfolds`, element `k` holding fold `k`'s `mr_inference` object (or NULL), each computed against *that fold's* candidate family and selected subgroup. Not aggregable – see the `mr_in_replicates` parameter.

Cross-Validation Types

- **Leave-One-Out (LOO):** When `Kfolds = nrow(df)`, each observation is held out once. Most thorough but computationally intensive.
- **K-Fold:** When `Kfolds < nrow(df)`, data is split into `K` roughly equal folds. Good balance of bias-variance tradeoff.

Output Metrics

The returned `resCV` data frame contains:

- `treat.recommend`: Prediction from CV model
- `treat.recommend.original`: Prediction from full-data model
- `cvindex`: Fold assignment
- `sg1, sg2`: Subgroup definitions found in each fold

See Also

[forestsearch](#) for initial subgroup identification [forestsearch_KfoldOut](#) for summarizing CV results [forestsearch_tenfold](#) for repeated K-fold simulations

Examples

```
## Not run:
# Run initial ForestSearch
fs_result <- forestsearch(
  df.analysis = trial_data,
  outcome.name = "time",
  event.name = "status",
```



```

    treat.name = "treatment",
    confounders.name = c("age", "biomarker")
  )

  # Run 10-fold cross-validation
  cv_results <- forestsearch_Kfold(
    fs.est = fs_result,
    Kfolds = 10,
    parallel_args = list(plan = "multisession", workers = 4),
    details = TRUE
  )

  # Summarize results
  cv_summary <- forestsearch_KfoldOut(cv_results, outall = TRUE)

  ## End(Not run)

```

forestsearch_KfoldOut *ForestSearch K-Fold Cross-Validation Output Summary*

Description

Summarizes cross-validation results for ForestSearch, including subgroup agreement and performance metrics.

Usage

```
forestsearch_KfoldOut(res, details = FALSE, outall = FALSE, digits = 4)
```

Arguments

<code>res</code>	List. Result object from ForestSearch cross-validation, must contain elements: <code>cv_args</code> , <code>sg_analysis</code> , <code>sg0.name</code> , <code>sg1.name</code> , <code>Kfolds</code> , <code>resCV</code> .
<code>details</code>	Logical. Print details during execution (default: FALSE).
<code>outall</code>	Logical. If TRUE, returns all summary tables; if FALSE, returns only metrics (default: FALSE).
<code>digits</code>	Integer. Decimal places for construction-time numeric formatting in the per-fold summary tables built when <code>outall = TRUE</code> (arm summaries, Diff, effect-estimate CI). Default 4 to provide round-down headroom for downstream display reformatting (e.g., <code>cv_summary_tables(digits = ...)</code>). Threaded through SG_tab_estimates_glm for GLM outcomes. Ignored when <code>outall = FALSE</code> .

Value

If `outall=FALSE`, a list with `sens_metrics_original` and `find_metrics`. If `outall=TRUE`, a list with summary tables and metrics.

Examples

```
## Not run:
library(survival)
df <- survival::gbsg
df$grade3 <- as.integer(df$grade == "3")
fs <- forestsearch(df,
  confounders.name = c("age", "meno", "size", "grade3", "nodes", "pgr", "er"),
  outcome.name = "rfstime", treat.name = "hormon", event.name = "status")
oob <- forestsearch_Kfold(fs.est = fs, Kfolds = nrow(df))
result <- forestsearch_KfoldOut(oob, outall = TRUE)

## End(Not run)
```

forestsearch_tenfold *ForestSearch Repeated K-Fold Cross-Validation*

Description

This function performs multiple independent K-fold cross-validations to assess the variability in subgroup identification. Each simulation:

1. Randomly shuffles the data
2. Performs K-fold CV
3. Records sensitivity and agreement metrics

Results are summarized across all simulations.

Usage

```
forestsearch_tenfold(
  fs.est,
  sims,
  Kfolds = 10,
  details = TRUE,
  seed = 8316951L,
  parallel_args = list(plan = "multisession", workers = 6, show_message = TRUE),
  keep_resCV = FALSE,
  mr_in_replicates = FALSE
)
```

Arguments

fs.est	List. ForestSearch results object from forestsearch .
sims	Integer. Number of simulation repetitions.
Kfolds	Integer. Number of folds per simulation (default: 10).
details	Logical. Print progress details (default: TRUE).

seed	Integer. Base random seed for fold shuffling. Default 8316951L. Each simulation uses seed + 1000 * ksim for reproducibility.
parallel_args	List. Parallelization configuration (elements plan / workers / show_message; see Parallelization Strategy). Here parallel_args controls simulation-level parallelism: the sims repetitions are distributed across workers, while each simulation runs its K-fold CV – and every inner <code>forestsearch</code> refit – sequentially (so workers are not oversubscribed by nested parallelism). As in <code>forestsearch_Kfold</code> , the plan is taken from parallel_args, not from any ambient <code>future::plan()</code> set by the caller (which is saved and restored on exit). For example, to run the simulations on four workers: <code>parallel_args = list(plan = "multisession", workers = 4)</code> .
keep_resCV	Logical. If TRUE, preserve the full per-subject cross-validation prediction data frame for every simulation in the returned <code>resCV_all</code> slot. Useful for post-hoc metric computation, reproducibility debugging, or auditing which subjects landed in which fold. Memory cost scales as <code>sims * nrow(df.est) * ~9 columns</code> (for a 686-subject trial with 200 sims, ~10 MB; for a 10,000-subject trial with 1,000 sims, ~1 GB). Default: FALSE. Regardless of this flag, a compact <code>fold_summary</code> (one row per sim x fold) is always returned; see Return.
mr_in_replicates	Logical. Whether multiplier resampling (MR) runs <i>inside</i> each CV fold. Default FALSE. <code>mr_inference</code> reaches each fold through the reconstructed call in <code>fs.est\$args_call_all</code>, so an analysis fitted with <code>mr_inference = TRUE</code> would otherwise run MR in every fold of every simulation – <code>sims * Kfolds</code> runs – and nothing consumes the result. Under the default the flag is stripped from the per-fold call, so that work is not done at all. The top-level analysis on <code>fs.est</code> is unaffected. TRUE lets <code>mr_inference</code> propagate unchanged into each fold <i>and</i> retains the results, returned as <code>\$mr_replicates</code> – a list with one element per successful simulation (named <code>sim<k></code> after its <code>sim_id</code>, so a dropped simulation shows as a gap rather than a silent re-index), each itself a list of length <code>Kfolds</code> holding that fold's <code>mr_inference</code> object or NULL. MR therefore runs in the folds only if the original analysis set <code>mr_inference = TRUE</code>. The retained objects are not aggregable into an estimate for the original analysis. Each fold re-runs the search on its training split, selects its own subgroup, and de-biases <i>that</i> subgroup – so the collection describes MR's behaviour across fold-specific selections, not a better estimate of the original quantity. Averaging them, or pooling their intervals, is an error.

Details

Runs repeated K-fold cross-validation simulations for ForestSearch and summarizes subgroup identification stability across repetitions.

Value

List with components:

sens_summary Named vector of median sensitivity metrics across simulations

find_summary Named vector of median subgroup-finding metrics

sens_out Matrix of sensitivity metrics (sims x metrics)

find_out Matrix of finding metrics (sims x metrics)

fold_summary Data frame with one row per (sim, fold) combination. Columns: sim, fold, n_test, sg1, sg2, grf_cuts, pconsistency, training_fs_hr, n_candidates_evaluated, any_found. The grf_cuts column records the GRF policy-tree cut expressions returned for each training fold (collapsed with " | " when multiple cuts are returned; NA when GRF was not used, failed, or returned no cuts). The pconsistency column records the consistency probability (Pcons) achieved by the identified subgroup on each fold (NA_real_ when no subgroup was identified or the training ForestSearch call errored). The training_fs_hr column records the in-sample effect estimate for the identified subgroup on the training fold, on its NATURAL SCALE regardless of outcome type (hazard ratio for survival; odds ratio, rate ratio, or risk ratio for GLM ratio measures after internal exponentiation; mean/risk difference for GLM additive measures); optimistically biased relative to any independent estimate and surfaced for diagnostic comparison only, with NA_real_ when no subgroup was identified. The n_candidates_evaluated column records the integer count of candidate subgroups actually evaluated for consistency on this training fold (populated whenever the consistency stage ran, even if zero candidates met the threshold and thus no subgroup was identified; NA_integer_ when the training call errored or consistency evaluation did not run). Always returned (compact; cheap). Lets you tabulate, for example, which subgroup was identified in each fold of each simulation, the empirical distribution of GRF cut choices across the full sim x fold grid, the relationship between GRF's cut and the final identified subgroup, or the near-miss consistency values among folds that did not surface a subgroup.

resCV_all List of length sims; element *i* is the per-subject resCV data frame from simulation *i*. Only populated when keep_resCV = TRUE; otherwise NULL.

timing_minutes Total execution time

sims Number of simulations run

Kfolds Number of folds per simulation

mr_replicates NULL under the default mr_in_replicates = FALSE. Otherwise a nested list: one element per successful simulation (named sim<k>), each a list of length Kfolds holding that fold's mr_inference object (or NULL), computed against *that fold's* candidate family and selected subgroup. Not aggregable – see the mr_in_replicates parameter.

Parallelization Strategy

Unlike the single K-fold function which parallelizes across folds, this function parallelizes across simulations for better efficiency when running many repetitions. Each simulation runs its K-fold CV sequentially.

See Also

[forestsearch_Kfold](#) for single K-fold CV [forestsearch_KfoldOut](#) for summarizing CV results

Examples

```
## Not run:
# Run 100 repetitions of 10-fold CV
```

```
tenfold_results <- forestsearch_tenfold(
  fs.est = fs_result,
  sims = 100,
  Kfolds = 10,
  parallel_args = list(plan = "multisession", workers = 6),
  details = TRUE
)

# View summary
print(tenfold_results$sens_summary)
print(tenfold_results$find_summary)

## End(Not run)
```

forest_height

Recommended figure height for a forest panel

Description

Converts a panel's row count to the fig.height (inches) used by the vignette chunks: rows * row_height + overhead, rounded to 0.1.

Usage

```
forest_height(
  x,
  panel = c("single", "combo", "highrisk"),
  row_height = 0.45,
  overhead = 1.5
)
```

Arguments

x	A "subgroup_sims_summary" object.
panel	"single", "combo", or "highrisk".
row_height	Inches per row (default 0.45).
overhead	Fixed inches for title, x-axis, and footnote (default 1.5).

Value

A single numeric, rounded to one decimal.

See Also

[plot.subgroup_sims_summary\(\)](#)

Examples

```
## Not run:
fh_single <- forest_height(S, "single")

## End(Not run)
```

format_bootstrap_table
<i>Format Bootstrap Results Table with gt</i>

Description

Creates a publication-ready table from ForestSearch bootstrap results, with bias-corrected confidence intervals, informative formatting, and optional subgroup definition footnote.

Usage

```
format_bootstrap_table(
  FSsg_tab,
  nb_boots,
  est.scale = "hr",
  boot_success_rate = NULL,
  sg_definition = NULL,
  title = NULL,
  subtitle = NULL
)
```

Arguments

FSsg_tab	Data frame or matrix from forestsearch_bootstrap_dofuture()\$FSsg_tab
nb_boots	Integer. Number of bootstrap iterations performed
est.scale	Character. "hr" or "1/hr" for effect scale
boot_success_rate	Numeric. Proportion of bootstraps that found subgroups
sg_definition	Character. Subgroup definition string to display as footnote (e.g., "{age>=50} & {nodes>=3}"). If NULL, no subgroup footnote is added.
title	Character. Custom title (optional)
subtitle	Character. Custom subtitle (optional)

Value

A gt table object

Examples

```
## Not run:
fs <- forestsearch(gbsg,
  confounders.name = c("age", "meno", "size", "grade3", "nodes", "pgr", "er"),
  outcome.name = "rfstime", treat.name = "hormon", event.name = "status")
fs_bc <- forestsearch_bootstrap_dofuture(fs, nb_boots = 100)
summaries <- summarize_bootstrap_results(fs$sg.harm, fs_bc)
format_bootstrap_table(summaries$table_data)

## End(Not run)
```

format_CI	<i>Format Confidence Interval for Estimates</i>
-----------	---

Description

Formats confidence interval for estimates.

Usage

```
format_CI(estimates, col_names, digits = 4)
```

Arguments

- estimates Data frame or data.table of estimates.
- col_names Character vector of column names for estimate, lower, upper.
- digits Integer. Decimal places for numeric formatting. Default 4 to provide round-down headroom for downstream display reformatting (e.g., summarize_bootstrap_results(digits = ...)).

Value

Character string formatted as \"estimate (lower, upper)\".

Examples

```
library(data.table)
est <- data.table(hr = 1.58, lower = 0.86, upper = 2.90)
format_CI(est, col_names = c("hr", "lower", "upper"))
```

format_fs_details	<i>Format ForestSearch Details Output for Beamer Two-Column Display</i>
-------------------	---

Description

Captures `forestsearch(details = TRUE)` console output and splits it into two columns for readable beamer slides. Left column shows variable selection (GRF, LASSO, candidate factors); right column shows subgroup search, consistency evaluation, and results.

Usage

```
format_fs_details(
  fs_output,
  split_after = "Candidate factors",
  fontsize = "scriptsize",
  col_widths = c(0.48, 0.52),
  max_width = 48
)
```

Arguments

<code>fs_output</code>	Character vector of captured output lines from <code>capture.output(forestsearch(..., details = TRUE))</code> .
<code>split_after</code>	Character string (regex). The output is split after the block matching this pattern. Default: "Candidate factors" which splits after the candidate factor list.
<code>fontsize</code>	Character. LaTeX font size for the output text. One of "tiny", "scriptsize", "footnotesize", "small". Default: "scriptsize".
<code>col_widths</code>	Numeric vector of length 2. Column widths as fractions of <code>\textwidth</code> . Default: <code>c(0.48, 0.52)</code> .
<code>max_width</code>	Integer. Maximum character width per line before wrapping. Long lines are wrapped at comma or space boundaries with a 4-space continuation indent. Default: 48 (suitable for half-slide columns at scriptsize).

Value

Invisibly returns a list with `left` and `right` character vectors. Side effect: emits LaTeX via `cat()` for use in a chunk with `results='asis'`.

Quarto Setup

No special LaTeX packages required. Works in any beamer frame without the `fragile` option.

Usage

In a Quarto beamer chunk with `results='asis'` and `echo=FALSE`, first capture the `forestsearch` output with `capture.output()`, then call `format_fs_details(fs_output)`.

Examples

```
## Not run:
fs_output <- capture.output(
  fs_res <- forestsearch(
    df = dat, outcome.name = "time", event.name = "status",
    treat.name = "trt", id.name = "id",
    confounders.name = confs, details = TRUE
  )
)
# Default: split after candidate factors
format_fs_details(fs_output, fontsize = "tiny")

# Custom split: after filtering summary
format_fs_details(fs_output, split_after = "Found.*candidate")

## End(Not run)
```

format_oc_results	<i>Format Operating Characteristics Results as GT Table</i>
-------------------	---

Description

Creates a formatted gt table from simulation operating characteristics results.

Usage

```
format_oc_results(
  results,
  analyses = NULL,
  metrics = "all",
  digits = 3,
  digits_hr = 3,
  title = "Operating Characteristics Summary",
  subtitle = NULL,
  use_gt = TRUE,
  effect_measure = NULL,
  dgm = NULL,
  subgroup_notation = c("harm", "benefit")
)
```

Arguments

results	data.table or data.frame. Simulation results from run_simulation_analysis or combined results from multiple simulations.
analyses	Character vector. Analysis methods to include. Default: NULL (all analyses in results)

metrics	Character vector. Metrics to display. Options include: "detection", "classification", "hr_estimates", "ahr_estimates", "cde_estimates", "subgroup_size", "all". Default: "all"
digits	Integer. Decimal places for proportions. Default: 3
digits_hr	Integer. Decimal places for hazard ratios. Default: 3
title	Character. Table title. Default: "Operating Characteristics Summary"
subtitle	Character or NULL. Table subtitle. When NULL (default) and a dgm is provided, an informative subtitle is auto-generated showing the number of simulations and true effect sizes using notation-aware labels (e.g., "100 sims, OR(H) = 2.00, OR(Hc) = 0.65").
use_gt	Logical. Return gt table if TRUE, data.frame if FALSE. Default: TRUE
effect_measure	Character or NULL. The effect measure used in the simulation (e.g., "HR", "OR", "MD"). When provided, identity-scale measures (MD, RD, IRD) are negated rather than reciprocated during benefit-search inversion. When NULL (default), auto-detected from dgm\$effect_measure if dgm is provided; otherwise ratio-scale inversion (1/x) is used for backward compatibility.
dgm	Optional DGM object (gbsg_dgm, aft_dgm_flex, or glm_dgm). When provided, effect_measure is auto-detected from dgm\$effect_measure (unless explicitly overridden) and column spanner labels adapt to the outcome type.
subgroup_notation	Character. "harm" (default) labels subgroups as H/Hc; "benefit" labels as G/Gc for benefit-search analyses (treatment switching).

Details

The function summarizes simulation results across multiple metrics:

- **Found:** Proportion of simulations finding a subgroup (any.H)
- **Classification:** Sen, spec, PPV, NPV
- **HR Estimates:** Mean Cox hazard ratios in true (H) and identified (H-hat) subgroups and their complements
- **AHR Estimates:** Mean average hazard ratios (from loghr_po) in true and identified subgroups
- **CDE Estimates:** Controlled direct effects (from theta_0/theta_1) in true and identified subgroups
- **Subgroup Size:** Average, min, max sizes

Column notation aligns with [build_estimation_table](#) and Leon et al. (2024): H = true (oracle) subgroup, H-hat = identified subgroup. The asterisk (*) is reserved for bootstrap bias-corrected estimates and is not used in this summary table.

Value

A gt table object (if use_gt = TRUE and gt package available) or data.frame

Examples

```
## Not run:
# format_oc_results() is called by summarize_simulation_results().
# See run_simulation_analysis() for the standard entry point.

## End(Not run)
```

fpr_approx

Approximate Procedure-Level False Positive Rate

Description

Computes an analytical approximation to the ForestSearch false positive rate under the null hypothesis of homogeneous treatment effect. The calculation accounts for (i) the sampling variability of the effect estimate in a subgroup of size `n_min`, (ii) the number of candidate subgroups searched (driven by the covariate space and `maxk`), and (iii) the screening threshold `c1`.

Usage

```
fpr_approx(
  effect_true = 0.65,
  p_event = 0.4,
  n_min = 60L,
  c1 = 1.25,
  n_confounders_cont = 6L,
  n_confounders_bin = 6L,
  cont_cutoff = 4L,
  maxk = 2L,
  effect_type = c("ratio", "difference"),
  treat_alloc = 0.5,
  rho = 0.3,
  n_eff = NULL,
  thresholds = NULL,
  verbose = TRUE
)
```

Arguments

<code>effect_true</code>	Numeric. True effect under H_0 on the ratio scale (e.g., OR or HR). Default 0.65 (protective treatment).
<code>p_event</code>	Numeric in (0, 1). Overall event rate (control arm baseline rate for binary; used to compute SE). Default 0.40.
<code>n_min</code>	Integer. Minimum subgroup size (<code>n.min</code> in <code>forestsearch()</code>). Default 60.
<code>c1</code>	Numeric. The screening threshold on a candidate subgroup's own effect – <code>forestsearch</code> 's <code>effect.threshold</code> (legacy <code>hr.threshold</code>). Candidates with

	effect > c1 pass screening. Default 1.25. This approximation covers the screening stage only, so the per-split effect threshold c2 (consistency.threshold) and the consistency rate p* (pconsistency.threshold) do not appear here.
n_confounders_cont	Integer. Number of continuous confounders. Default 6.
n_confounders_bin	Integer. Number of binary confounders. Default 6.
cont_cutoff	Integer. Number of cut points per continuous covariate (cont_cutoff in forestsearch()). Each continuous variable produces cont_cutoff candidate subgroups. Default 4.
maxk	Integer. Maximum interaction depth. 1 = single-variable subgroups only; 2 = up to two-variable interactions. Default 2.
effect_type	Character. "ratio" (OR, HR, IRR: log-scale SE) or "difference" (RD, MD: identity-scale SE). Default "ratio".
treat_alloc	Numeric in (0, 1). Treatment allocation fraction. Default 0.5 (1:1 randomisation).
rho	Numeric in [0, 1). Assumed average pairwise correlation between candidate test statistics, used for the correlation-adjusted FPR estimate. Default 0.30. Set to 0 for the independence (upper-bound) estimate only. Ignored when n_eff is set.
n_eff	Numeric or NULL. Pre-specified effective number of independent tests. When set, overrides the rho-based Sidak adjustment. Default NULL (compute from rho).
thresholds	Numeric vector. Additional thresholds to evaluate besides c1. Default c(1.0, 1.5, 2.0).
verbose	Logical. Print a formatted summary. Default TRUE.

Details

The approximation assumes:

- Normal approximation to the log-OR (or log-HR) distribution.
- Independent candidates (upper bound; true FPR is lower due to overlap). A correlation-adjusted estimate is also returned.
- Equal allocation to treatment arms within each subgroup.
- No consistency check adjustment (the reported FPR is the *screening-stage* FPR before pconsistency.threshold filtering).

Value

A list (class "fpr_approximation") with components:

se_log_effect SE of log(effect) in a subgroup of size n_min.
n_candidates Total number of candidate subgroups searched.
n_single Number of single-variable candidates.
n_pairs Number of two-variable candidates (0 if maxk == 1).

per_candidate Named numeric vector: $P(\text{effect} > \text{thr})$ for each threshold evaluated.

fpr_indep Named numeric vector: procedure-level FPR assuming independent candidates, for each threshold.

fpr_adjusted Named numeric vector: correlation-adjusted FPR, for each threshold.

params List of input parameters.

Examples

```
# ACTG175 binary harm search
fpr_approx(effect_true = 0.65, p_event = 0.40, n_min = 60, c1 = 1.25)

# Tighter threshold
fpr_approx(effect_true = 0.65, p_event = 0.40, n_min = 60, c1 = 2.0)

# Larger minimum subgroup
fpr_approx(effect_true = 0.65, p_event = 0.40, n_min = 100, c1 = 1.25)
```

fpr_calibration	<i>Direct Simulation-Based FPR Calibration at a Specific Threshold Pair</i>
-----------------	---

Description

Estimates the procedure-level false-positive rate (FPR) for a [forestsearch](#) analysis at a **specific** (c_1, c_2) threshold pair by running nsims null simulations. No power-law model or threshold-independence assumption is required.

Usage

```
fpr_calibration(
  c1,
  c2,
  df.analysis = NULL,
  fs_params = list(),
  nsims = 200L,
  nsims_verify = 0L,
  null_fn = NULL,
  n_sample = NULL,
  n_min = 60L,
  outcome_type_approx = NULL,
  d_eff_approx = NULL,
  prop_cens = 0,
  p_event = 0.5,
  sigma_y = 1,
  sim_seed = 42L,
  n_workers = 1L,
  quiet = FALSE
)
```

Arguments

c1	Numeric scalar. The screening threshold on a candidate subgroup's own effect. c1 is this function's name for <code>forestsearch</code> 's <code>effect.threshold</code> (legacy <code>hr.threshold</code>), and is passed through to it under that name, on the natural scale – <code>forestsearch()</code> resolves the comparison scale per estimand.
c2	Numeric scalar. The per-split effect threshold . c2 is this function's name for <code>forestsearch</code> 's <code>consistency.threshold</code> (legacy <code>hr.consistency</code>), passed through under that name on the natural scale. It is an effect threshold applied within each split half, not the consistency rate: the rate is <code>pconsistency.threshold</code> and travels in <code>fs_params</code> . Must satisfy $c2 \leq c1$.
df.analysis	Data frame. The analyst's dataset. Used directly in permutation mode; also used to extract N for the per-subgroup approximation. Can be NULL if <code>null_fn</code> is supplied and <code>n_sample</code> is given explicitly.
fs_params	Named list. All <code>forestsearch()</code> arguments except <code>hr.threshold</code> and <code>hr.consistency</code> (which are set to <code>c1/c2</code>) and <code>df.analysis</code> (supplied separately). Must include at minimum <code>outcome.name</code> , <code>treat.name</code> , <code>id.name</code> , and — for binary/continuous/count — <code>outcome_type</code> and <code>effect_measure</code> .
nsims	Integer. Number of null replicates. Default 200. At FPR ~ 10%, <code>nsims = 200</code> gives a 95% CI half-width of ~±2.1 pp; <code>nsims = 500</code> gives ~±1.3 pp.
nsims_verify	Integer. Number of independent verification replicates. Uses a separate seed range from the calibration batch to provide an honest check of the corrected FPR. Default 0 (skip verification). Set to <code>nsims</code> or larger for production use.
null_fn	Function or NULL. If provided, called as <code>null_fn(seed)</code> and must return a data frame with the same column structure as <code>df.analysis</code> . Overrides permutation mode. Default NULL (permutation mode).
n_sample	Integer or NULL. Sample size used when <code>null_fn</code> is provided. Ignored in permutation mode (N is taken from <code>nrow(df.analysis)</code>).
n_min	Integer. Minimum subgroup size for the per-subgroup approximation P_1 . Should match <code>fs_params\$n.min</code> . Default 60.
outcome_type_approx	Character. One of "binary", "survival", "count", "continuous". Used only for computing the asymptotic P_1 approximation. If NULL (default), inferred from <code>fs_params\$outcome_type</code> .
d_eff_approx	Numeric or NULL. Effective information for P_1 . If NULL (default), estimated from <code>df.analysis</code> (or <code>n_sample</code>) using <code>d_eff_*</code> . Supply explicitly if you want to override (e.g. use an expected event count rather than observed).
prop_cens	Numeric. Censoring proportion for survival outcomes. Used only when <code>d_eff_approx</code> is not supplied and <code>outcome_type_approx = "survival"</code> . Default 0.
p_event	Numeric. Event probability for binary outcomes. Used only when <code>d_eff_approx</code> is not supplied and <code>outcome_type_approx = "binary"</code> . Default 0.5.
sigma_y	Numeric. Residual SD for continuous outcomes. Default 1.
sim_seed	Integer. Base random seed. Each replicate uses <code>sim_seed + i</code> . Default 42.
n_workers	Integer. Parallel workers for the foreach loop. Default 1 (sequential). Set to <code>parallel::detectCores() - 1</code> for production runs.
quiet	Logical. Suppress progress messages. Default FALSE.

Details

Two null-generation modes are supported:

null_fn provided Each replicate draws a fresh null dataset from `null_fn(seed)`. Use this when you have a parametric data generating mechanism (DGM) or want to mimic a specific study design.

df.analysis only (permutation mode) Each replicate permutes the treatment column within the analyst's actual dataset. This preserves the observed covariate and outcome distributions and is the natural H_0 reference for a real analysis. Requires the `treat.name` column to be present in `df.analysis`.

Permutation mode:

Under the null hypothesis of no subgroup heterogeneity, the treatment assignment is exchangeable with the observed outcomes. Permuting `treat.name` within each replicate breaks any true treatment-by-subgroup interaction while preserving the marginal distributions of all other variables. This is a model-free H_0 reference that adapts automatically to the actual dataset structure.

P1 approximation:

The asymptotic per-subgroup P_1 at $(c1, c2)$ is computed via `compute_detection_probability_glm()` using the effective information d_{eff} at `n.min`. The ratio $L_{\text{eff}} = \log(1 - \text{FPR}) / \log(1 - P_1)$ is the implied multiplicity factor specific to this threshold pair and dataset — it replaces the universal power-law model and does not assume threshold-independence.

Value

An object of class "fpr_calibration" with components:

`fpr` Numeric. Estimated procedure-level FPR.

`fpr_ci` Numeric vector of length 2. 95% Wilson CI.

`n_found` Integer. Number of replicates where a subgroup was found.

`nsims` Integer. Total replicates attempted.

`n_failed` Integer. Replicates that errored.

`P1` Numeric. Per-subgroup approximation P_1 at $(c1, c2)$.

`d_eff` Numeric. Effective information used for P_1 .

`L_eff_implied` Numeric. Implied L_{eff} from the simulation result: $\log(1 - \text{FPR}) / \log(1 - P_1)$.

`c1, c2` Numeric. The threshold pair.

`N` Integer. Sample size.

`mode` Character. "permutation" or "dgm".

`call_args` List. Key call arguments for reproducibility.

See Also

[forestsearch](#), [compute_detection_probability_glm](#), [d_eff_binary](#), [d_eff_survival](#)

Examples

```
## Not run:
# ---- Binary example (permutation mode) ----
set.seed(1)
df <- data.frame(
  id = 1:500, treat = rbinom(500, 1, 0.5),
  bm1 = as.factor(rbinom(500, 1, 0.7)),
  bm2 = as.factor(rbinom(500, 1, 0.5)),
  age = round(rnorm(500, 55, 10)),
  y = rbinom(500, 1, 0.30) # null: no treatment effect
)

fs_params <- list(
  confounders.name = c("bm1", "bm2", "age"),
  outcome.name = "y", event.name = "y",
  treat.name = "treat", id.name = "id",
  outcome_type = "binary", effect_measure = "OR",
  adverse_outcome = TRUE,
  pconsistency.threshold = 0.90, fs.splits = 400,
  n.min = 60, maxk = 2,
  use_lasso = TRUE, use_grf = TRUE,
  is.RCT = TRUE, details = FALSE, quiet = TRUE,
  parallel_args = list(plan = "sequential", workers = 1)
)

res <- fpr_calibration(
  c1 = 1.70, c2 = 1.70,
  df.analysis = df,
  fs_params = fs_params,
  nsims = 200,
  n_min = 60, outcome_type_approx = "binary", p_event = 0.30,
  sim_seed = 42, n_workers = 4
)
print(res)

## End(Not run)
```

fpr_calibration_transfer

Transfer Corrected FPR Prediction to a Nearby Threshold Pair

Description

Uses the implied L_{eff} from a calibrated `fpr_calibration` object to predict the procedure-level FPR at $(c_1 + \delta_1, c_2 + \delta_2)$, then optionally runs a direct simulation at the new pair to validate the prediction.

Usage

```
fpr_calibration_transfer(
  cal_result,
  delta = c(0.05, 0.1, 0.2),
  delta_c1 = NULL,
  delta_c2 = NULL,
  nsims_check = 200L,
  ...
)
```

Arguments

<code>cal_result</code>	An object of class "fpr_calibration" produced by <code>fpr_calibration</code> . Must contain a finite <code>L_eff_implied</code> .
<code>delta</code>	Numeric scalar. Added to both c_1 and c_2 . Positive values tighten both thresholds; negative values relax them. Ignored if <code>delta_c1</code> or <code>delta_c2</code> are supplied.
<code>delta_c1</code>	Numeric or NULL. Increment for c_1 only.
<code>delta_c2</code>	Numeric or NULL. Increment for c_2 only.
<code>nsims_check</code>	Integer. Simulations to run at the new pair for empirical validation. Default 200. Set to 0 to skip.
<code>...</code>	Additional arguments passed to <code>fpr_calibration</code> for the validation run (e.g. <code>n_workers</code> , <code>quiet</code>).

Value

A data frame with one row per delta value containing: `c1_new`, `c2_new`, `P1_new`, `fpr_pred` (predicted corrected FPR using transferred `L_eff_implied`), `fpr_sim` (direct simulation, if `nsims_check > 0`), `fpr_sim_lo`, `fpr_sim_hi` (95% (whether `fpr_pred` falls within the simulation CI).

Examples

```
## Not run:
# Calibrate at base pair
cal <- fpr_calibration(c1 = 1.70, c2 = 1.70, null_fn = my_dgm,
  n_sample = 500, fs_params = fs_binary,
  nsims = 200, n_min = 60,
  outcome_type_approx = "binary", p_event = 0.30)

# Predict at base + 0.05, +0.10, +0.20 and validate
transfer_results <- fpr_calibration_transfer(
  cal, delta = c(0.05, 0.10, 0.20),
  nsims_check = 200, n_workers = 4)

print(transfer_results)

## End(Not run)
```

frontier_member_flags *Per-Subject Membership Matrix for Pareto Frontier Members*

Description

For each candidate subgroup on `fs$grp.consistency$out_sg$pareto_frontier`, returns a binary indicator (one row per subject in `fs$df.est`, one column per frontier member) marking which subjects belong to that subgroup.

Usage

```
frontier_member_flags(fs)
```

Arguments

`fs` A forestsearch result object.

Details

This lets the user compute downstream quantities the frontier table does not surface directly: per-frontier-member sample sizes by covariate stratum, set comparisons between frontier members ("which subjects are in the selected subgroup but not on the runner-up?"), and survival summaries by frontier member.

Value

A list with three elements:

`flags` Integer matrix of dimension `n_subjects` x `n_frontier_members`, with 1 = subject in subgroup, 0 = not.

`member_labels` Character vector of length `n_frontier_members`, each entry the "cut1 & cut2 & ..." definition of that frontier member.

`is_selected` Logical vector of length `n_frontier_members`, marking which member is the selected subgroup.

See Also

[pareto_frontier_table](#), [compute_frontier_cis](#), [plot_pareto_frontier](#).

Examples

```
## Not run:
fmf <- frontier_member_flags(fs)
# How many subjects in each frontier member?
colSums(fmf$flags)
# Which subjects are unique to the selected subgroup?
sel <- which(fmf$is_selected)
rowSums(fmf$flags) == 1L & fmf$flags[, sel] == 1L
```

```
## End(Not run)
```

fs_assert_membership	<i>Assert the exported selected-subgroup membership matches forestsearch's own</i>
----------------------	--

Description

Integrity check for `fs_export_maxeff_family()`: the materialized membership column of the selected candidate must equal, row for row, the per-subject membership forestsearch itself computed (`fit$grp.consistency$sg.harm.id`). A mismatch means the label-to-dummy reconstruction is wrong, so this stops with an error rather than letting a silently different subgroup be de-biased. `fs_to_guohe()` runs this check by default (`verify = TRUE`).

Usage

```
fs_assert_membership(exp, fit)
```

Arguments

<code>exp</code>	The list returned by <code>fs_export_maxeff_family()</code> .
<code>fit</code>	The forestsearch object exp was exported from.

Value

TRUE, invisibly, if the memberships agree; otherwise an error is thrown.

See Also

`fs_export_maxeff_family()`, `fs_to_guohe()`

Examples

```
set.seed(11)
n <- 200
df <- data.frame(id = seq_len(n), treat = rbinom(n, 1, 0.5),
                 age = round(rnorm(n, 55, 12)),
                 bm = factor(rbinom(n, 1, 0.5)))
harm <- df$age <= 50 & df$bm == "1"
tt <- rexp(n, 0.05 * exp(log(2.5) * df$treat * harm))
df$time <- pmin(tt, 60)
df$event <- as.integer(tt <= 60)
fit <- forestsearch(
  df.analysis = df,
  outcome.name = "time", event.name = "event",
  treat.name = "treat", id.name = "id",
  confounders.name = c("age", "bm"),
```

```

sg_focus = "maxeff", use_grf = FALSE, use_lasso = FALSE,
fs_splits = 50, n.min = 30, d0.min = 5, d1.min = 5,
hr.threshold = 1.1, hr.consistency = 1.0,
pconsistency.threshold = 0.5, maxk = 2,
details = FALSE, plot.sg = FALSE,
parallel_args = list(plan = "sequential", workers = 1L,
                      show_message = FALSE)
)
exp_fam <- fs_export_maxeff_family(fit)
fs_assert_membership(exp_fam, fit)

```

```
fs_declaration_calibration
```

Calibrated declaration threshold and family-wise size of the p-star screen

Description

Post-hoc, opt-in, reported diagnostics of the consistency screen, computed from multiplier draws the package already produces. The screen admits a candidate g when its consistency rate, rounded to `pconsistency.digits`, is at least `p_star`; on the resample path that is exactly the standardized statistic $T(g) = (\text{beta_hat}(g) - c_cons) / \text{sigma_D}(g)$ clearing the effective cutoff $z_pstar = \text{qnorm}((1 + pcons_eff) / 2)$ (see the Rounded admission rule section). This function reports how often the maximum of that statistic over the candidate family would clear the screen's cutoff under the null perturbation law (`fw_size`), and the cutoff that would hold the family-wise declaration rate at α (`kappa_hat`).

Usage

```

fs_declaration_calibration(
  fit,
  alpha = 0.05,
  family = c("prereduction", "reduced"),
  ...,
  c0 = NULL
)

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

```

Arguments

<code>fit</code>	A forestsearch object fitted with the declaration field retained, or the list returned by <code>fs_mr_inference()</code> with <code>keep_declaration_field = TRUE</code> .
<code>alpha</code>	Target family-wise declaration rate, in (0, 1). Default 0.05.
<code>family</code>	"prereduction" (default) or "reduced"; see the Family section.

...	Only p_star and c_cons are read, and only when the fit's resolved admission set carries no consistency floor (for example sg_focus = "maxeff"); they then name the hypothetical screen whose size is evaluated. c_cons is on the comparison scale (log for ratio measures).
c0	NULL (default) or a numeric vector of clinically specified, pre-specified protected null levels; see the Protected null level section. Named only (it follows ...).
x	An fs_declaration_calibration object.

Details

It reads a fitted object and returns a new one. It does not modify its input, does not re-run the search, does not call the consistency engine, and does not change what the search admitted: admitted_calibrated is the set the calibrated rule *would* admit, reported beside admitted_current.

Value

An object of class fs_declaration_calibration: a list with kappa_hat, fw_size, alpha, p_star, digits, digits_source, pcons_eff, z_pstar (the effective z cutoff of the rounded screen), consistency_method, consistency_method_source, c_cons, c_screen, B, multiplier_law, quantile_type, family_source, family_label, n_family_prereduction, n_family_reduced, admitted_current (the candidates the executed screen admitted; NULL when the fit is a bare fs_mr_inference() result), admitted_calibrated, admitted_pstar (the relabelled current rule over the same family), screened (the candidates the consistency screen evaluated, when known), Mstar, beta_hat, sigma_D, T_hat, column_sd, zstar_mean, field_cor (families of at most 8 candidates, else NULL), and a reduction list recording the replay of the near-duplicate reduction. When c0 is given it also carries c0, a list with table (one row per c0: c0, c0_cmp, kappa_hat, fw_size, pstar_settable, pstar_achievable, pcons_eff_settable, z_eff_settable, z_gap, digits_fine, pstar_fine, z_gap_fine, n_admitted_calibrated, the 0.90 / 0.95 / 0.99 quantiles of Mstar_c0, and is_c2), admitted_calibrated (a list indexed by c0), Mstar_c0 and source ("capture" or "field_matrix"). Every other element is unchanged by c0.

Definitions

With shared multipliers $xi[b, i]$ (one vector per draw b , reused across every candidate) and the dfbeta influence $db[g, i]$:

- $Zstar[b, g] = \sum_i xi[b, i] * db[g, i] / \sigma_D(g)$, with $\sigma_D(g)^2 = \sum_i db[g, i]^2$ – the robust scale the screen itself uses, never a model-based standard error.
- $Mstar[b] = \max_g Zstar[b, g]$ over the family: one-sided, in the harm direction.
- $kappa_hat(alpha)$ is the empirical (type = 1) $1 - \alpha$ quantile of Mstar.
- $fw_size = \text{mean}(Mstar > z_pstar)$, $z_pstar = qnorm((1 + pcons_eff) / 2)$. Its threshold is the fit's own p_star at the fit's own pconsistency.digits, not alpha: it is the family-wise size of the screen as implemented, not of the calibrated rule.
- The calibrated admission rule is $\beta_hat(g) \geq \max(c_screen, c_cons + kappa_hat * \sigma_D(g))$, the current rule with z_pstar replaced by kappa_hat.

Rounded admission rule

The screen admits on $\text{round}(P_{\text{cons}}, \text{digits}) \geq p_{\text{star}}$, with digits the fit's `pconsistency.digits`. With p_{star} rounded up to the 10^{digits} grid (g), that is $P_{\text{cons}} \geq p_{\text{cons_eff}} = g - 0.5 \times 10^{\text{digits}}$, a bar below p_{star} itself (0.895 for $p_{\text{star}} = 0.90$, $\text{digits} = 2$). On the resample path $P_{\text{cons}} = 2 \times \text{pnorm}(T) - 1$, so the screen is $T \geq z_{\text{pstar}} = \text{qnorm}((1 + p_{\text{cons_eff}}) / 2)$. `fw_size`, `admitted_pstar` and the settable p^* are computed at that effective threshold; `fw_size` is therefore the family-wise size of the screen as implemented and depends on `pconsistency.digits`. `kappa_hat` does not: it is a quantile of the maximum statistic and is compared with T directly, with no rounding.

`digits` is read from the fit's `args_call_all$pconsistency.digits`; when absent (a bare `fs_mr_inference()` result, or an older fit) the `subgroup.consistency()` default of 2 is used. `digits_source` records which.

The correspondence with T holds only under `consistency_method = "resample"`. Under "split", P_{cons} is a split proportion k / n_{valid} and admission is not a threshold on T , so `fw_size`, `z_pstar`, `admitted_pstar` and the settable- p^* columns are NA / NULL rather than computed; `kappa_hat` and the calibrated rule are still returned. The method is read from `args_call_all$consistency_method`; a bare `fs_mr_inference()` result is treated as `resample` (the field is the closed-form statistic), recorded in `consistency_method_source`.

Family

`family = "prereduction"` (the default, and the only value meant for the calibrated rule) takes the maximum over the multiplier-resampling family: every enumerated candidate of at most `maxk` factors meeting the size minimum, before any reduction keyed on fitted quantities. The near-duplicate reduction keys on sample-fitted summaries, so a family reduced by it is outcome-dependent; the pre-reduction family is covariate-measurable, and is also the conservative choice (a larger family raises the maximum). The per-arm event minima are not replayed in that family, which makes it a superset of the one the search chose among.

`family = "reduced"` removes the candidates the near-duplicate reduction removed and is a diagnostic only, so the gap is measurable. Its result is conditional on the realized family and is labelled so; it is never meant for admission. It needs a `forestsearch` fit (the reduction lives in the identifier) and the full field matrix (`keep_field_matrix = TRUE`).

Protected null level

`c1` and `c2` state the claim; `c0` states what the claim is protected against: a clinically specified, pre-specified benefit level (for example HR 0.75). The family-wise declaration rate is then controlled at α whenever every candidate's true effect is at least as good as `c0`, rather than only when every candidate sits at `c2`. With $\text{delta}_g = (c_{\text{cons}} - c_{0\text{cmp}}) / \text{sigma}_D(g)$:

- `Mstar_c0[b] = max_g { Zstar[b, g] - delta_g }`;
- `kappa_hat(c0)` is its empirical (`type = 1`) $1 - \alpha$ quantile;
- `fw_size(c0) = mean(Mstar_c0 > z_pstar)`, at the rounded admission rule;
- admission is unchanged in form, $T(g) \geq \text{kappa_hat}(c0)$;
- guidance on what to set to run `kappa_hat(c0)` as a p^* screen at the fit's `digits`: `pstar_settable`, the smallest p^* on the 10^{digits} grid whose effective threshold is at or above `kappa_hat(c0)`; that threshold as `pcons_eff_settable` (P_{cons} scale) and `z_eff_settable` (z scale); `z_gap =`

$z_{\text{eff_settable}} - \kappa_{\text{hat}}(c_0)$ (positive = conservative); and $\text{digits_fine} / \text{pstar_fine} / z_{\text{gap_fine}}$, the smallest digits (searched over 1 to 12) at which the gap falls below 0.01, with its p^* . When no $p^* \leq 1$ reaches $\kappa_{\text{hat}}(c_0)$ at the fit's digits, pstar_achievable is FALSE and the settable columns are NA; the print method says so. This is guidance on what to set, not an identity: the settable screen is $\kappa_{\text{hat}}(c_0)$ rounded up to what the grid can express, so it admits a subset of what $\kappa_{\text{hat}}(c_0)$ admits.

If every candidate's true effect is c_0_{cmp} , $T(g)$ is centred at $-\text{delta_g}$, so the null law of $\max_g T(g)$ is that of the shifted maximum. At $c_0 = c_2$, $\text{delta_g} = 0$ and every quantity is the unshifted one.

c_0 is on the natural scale of the consistency threshold c_2 (the HR on the survival path; the ratio for OR / RR / IRR; the difference for RD / MD) and is mapped exactly as c_2 is: $\log()$ for ratio measures, identity otherwise. $c_0 \leq c_2$ is required. A non-positive c_0 on a ratio path (a log supplied by mistake) errors; on an identity path a wrong-scale c_0 cannot be detected.

The shifted maxima are read from the capture when the fit carries them for every requested c_0 (declaration_c_0 at fit time); otherwise they are computed from the stored field matrix ($\text{keep_field_matrix} = \text{TRUE}$); otherwise the call errors. It never falls back to the unshifted maximum. `family = "reduced"` always computes from the field matrix.

Inputs

The field is retained only on request. Fit with `mr_inference = TRUE` and `mr_inference_args = list(keep_declaration_field = TRUE)` (add `keep_field_matrix = TRUE` for the reduced diagnostic), or call `fs_mr_inference(..., keep_declaration_field = TRUE)` and pass its result. The multiplier law and B are those of the MR draws and are recorded in the result.

See Also

[fs_mr_inference\(\)](#), [fs_family_report\(\)](#), [fs_fdr_report\(\)](#).

Examples

```
## Not run:
fit <- forestsearch(df, ..., mr_inference = TRUE,
                   mr_inference_args = list(keep_declaration_field = TRUE))
fs_declaration_calibration(fit, alpha = 0.05)

## End(Not run)
```

Description

Draws replicates through the DGM's own generator and reports, per sample size, how often the planted harm region Q would be **undeclarable** (too small for the search's own size test), how often it falls under the per-arm events floor, and how often the analysis estimand does not exist on it.

Usage

```
fs_dgm_feasibility(
  dgm,
  n = c(500, 750, 1000, 2000),
  n.min = 60,
  d0.min = 10,
  d1.min = 10,
  n_rep = 200L,
  tolerance = 0.05,
  effect_measure = NULL,
  seed = NULL,
  rand_ratio = 1
)

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

Arguments

dgm	An object of class "glm_dgm" – the interface the operating-characteristics family accepts (fs_oc_predict , fs_oc_grid , which validate dgm\$df_super and its flag_harm column). Survival DGMs are not accepted; see Note.
n	Numeric vector of sample sizes to evaluate.
n.min, d0.min, d1.min	The search's own floors, passed so the report describes the campaign that will actually run. Defaults are the package defaults (60, 10, 10). Nothing is imposed: these are read, not applied.
n_rep	Integer. Replicates per sample size (default 200).
tolerance	Numeric. The largest undeclarable share that still counts as feasible (default 0.05).
effect_measure	Character or NULL. The estimand whose existence condition is checked. NULL (default) reads dgm\$effect_measure.
seed	Integer or NULL. Base seed; replicate i of sample size n is drawn at seed + i.
rand_ratio	Numeric. Passed to <code>simulate_from_glm_dgm()</code> .
x	An <code>fs_dgm_feasibility</code> object.
...	Ignored.

Details

The motivating case: at $n = 500$ a region averaging 48.5 subjects is at or below $n.min = 60$ in 96\ structurally unable to recover a region of the planted size – what it declares is a larger overlapping region. That is a property of the design, knowable before any replicate is run, and this function is how to know it.

Value

An object of class "fs_dgm_feasibility": a list with \$table (one row per n), \$feasible (a single logical – TRUE when every undeclarable share is at or below tolerance), and \$args.

What is counted

Per n, over n_rep draws:

share_undeclarable share with $|Q| \leq n.min$. The comparison is $\leq$, exactly as subgroup.search() tests it (nx $\leq$ n.min rejects), so a region must *exceed* n.min to be declarable.

share_under_events share with fewer than d0.min events in the control arm or d1.min in the treated arm. Binary and survival only – the floor is skipped entirely for continuous and count outcomes.

share_nonestimable share on which the estimand does not exist, under the same per-estimand condition the estimator boundary applies: OR needs all four cells ≥ 1 ; RR, IRR and HR need an event in each arm; RD, IRD and MD have no condition and this share is always 0.

Together with per-arm cell summaries (mean, 5th and 95th percentile, and minimum of each cell across replicates).

Drawing through the DGM's own generator

Replicates are drawn with `simulate_from_glm_dgm()` – the same function the campaign templates call – never a re-implementation, so the population this reports on is the population a campaign would run on.

This function **does not change the RNG kind**. It seeds each replicate with `set.seed()` under whatever generator is current and restores the caller's RNG state on exit. The caller's own obligation is the mirror of that one: **build and calibrate the DGM before any replicate switches the RNG kind**. Calibrating a DGM after a switch to "L'Ecuyer-CMRG" yields a *different* super-population, and every count here would then describe a population no campaign ever used.

Note

Survival DGMs are not supported, and what they would need is this: the OC family's contract is `inherits(dgm, "glm_dgm")` with a `df_super` carrying `flag_harm`. `setup_gbsg_dgm()` produces a different object whose own generator is `simulate_from_dgm(dgm, n, analysis_time, cens_adjust, seed)` – a different signature, and it takes two censoring arguments that have no GLM counterpart and that change the per-arm event counts this function reports. Supporting it means a generator-dispatch layer plus passing `analysis_time` and `cens_adjust` through; that is a separate change and is deliberately not guessed at here.

See Also

`simulate_from_glm_dgm`, `fs_oc_predict`

Examples

```
set.seed(1)
N <- 400
d <- data.frame(treat = rbinom(N, 1, 0.5),
               age = round(rnorm(N, 55, 12)),
               sex = factor(rbinom(N, 1, 0.5), levels = 0:1))
d$y <- rbinom(N, 1, plogis(-0.5 + 0.3 * d$treat))
dgm <- generate_glm_dgm(data = d, factor_vars = "sex",
                      continuous_vars = "age", outcome_var = "y",
                      treatment_var = "treat", outcome_type = "binary",
                      effect_measure = "OR", subgroup_vars = "age",
                      subgroup_cuts = list(age = list(type = "greater",
                                                       quantile = 0.55)),
                      k_inter = 1, n_super = 2000, seed = 8316951)
feas <- fs_dgm_feasibility(dgm, n = c(500, 1000), n_rep = 50)
feas
feas$feasible
```

fs_dgm_scale

Sampling Scale of the Difference-in-Means Estimator

Description

Computes the finite-population sampling scale of the within-region difference-in-means estimator directly from a data generating mechanism's individual-level potential outcomes.

Usage

```
fs_dgm_scale(
  dgm,
  regions = NULL,
  harm_col = "flag_harm",
  rand_ratio = 1,
  labels = c("Q", "Qc", "S")
)
```

Arguments

dgm	An object of class "glm_dgm" from generate_glm_dgm .
regions	Optional named list of region specifications. Each element is either a logical vector of length <code>nrow(dgm\$df_super)</code> or an integer vector of row indices. Defaults to <code>NULL</code> , which uses the true harm region, its complement, and the whole super-population.
harm_col	Character. Name of the true-region indicator column in <code>df_super</code> . Default "flag_harm".

rand_ratio	Numeric. Treatment:control randomisation ratio, so the randomisation probability is $\text{rand_ratio} / (1 + \text{rand_ratio})$. Default 1 (1:1), matching simulate_from_glm_dgm .
labels	Optional character vector of length 3 renaming the default regions. Ignored when regions is supplied. Default <code>c("Q", "Qc", "S")</code> .

Details

For a region g the estimator is the within-region difference in arm means. Conditioning on the realized arm counts,

$$\text{Var}[\hat{\beta}(g)] = V_{\text{eff}}(g)/(nP(g)),$$

with the region- and n -free constant

$$V_{\text{eff}}(g) = V_1(g)/p + V_0(g)/(1-p),$$

where p is the randomisation probability and $V_w(g)$ is the total within-arm outcome variance in g under arm w :

$$V_w(g) = m_g[v_w] + V_g[\mu_w].$$

Here μ_w is the arm mean surface, $v_w(x) = \text{Var}(Y \mid X = x, W = w)$ the conditional outcome variance, and $m_g[\cdot]$, $V_g[\cdot]$ finite-population moments over the rows of the super-population lying in g . The conditional variance is supplied by the outcome family:

continuous $v_w = \sigma^2$, the residual variance (constant).

binary $v_w = p_w(1 - p_w)$.

count $v_w = \mu_w$ (Poisson).

Value

An object of class `c("fs_dgm_scale", "list")`:

regions Data frame, one row per region, with columns `region`, `n_g`, `P_g`, `m_mu0`, `m_mu1`, `m_tau`, `V_mu0`, `V_mu1`, `V_tau`, `C_mu0_tau`, `v_cond0`, `v_cond1`, `V_arm0`, `V_arm1`, `bracket`, `V_eff`.

sigma Residual standard deviation for continuous outcomes, otherwise `NA_real_`.

outcome_type, **effect_measure** Copied from `dgm`.

rand_ratio, **p_treat** Allocation used.

n_super Super-population size.

Balanced-arm reading

At 1:1 allocation $V_{\text{eff}}(g) = 2\{V_0(g) + V_1(g)\}$, and `bracket` $= V_{\text{eff}}(g)/4$ admits the decomposition

$$\sigma^2 + V_g[\mu_0] + C_g[\mu_0, \tau] + \frac{1}{2}V_g[\tau],$$

with $\tau = \mu_1 - \mu_0$ the conditional average treatment effect. The component columns are returned for every region and every outcome type; the decomposition above is exact for continuous outcomes at 1:1 allocation. Away from 1:1, `bracket` remains $V_{\text{eff}}/4$ but no longer equals that sum, so prefer `V_eff` for any downstream calculation.

Idealisations

V_eff treats the region size and the arm split as fixed at their expectations. A trial draws $n_g \sim \text{Bin}(n, P(g))$ and $n_1 | n_g \sim \text{Bin}(n_g, p)$, and by Jensen's inequality $E[1/n_1 + 1/n_0]$ and $E[1/n_g]$ exceed their fixed-count counterparts. Use `fs_scale_se` with `jensen = TRUE` for the unconditional standard deviation.

Scope

Exact for identity-scale effect measures ("MD", "RD", "IRD"), where the estimator is a difference in means. Ratio measures ("OR", "RR", "IRR") require a delta-method layer and are rejected rather than silently approximated.

See Also

`fs_scale_se`, `generate_glm_dgm`

Examples

```
## Not run:
dgm <- generate_glm_dgm(..., outcome_type = "continuous",
                        effect_measure = "MD")
sc <- fs_dgm_scale(dgm)
sc$regions[, c("region", "P_g", "bracket", "V_eff")]

# Standard deviation of the estimator on the true region at n = 500
fs_scale_se(sc, n = 500, region = "Q")

# Arbitrary regions, e.g. a candidate family
fs_dgm_scale(dgm, regions = list(big = dgm$df_super$age > 40))

## End(Not run)
```

fs_export_maxeff_family

Materialize the maxeff candidate family as 0/1 membership columns

Description

Reconstructs, for every candidate in the deduplicated maxeff family kept on a `forestsearch(sg_focus = "maxeff")` fit, its per-subject 0/1 membership on the estimation frame `fit$df.est`. Each candidate's cut labels (e.g. "`{er <= 0}`", or a negated cut "`!{size <= 20}`") are mapped back to the aligned dummy columns of `fit$df.est` via `fit$confounders.evaluated / fit$confounders.candidate`, and a candidate's membership is the intersection (AND) over its cuts. The result is the materialized family that `guohe_algorithm3()` consumes; `fs_to_guohe()` calls this internally.

Usage

```
fs_export_maxeff_family(fit, prefix = "sg_")
```

Arguments

fit	A forestsearch object fitted with sg_focus = "maxeff".
prefix	Character prefix for the generated membership column names (sg_0001, sg_0002, ... by default).

Value

A list with elements data (fit\$df.est with one appended 0/1 membership column per candidate), candidates (the appended column names), selected (the column name of the forestsearch-selected candidate; the family table is sorted selected-first), and family (the candidate family table fit\$grp.consistency\$out_sg\$result).

See Also

[fs_to_guohe\(\)](#), [fs_assert_membership\(\)](#)

Examples

```
set.seed(11)
n <- 200
df <- data.frame(id = seq_len(n), treat = rbinom(n, 1, 0.5),
                 age = round(rnorm(n, 55, 12)),
                 bm = factor(rbinom(n, 1, 0.5)))
harm <- df$age <= 50 & df$bm == "1"
tt <- rexp(n, 0.05 * exp(log(2.5) * df$treat * harm))
df$time <- pmin(tt, 60)
df$event <- as.integer(tt <= 60)
fit <- forestsearch(
  df.analysis = df,
  outcome.name = "time", event.name = "event",
  treat.name = "treat", id.name = "id",
  confounders.name = c("age", "bm"),
  sg_focus = "maxeff", use_grf = FALSE, use_lasso = FALSE,
  fs.splits = 50, n.min = 30, d0.min = 5, d1.min = 5,
  hr.threshold = 1.1, hr.consistency = 1.0,
  pconsistency.threshold = 0.5, maxk = 2,
  details = FALSE, plot.sg = FALSE,
  parallel_args = list(plan = "sequential", workers = 1L,
                       show_message = FALSE)
)
exp_fam <- fs_export_maxeff_family(fit)
head(exp_fam$candidates)
exp_fam$selected
```

fs_family_report

Report which stages of the candidate family are data-dependent

Description

A user who sets `use_lasso = FALSE`, `use_grf = FALSE`, `use_dina = FALSE` and `vi.grf.min = NULL` may reasonably believe the candidate family `forestsearch` searches is now fixed. It is not. Continuous cuts are placed at *sample* quantiles (`get_FSdata()`); the prevalence, redundancy and size floors (`minp`, `rmin`, `n.min`) are applied to *sample* counts (`subgroup.search()`); and the consistency stage removes near-duplicate candidates keyed on their fitted *statistics*, with no argument that turns it off (`remove_near_duplicate_subgroups()`). Those stages are the method, not screening. This function says so, for a given argument set, in a form that cannot overpromise.

Usage

```
fs_family_report(x, data = NULL, outcome_type = NULL)
```

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

Arguments

<code>x</code>	Either a named list of <code>forestsearch</code> arguments (as one would pass to <code>do.call(forestsearch, x)</code> ; unspecified arguments take <code>forestsearch()</code> 's formal defaults), or a fitted <code>forestsearch</code> object, in which case its <code>args_call_all</code> is read.
<code>data</code>	Optional data frame. When supplied, the report is grounded in counts: the number of cut columns <code>get_FSdata()</code> produces on this data under these arguments, and the number of up-to-maxk combinations that implies. Nothing is fitted; the search is not run.
<code>outcome_type</code>	One of "survival", "binary", "continuous", "count". Taken from <code>x</code> when present there; required otherwise, because the per-arm floor row depends on it.
<code>...</code>	Ignored.

Details

It reports and changes nothing. It never fits a model, never runs the search, and is called by nothing else in the package. **No combination of arguments makes the family deterministic while cuts are placed at sample quantiles;** the best a caller can do is switch off the disableable stages, and the printed footer lists what remains.

The report mirrors `forestsearch()`'s own argument resolution rather than reading arguments at face value: `sg_focus` aliases are normalised; `sg_focus = "maxeff"` zeroes `minp`, `rmin` and `pconsistency.threshold`, sets `stop_threshold` to `NULL`, `use_twostage` to `FALSE` and `max_subgroups_search` to `Inf`, and disables the effect floor; `stop_threshold` is reset to `NULL` for every focus other than "maxeffCons" (it is meaningful only there); `max_n_confounders` is applied only inside the GRF variable-importance block, so it is inert when `vi.grf.min` is `NULL`; and the per-arm event floors `d0.min` / `d1.min` are skipped for continuous and count outcomes.

Value

A data frame of class `c("fs_family_report", "data.frame")` with one row per stage and columns stage, arguments (the governing `forestsearch()` formals), values (their resolved values, formatted), status (one of "deterministic", "disabled", "inert", "data-dependent", "data-dependent (not disableable)") and note (one line: why, and what would change it; NA where nothing would). Attributes: `verdict` (one sentence), `status_counts` (a named integer vector), `data_supplied` (logical), and – when data is supplied – `n_cut_columns` and `n_combinations`.

See Also

[forestsearch](#), [fs_oc_family_enumerate](#) (the population-frame enumeration used by the operating-characteristic wrapper, which is deterministic in the DGM precisely because it does not cut at sample quantiles).

Examples

```
rep <- fs_family_report(
  list(confounders.name = c("age", "preanti", "hemo"),
    conf.cont_jcuts = list(age = 10, preanti = 10),
    n.min = 60, maxk = 2, sg_focus = "maxeffCons",
    use_lasso = FALSE, use_grf = FALSE, use_dina = FALSE,
    vi.grf.min = NULL),
  outcome_type = "continuous")
print(rep)
attr(rep, "verdict")

# grounded in counts: cut columns and combinations on a data frame
set.seed(1)
df <- data.frame(age = round(rnorm(200, 40, 8)),
  preanti = round(rexp(200, 1 / 500)),
  hemo = rbinom(200, 1L, 0.1))
rep2 <- fs_family_report(
  list(confounders.name = c("age", "preanti", "hemo"),
    conf.cont_jcuts = list(age = 10, preanti = 10), maxk = 2,
    use_lasso = FALSE, use_grf = FALSE, vi.grf.min = NULL),
  data = df, outcome_type = "continuous")
attr(rep2, "n_cut_columns"); attr(rep2, "n_combinations")
```

Description

Characterizes the operating false-discovery rate (FDR) of a **fitted forestsearch** analysis and tests whether requiring harm confirmation on the de-biased effect reduces it. This is a **measurement** tool, not a calibration tool: it does not solve for thresholds that control an FDR target. For the (c_1, c_2) pair the analysis actually used, it reports two quantities under a null:

Usage

```
fs_fdr_report(
  fs_analysis,
  df_analysis,
  null = c("hom", "harm"),
  c_confirm = c(0.9, 1, 1.25),
  nsims = 1000L,
  n = NULL,
  confounders = NULL,
  n_super = 25000L,
  analysis_time = 84,
  cens_adjust = log(1.5),
  event_min = 5L,
  n_workers = NULL,
  seed_base = 8316951L,
  quiet = FALSE
)
```

Arguments

<code>fs_analysis</code>	A fitted <code>forestsearch()</code> object. Must carry <code>args_call_all</code> (the stored call arguments), including <code>hr.threshold</code> and <code>hr.consistency</code> .
<code>df_analysis</code>	The analyst's data frame. Used only to default the simulated sample size (<code>n = nrow(df_analysis)</code>); the null data are generated from the GBSG-AFT DGM, not resampled from this frame.
<code>null</code>	Character vector; which null(s) to simulate. "hom" is the homogeneous null (<code>k_inter = 0</code> : a real, uniform treatment effect, no modifier); "harm" is the null-harm-region null (<code>target_hr_harm = 1.0</code> : a real modifier tuned so the flagged region has HR = 1.0 while treatment stays active elsewhere). Default both.
<code>c_confirm</code>	Numeric vector of de-biased-HR harm-confirmation thresholds at which to report B, i.e. the values <code>t_confirm</code> is swept over. Default <code>c(0.9, 1.0, 1.25)</code> . On real data harm confirmation is inert; here it annotates the null's operating characteristics only.
<code>nsims</code>	Integer; null replicates per null. Default 1000.
<code>n</code>	Integer or NULL; simulated trial size. Default <code>nrow(df_analysis)</code> . The published reference baseline used <code>n = 700</code> ; pass that to reproduce it exactly.
<code>confounders</code>	Character vector or NULL; candidate covariates. Default NULL inherits <code>confounders.name</code> from the fitted object (intersected per replicate with the DGM columns), falling back to the GBSG covariate set when the analysis carried none.
<code>n_super</code>	Integer; super-population size for the DGM. Default 25000.
<code>analysis_time</code>	Numeric; administrative censoring time (months). Default 84.
<code>cens_adjust</code>	Numeric; censoring shift passed to <code>simulate_from_dgm()</code> . Default <code>log(1.5)</code> .
<code>event_min</code>	Integer; minimum events for a replicate to be scored. Replicates below this (or with a single treatment arm) are counted as visible failures, never silently dropped. Default 5.

n_workers	Integer or NULL; parallel workers for the outer replicate loop. Default NULL uses ~90% of physical cores. The inner forestsearch() call always runs sequentially (parallel_args = list(plan = "sequential")) to avoid nested parallelism, which breaks forestsearch.
seed_base	Integer; base seed. Each null uses a disjoint seed block and each replicate an explicit seed, so results are plan-invariant. Default 8316951.
quiet	Logical; suppress progress and per-null messages. Default FALSE.

Details

- **A** (raw FDR): $A = P(\text{declare any harm subgroup} \mid c_1, c_2)$.
- **B(g)** (confirmed FDR): $B(g) = P(\text{declare AND de-biased } HR \geq g)$. Because harm confirmation can only remove declarations, $B(g) \leq A$ always; the gap $A - B(g)$ is what the de-biasing step buys at confirmation threshold g .

All thresholds and identifier knobs are inherited from `fs_analysis$args_call_all`, so the FDR reported is that of the analysis as run. Multiplier resampling is forced on internally (`mr_inference = TRUE`) regardless of the fitted setting, because B requires the per-replicate de-biased HR.

What "false discovery" means here:

Under "hom" the treatment effect is real but uniform, so any declared subgroup is a manufactured heterogeneity – the winner's-curse-of-search rate. Under "harm" a real modifier exists and the flagged region sits exactly at $HR = 1.0$, so a small upward fluctuation reads as harm; this is the more adversarial null. The declared cut need not overlap `flag_harm`: under either null any declaration is a false discovery.

The confirmation accounting (why `n_deb_na` matters):

$B(g)$ counts a declaration only when its de-biased HR is finite and clears g ; a declaration whose de-biased HR could not be computed counts as unconfirmed. Such cases are counted in `n_deb_na` and reported, so that $A - B$ is never mistaken for pure de-biasing benefit when part of it is de-biased-HR unavailability.

Inheritance and rebinding:

Identifier knobs (thresholds, subgroup_method, consistency_method, conf_force, conf.cont_jcuts, maxk, MR arguments, ...) transfer verbatim from `args_call_all`. The data-binding arguments – the five column names and the confounder set – are rebound to the DGM, because `simulate_from_dgm()` emits fixed names (`y_sim / event_sim / treat_sim / id / flag_harm`). `mr_inference` is forced TRUE so B is computable.

Value

An object of class "fs_fdr_report": a list with

`fdr` Data frame, one row per null x `c_confirm`, with `A`, `A_lo`, `A_hi` (raw FDR and 95% Wilson CI), `B`, `B_lo`, `B_hi` (confirmed FDR and CI), reduction ($A - B$), and the accounting columns `n_valid`, `n_declared`, `n_deb_na`, `fs_errors`, `unhandled`.

`hr_structure` Data frame of the realized hazard ratios each DGM encodes – the structural check that the nulls are different worlds.

`verdict` "CLEAN" if no replicate failed across any null, else "WARNING".

n_errors Total failed replicates (forestsearch errors + unhandled).
 c1, c2, n, nsims, nulls The inherited thresholds and run settings.
 call_args Reproducibility metadata.

See Also

[forestsearch](#) (and its vocabulary section), [fs_mr_inference](#) for the MR step this forces on, [mr_estimates_table](#), [forestsearch_bootstrap_dofuture](#) for the full bootstrap, [fpr_calibration](#), [setup_gbsg_dgm](#), [calibrate_k_inter](#), [simulate_from_dgm](#)

Examples

```
## Not run:
# A fitted survival/Cox analysis (thresholds and knobs live on the object)
fit <- forestsearch(
  df.analysis = gbsg_df,
  outcome.name = "time", event.name = "status",
  treat.name = "treat", id.name = "id",
  confounders.name = c("er", "age", "meno", "pgr", "nodes", "size", "grade"),
  is.RCT = TRUE, est.scale = "hr",
  subgroup_method = "consistency", consistency_method = "resample",
  hr.threshold = 1.25, hr.consistency = 1.0,
  pconsistency.threshold = 0.90, maxk = 2,
  mr_inference = TRUE)

# Characterize its operating FDR and whether harm confirmation helps
rep <- fs_fdr_report(fit, gbsg_df, nsims = 1000, n_workers = 100)
rep                                     # prints the realized-HR table and the A/B table
rep$fdr                               # one row per null x c_confirm

# Reproduce the published reference (n = 700): A ~ 0.138 (hom), 0.278 (harm)
fs_fdr_report(fit, gbsg_df, n = 700, nsims = 1000, n_workers = 100)

## End(Not run)
```

fs_focus_tag

Stem tag for a (subgroup_method, sg_focus) pair

Description

Returns the short label used in result-file stems and plot annotations for the rule that a given (subgroup_method, sg_focus) combination actually runs – not the spelling the caller passed. The two differ whenever an alias or a behavioural synonym is in play, which is precisely when a hand-written tag goes wrong.

Usage

```
fs_focus_tag(subgroup_method, sg_focus)
```

Arguments

subgroup_method	Character scalar. One of "consistency", "dina", "grf". Any value other than "consistency" takes the effect-argmax collapse.
sg_focus	Character scalar. Any accepted sg_focus spelling, alias or canonical.

Details

The collapse is engine-specific:

"consistency" eff, hr and maxcons all tag as "maxcons" – they name the consistency argmax. maxeff and maxeffCons stay distinct, because the consistency floor genuinely separates them here.

"dina", "grf" eff, hr, maxcons, maxeff and maxeffCons all tag as "eff". Neither engine computes a Pcons, so the consistency qualifier has nothing to bind to and all five rank by order(-eff).

The band foci collapse the same way on every engine: hrMaxSG tags as "effMaxSG" and hrMinSG as "effMinSG". maxSG and minSG are canonical foci in their own right and pass through.

Value

Character scalar; the stem tag.

See Also

[forestsearch](#) for the per-engine semantics of each focus.

Examples

```
fs_focus_tag("consistency", "eff") # "maxcons"
fs_focus_tag("dina", "eff") # "eff"
fs_focus_tag("dina", "maxeffCons") # "eff" -- not "maxeffCons"
fs_focus_tag("grf", "hrMaxSG") # "effMaxSG"
```

fs_identification_summary

Identification structure across a sweep

Description

Applies [fs_identification_structure\(\)](#) across a named collection of bundles – typically one per trial size – and returns a single object the plot and table helpers consume. This is the package-level replacement for the inline fs_identification_figures*.qmd fragments.

Usage

```
fs_identification_summary(
  bundles,
  anchor,
  partner,
  proxy,
  confounders = NULL,
  sweep_label = "Trial size (n)",
  sg_def_col = "sg_def",
  covs_col = "covs"
)
```

Arguments

bundles A named list. Each element is a results data frame, a bundle carrying results, or a path to an .rds. Names label the sweep points (for example "500" and "1000") and are used on the x axis in the order supplied.

anchor, partner, proxy Character. The planted-structure triple.

confounders Character vector. The analysis covariate pool.

sweep_label Character. Axis label for the sweep dimension.

sg_def_col, covs_col Character. Passed through.

Value

An object of class `fs_identification`: a list with `per_point` (the raw aggregates), tidy involvement and structure data frames, an accuracy data frame, and the triple it was built with.

Examples

```
## Not run:
id <- fs_identification_summary(
  list(`500` = "results/fs..._n500_combined_1_500.rds",
       `1000` = "results/fs..._n1000_combined_1_500.rds"),
  anchor = "er", partner = "meno", proxy = "age",
  confounders = c("er", "age", "meno", "pgr", "nodes", "size", "grade"))
plot_fs_identification_involvement(id)
plot_fs_identification_structure(id)
fs_identification_table(id)

## End(Not run)
```

fs_identification_table	<i>Identification-structure table</i>
-------------------------	---------------------------------------

Description

The supplement’s structure table: detection rate, the five structure shares, anchor recovery, and the conditional partner-versus-proxy split, one column per sweep point. Returned as a data frame by default so a LaTeX document can style it however it likes; as_gt = TRUE renders it with gt.

Usage

```
fs_identification_table(x, as_gt = FALSE, digits = 1)
```

Arguments

- x An fs_identification object.
- as_gt Logical. Render with gt instead of returning the data frame.
- digits Integer. Digits on the percent labels.

Value

A data frame, or a gt_tbl. The accuracy sentence the supplement carries as a table footnote is attached as the "accuracy_note" attribute of the data frame.

FS_labels	<i>Convert Factor Code to Label</i>
-----------	-------------------------------------

Description

Converts q-indexed codes to human-readable labels using the confs_labels mapping. Supports both full format ("q1.1", "q3.0") and short format ("q1", "q3"). Handles vector input via recursion.

Usage

```
FS_labels(Qsg, confs_labels)
```

Arguments

- Qsg Character. Factor code in format "q<index>.<action>" or "q<index>". For the full format, action 0 = NOT (complement), action 1 = IN (member). Short format defaults to action 1. Can also be a character vector for vectorized input.
- confs_labels Character vector. Labels for each factor, indexed by factor number.

Value

Character. Human-readable label wrapped in braces, e.g., "{age <= 50}" or "{age <= 50}" for complement. Returns the original code if no match is found.

Examples

```
## Not run:
labels <- c("age <= 50", "tumor size <= 20", "nodes <= 3")
FS_labels("q1.1", labels)
FS_labels("q1.0", labels)
FS_labels("q2", labels)
FS_labels(c("q1", "q3"), labels)

## End(Not run)
```

fs_mr_oc_summary	<i>Operating-Characteristics Summary of a Multiplier-Resampling Payload</i>
------------------	---

Description

Summarises a simulation payload produced by the multiplier-resampling (MR) harness into the estimation, coverage and classification quantities used in simulation summaries, returning plain numbers rather than a formatted table.

Usage

```
fs_mr_oc_summary(
  payload,
  estimators = NULL,
  blocks = c("H", "Hc"),
  digits = NULL
)
```

Arguments

payload	A list with results, truth and meta, as written by the MR batch harness, or a path to such an .rds file.
estimators	Character vector selecting estimators, from "oracle", "naive", "FB", "MR". Default NULL takes all present, dropping "FB" when the run carried no full bootstrap (meta\$nb_boots absent or below 1).
blocks	Character vector of blocks, from "H" and "Hc". Default both.
digits	Integer or NULL. NULL (default) returns unrounded values; a formatter should round, not the computation.

Value

An object of class `c("fs_mr_oc", "list")`:

`estimation` Data frame, one row per block and estimator: `n`, `avg`, `sd_emp`, `se_hat`, the four `bias_*`, the four `cov_*`, and `width`.

`identification` Data frame with one row per convention: `detection rate`, `mean subgroup size`, and `mean sens/spec/ppv/npv`.

`targets` The oriented target values used.

`meta` Key run metadata, echoed.

Orientation

Estimator columns (`or_`, `nv_`, `fb_`, `mr_`) are stored on the ORIENTED scale, where positive means harm. `betaHhat_H` and `betaHhat_Hc` are stored on the RAW outcome scale. The bridge is `orient = if (adverse_outcome) 1 else -1`, applied here as the simulation harness applies it. Getting this backwards silently negates every bias.

Four targets, deliberately

Every estimator is scored against every target in both blocks, so that disagreement is visible rather than assumed away:

`beta` Per-replicate exact $\beta(\hat{H})$ – the conditional estimand at the realised region.

`oracle` Per-replicate refit on the true region in that trial.

`struct` Scalar structural potential-outcome effect in the true region; zero Monte Carlo error.

`marg` Scalar fitted effect in the true region on one realised draw; carries sampling noise.

Bias is ABSOLUTE, in outcome units. On an identity scale a complement effect can sit near zero, so a relative bias would explode.

Two classification conventions

Both are returned, because the project contains both and they disagree whenever detection is not near certain:

`conditional` Averaged over DETECTED replicates only. This is the detected-only summary.

`unconditional` Averaged over ALL replicates, following `build_classification_table()` and Leon et al. (2024, Table 1), with non-detection scored as `sens = 0`, `ppv = 0`, `spec = 1`, `npv = 1`.

With detection at 999/1000 the two agree to three decimals; in a null cell at 0.930 they do not. Quote which one you mean.

See Also

[fs_dgm_scale](#)

Examples

```
## Not run:
oc <- fs_mr_oc_summary("fs_maxeffCons_mr_md40_knoise0_n500_res_1_1000.rds")
oc$estimation[oc$estimation$block == "H", c("estimator", "avg", "bias_beta")]
oc$identification

## End(Not run)
```

```
fs_oc_family_enumerate
```

Enumerate the candidate family on a DGM's population frame

Description

Builds the family of candidate subgroups that [forestsearch](#) would search over, but on the DGM's super-population frame `dgm$df_super` rather than on a sampled trial, so that every cut lands at a population quantile and every prevalence, purity and overlap is a population proportion. The result carries the nine quantities the operating-characteristics prediction of [fs_oc_predict](#) consumes.

Usage

```
fs_oc_family_enumerate(
  dgm,
  forestsearch_args,
  n,
  max_M = 2000L,
  verbose = FALSE
)
```

Arguments

<code>dgm</code>	An object of class "glm_dgm" from generate_glm_dgm , with the true-region indicator in <code>df_super\$flag_harm</code> .
<code>forestsearch_args</code>	Named list of forestsearch arguments. <code>confounders.name</code> is required; the <code>cut</code> , <code>maxk</code> , <code>minp</code> , <code>rmin</code> , <code>n.min</code> and <code>n.min.frac</code> entries are read, and any omitted entry takes <code>forestsearch()</code> 's default.
<code>n</code>	Integer. Trial size at which the family is evaluated: sets the size floor <code>n.min / n</code> and the <code>sqrt(1000 / n)</code> rescaling of the anchored standard errors.
<code>max_M</code>	Integer. Size guard: if more than <code>max_M</code> candidates survive the floors, the function stops before allocating the M-by-M overlap matrix (the downstream symmetric root is $O(M^3)$).
<code>verbose</code>	Logical. Print the enumeration counts at each stage.

Details

Cuts. The cut-related entries of `forestsearch_args` (`confounders.name`, `conf.cont_jcuts`, `cut_type`, `conf.cutoff`, `conf.cont_medians`, `conf.cont_medians_force`, `conf_force`, `defaultcut_names`, `exclude_cuts`, `collapse_cuts`, `collapse_cuts_args`) are handed to the package's own `get_FSdata()` on `df_super`. Both directions of every cut are generated, as the search does. LASSO, GRF and DINA cut sources are off: they are sample-fitted screens, not part of the cut specification.

Combinations. All combinations of up to `maxk` indicator columns are enumerated with `generate_combination_indices()` in the order `forestsearch()` uses when it composes the MR family.

Structural floors (all on the population frame, in the order of `evaluate_combination_with_status()`):

1. combinations with a constant column or an empty pairwise intersection are skipped (the search's status 0);
2. `minp`: every constituent factor must have population prevalence $\geq \text{minp}$;
3. `rmin`: each added factor must shrink the membership by more than `rmin` subjects of a trial of size `n`, i.e. by more than rmin / n in population proportion;
4. `size`: $P_g \geq n.\text{min} / n$, where `n.min` is resolved as `forestsearch()` resolves it – the supplied value (default 60), or $\max(60, \text{ceiling}(n.\text{min}.\text{frac} * n))$ when `n.min` = NULL.

The GRF variable-importance pre-screen, the effect screen, the consistency screen and the near-duplicate removal are *not* applied. Candidates whose population membership vectors are identical are collapsed to the first one enumerated.

Null DGMs. A DGM whose `df_super$flag_harm` has no member (Q empty; `generate_glm_dgm(model = "null")`) is detected structurally, cross-checked against `dgm$model` when present. Under the null every candidate has the same true effect, so the fields become: `beta_g` = the common effect (the DGM's `effect_Qc` = `effect_ITT`, oriented as the alternative path orients); `se_g` from the whole-population effective variance (`fs_dgm_scale(dgm, regions = list(S = ...))`, the S row) at (`n`, `Pg`) with the same prevalence scaling; `PQg` = 0; `sens_g` = NA (0/0, undefined – not zero); `spec_g` = $1 - P_g$; `PQ` = 0 (from which $\text{NPV} = 1$ follows downstream). Enumeration, floors, `ovl` and the covariance are unchanged. The element `null` records the branch taken.

Scale. Every mean and standard error is derived from `fs_dgm_scale(dgm)`. The orientation is the harm direction $s = \text{sign}(m_tau[Q])$, so the planted effect is oriented positive: with Q the true harm region, $\tau_{Qc} = s * m_tau[Qc]$, $bint = s * (m_tau[Q] - m_tau[Qc])$, $se_{1000} = \sqrt{V_eff[Q] / (1000 * P(Q))}$, and for each candidate $\beta_g = \tau_{Qc} + bint * P_{Qg}$ – the signed mixture $s * (m_tau[Qc] + (m_tau[Q] - m_tau[Qc]) * P_{Qg})$ – and $se_g = se_{1000} * \sqrt{1000 / n} * \sqrt{P(Q) / P_g}$ (sign-free, from $V_eff[Q]$). Opposite-sign families ($\text{sign}(m_tau[Qc]) \neq s$) are supported: benefit-direction candidates carry oriented-negative `beta_g`, and `tauQc` may be negative for such families. When both region effects share a sign the values coincide exactly with the former oriented-absolute reading. A DGM with `m_tau[Q]` exactly zero is rejected (no harm direction to orient by): plant a nonzero Q effect or use the null path.

Value

An object of class `c("fs_oc_family", "list")` with elements

`lab` Character, length M: the rule of each candidate.

`Pg` Population prevalence $P(g)$.

`PQg` Purity $P(g \& Q) / P(g)$.

beta_g Oriented mixture mean $s * (m_tau[Qc] + (m_tau[Q] - m_tau[Qc]) * PQg)$, with $s = \text{sign}(m_tau[Q])$ the harm direction.
se_g Anchored standard error at n .
sens_g $P(g \& Q) / P(Q)$.
spec_g $1 - P(g \& Qc) / P(Qc)$.
ovl M-by-M matrix of $P(g_i \& g_j)$.
M Number of candidates.
PQ $P(Q)$, the prevalence of the true region.
memb Logical matrix, $nrow(df_super)$ by M , the population membership of each candidate.
null Logical: TRUE when the null branch was taken (Q empty).
orientation Alternative branch only: list with the harm-direction sign s , the signed region effects m_tau_Q and m_tau_Qc , and the oriented mixture coefficients $\tau_{Qc} = s * m_tau_Qc$ (may be negative for opposite-sign families) and $bint = s * (m_tau_Q - m_tau_Qc)$. Absent on the null branch.
scale The `fs_dgm_scale` object used.
n The trial size.
args_used The `forestsearch_args` entries consumed, with the resolved `n.min`.
cuts The cut expressions `get_FScuts()` produced.
counts Named integer vector: candidates at each stage.

See Also

[fs_oc_predict](#), [fs_dgm_scale](#)

Examples

```

set.seed(1)
N <- 2000
age <- round(rnorm(N, 35, 9)); pre <- round(rexp(N, 1 / 500))
V <- factor(rbinom(N, 1, 0.42), levels = 0:1)
inQ <- as.integer(age > 34 & pre <= 745)
mu0 <- 40 + 0.2 * age
dgm <- structure(list(
  df_super = data.frame(age = age, preanti = pre, V = V,
    mu0 = mu0, mu1 = mu0 - 26 - 14 * inQ,
    flag_harm = inQ),
  outcome_type = "continuous", effect_measure = "MD",
  model_params = list(sigma = 127.5)), class = c("glm_dgm", "list"))
fam <- fs_oc_family_enumerate(
  dgm, list(confounders.name = c("age", "preanti", "V"),
    conf.cont_jcuts = list(age = 4, preanti = 4), n.min = 60),
  n = 500)
fam

```

fs_oc_grid

*Sweep the declaration thresholds on one draw set per trial size***Description**

Evaluates every `fs_oc_predict()` quantity over the full crossing of `n`, `c1` and `c2`, enumerating the family and drawing the candidates' fitted effects **once per** `n` (and per gate) and sweeping the thresholds against those draws. Thresholds enter only the gate, so the sweep costs arithmetic, not draws.

Usage

```
fs_oc_grid(
  dgm = NULL,
  forestsearch_args = list(),
  n,
  c1,
  c2,
  family = NULL,
  consistency_method = c("resample", "split"),
  pconsistency = NULL,
  draws = 2e+05,
  block = 50000,
  seed = NULL,
  verbose = FALSE,
  ...
)
```

Arguments

<code>dgm</code>	An object of class "glm_dgm"; used only when <code>family = NULL</code> .
<code>forestsearch_args</code>	Named list of forestsearch arguments. Supplies the default <code>c1</code> (<code>effect.threshold</code>) and <code>c2</code> (<code>consistency.threshold</code>) and, when <code>family = NULL</code> , the cut specification and floors for fs_oc_family_enumerate .
<code>n</code>	Numeric vector of trial sizes.
<code>c1, c2</code>	Numeric vectors of screening and consistency floors; the grid is their full crossing with <code>n</code> .
<code>family</code>	<code>NULL</code> (enumerate per <code>n</code>) or an <code>fs_oc_family</code> object, accepted only when its <code>n</code> equals the single <code>n</code> requested – a family built at one <code>n</code> has the wrong prevalence floor for another.
<code>consistency_method</code>	"resample", "split", or both.
<code>pconsistency</code>	Numeric in (0, 1). Consistency-rate threshold for the "resample" gate – the <code>p*</code> of forestsearch 's <code>pconsistency.threshold</code> : the fraction of splits that must

	clear c2, not an effect threshold. Default forestsearch_args\$pconsistency.threshold, itself defaulting to forestsearch()'s default (0.90); an explicit value overrides. Ignored by "split".
draws	Integer. Number of Monte-Carlo draws.
block	Integer or Inf. Draw-block size; Inf means one block (exact). Default 5e4.
seed	Integer or NULL. Passed to set.seed() before the draws when non-NULL.
verbose	Logical. Report per-n timings.
...	Passed to fs_oc_family_enumerate when family = NULL (max_M, verbose).

Details

For each n the family is enumerated with `fs_oc_family_enumerate` (or the supplied family is used when its n matches), the draw set is generated under seed (the same seed for every n : common random numbers across trial sizes), and every $(c1, c2)$ is evaluated on it. With `block = Inf` the whole draw set is held and the computation is exactly `fs_oc_predict`'s – one grid point is identical() to a call of `fs_oc_predict()` at the same settings and seed. With a finite block the draws are generated in row blocks of that size and the quantities accumulated, so memory is $O(\text{block} \times M)$ instead of $O(\text{draws} \times M)$; blocked results agree with the one-block results to Monte-Carlo precision (the RNG stream is laid out differently and proportions are formed as count / draws), not bit-for-bit.

Value

An object of class `c("fs_oc_grid", "list")`:

table Data frame, one row per $(n, \text{gate}, c1, c2)$: n , consistency_method, pconsistency, $c1$, $c2$, M , draws, block, seed, det_rate, det_rate_se, EnH, Esens, Espec, Eppv, Enpv, EbetaH, Enaive_bias, mass_below.

results List, parallel to the rows of table, of the full per-point objects (class `fs_oc_predict`, with `P1`, `p_sel`, `sel_c` and their MC SEs).

families Per n : M , the floor, the stage counts and the family.

timing Per n and gate: seconds for enumeration, drawing and sweeping.

settings The call's settings.

See Also

[fs_oc_predict](#), [fs_oc_invert](#)

Examples

```
piQ <- 0.34
fam <- structure(list(
  lab = c("Q", "P", "D"), Pg = c(piQ, 0.45, 0.31),
  PQg = c(1, 0.28 / 0.45, 1), sens_g = c(1, 0.28 / piQ, 0.31 / piQ),
  spec_g = c(1, 1 - 0.17 / (1 - piQ), 1),
  ovl = matrix(c(piQ, 0.28, 0.31, 0.28, 0.45, 0.28 * 0.31 / piQ,
    0.31, 0.28 * 0.31 / piQ, 0.31), 3, 3),
  M = 3L, PQ = piQ, n = 500), class = c("fs_oc_family", "list"))
```

```
fam$beta_g <- 26 + 14 * fam$PQg
fam$se_g   <- 13.7 * sqrt(2) * sqrt(piQ / fam$Pg)
g <- fs_oc_grid(family = fam, n = 500, c1 = c(20, 30, 40), c2 = 10,
               consistency_method = "resample", draws = 2e4, seed = 1)
g
```

fs_oc_invert

Invert the family declaration rate for a target

Description

Finds the threshold – c_1 at fixed c_2 , or c_2 at fixed c_1 – at which the family declaration rate equals target, on one fixed draw set: "the c_1 giving 80\ read as a type-I error under a null DGM.

Usage

```
fs_oc_invert(
  dgm = NULL,
  forestsearch_args = list(),
  n,
  target,
  solve_for = c("c1", "c2"),
  c1 = NULL,
  c2 = NULL,
  family = NULL,
  consistency_method = c("resample", "split"),
  pconsistency = NULL,
  draws = 2e+05,
  seed = NULL,
  tol = 0.001,
  ...
)
```

Arguments

dgm	An object of class "glm_dgm"; used only when family = NULL.
forestsearch_args	Named list of forestsearch arguments. Supplies the default <code>c1</code> (<code>effect.threshold</code>) and <code>c2</code> (<code>consistency.threshold</code>) and, when family = NULL, the cut specification and floors for fs_oc_family_enumerate .
n	Single trial size.
target	Numeric in (0, 1); may be a vector when solve_for = "c1". Target family declaration rate(s).
solve_for	"c1" (at fixed c_2) or "c2" (at fixed c_1).
c1, c2	The fixed threshold (the one not solved for; the solved-for argument is ignored). Defaults from forestsearch_args.

family	NULL (enumerate per n) or an fs_oc_family object, accepted only when its n equals the single n requested – a family built at one n has the wrong prevalence floor for another.
consistency_method	One gate.
pconsistency	Numeric in (0, 1). Consistency-rate threshold for the "resample" gate – the p* of forestsearch 's pconsistency.threshold: the fraction of splits that must clear c2, not an effect threshold. Default forestsearch_args\$pconsistency.threshold, itself defaulting to forestsearch()'s default (0.90); an explicit value overrides. Ignored by "split".
draws	Integer. Number of Monte-Carlo draws.
seed	Integer or NULL. Passed to set.seed() before the draws when non-NULL.
tol	Bisection tolerance on the threshold scale (solve_for = "c2" only).
...	Passed to fs_oc_family_enumerate when family = NULL.

Details

Solving for c1: an order statistic. At fixed (n, gate, c2) neither the eligible set nor the winner depends on c1: with eligible_g = (Bhat_g - c2 >= z_p * se_g) ("resample") or (W1_g >= c2) & (W2_g >= c2) ("split"), each draw's winner is $w = \text{argmax}_{\{\text{eligible}\}} \text{Bhat}_g$ and its statistic $T = \text{Bhat}_w$ (-Inf when nothing is eligible). Declaration at c1 is exactly $T \geq c1$, and w is the maxef-fCons winner whenever anything declares. So the reduction is computed once (`.fs_oc_reduce()`), the declaration rate at any c1 is $\text{mean}(T \geq c1)$, and the largest c1 attaining a target is the k-th largest T with $k = \text{ceiling}(\text{target} * \text{draws})$: an order statistic, no search. The **ceiling** is $\text{mean}(T > -\text{Inf})$, the fraction of draws with any eligible member, set by the consistency screen alone; a target above it is unattainable and returns NA with the ceiling and the binding threshold named – no extrapolation. target may be a vector: one reduction serves every target.

Solving for c2 at fixed c1: the eligible set moves with c2, so the rate is bracketed from the draws' range and bisected to tol on fixed draws (a monotone non-increasing step function); the returned value is the largest threshold whose rate is still at least target.

Every result reports the achieved rate and its MC SE, so the resolution the draw count supports is visible.

Value

For a single target, an object of class `c("fs_oc_invert", "list")`: value (the threshold, or NA), achieved (rate at value), achieved_se, target, ceiling, ceiling_se, binding (which threshold sets the ceiling), attainable, solve_for, fixed, k (the order-statistic rank, solve_for = "c1"), next_step_rate, iterations (bisections, solve_for = "c2"), settings. For a vector target, a list of such objects (class `fs_oc_invert_list`) with a table attribute.

See Also

[fs_oc_grid](#)

Examples

```
piQ <- 0.34
fam <- structure(list(
  lab = c("Q", "P", "D"), Pg = c(piQ, 0.45, 0.31),
  PQg = c(1, 0.28 / 0.45, 1), sens_g = c(1, 0.28 / piQ, 0.31 / piQ),
  spec_g = c(1, 1 - 0.17 / (1 - piQ), 1),
  ovl = matrix(c(piQ, 0.28, 0.31, 0.28, 0.45, 0.28 * 0.31 / piQ,
    0.31, 0.28 * 0.31 / piQ, 0.31), 3, 3),
  M = 3L, PQ = piQ, n = 500), class = c("fs_oc_family", "list"))
fam$beta_g <- 26 + 14 * fam$PQg
fam$se_g <- 13.7 * sqrt(2) * sqrt(piQ / fam$Pg)
fs_oc_invert(family = fam, n = 500, target = 0.5, solve_for = "c1", c2 = 10,
  consistency_method = "resample", draws = 2e4, seed = 1)
```

fs_oc_predict	<i>Predicted operating characteristics of the search over a candidate family</i>
---------------	--

Description

Monte-Carlo prediction of the maxeffCons search's operating characteristics – family declaration rate, per-candidate declaration, the selection distribution, expected selected-subgroup size, sensitivity, specificity, PPV, NPV, the oriented effect on the selected rule and the naive selection bias – from the joint normal law of the candidates' fitted effects. The family is a population enumeration from [fs_oc_family_enumerate](#) or a supplied fs_oc_family object.

Usage

```
fs_oc_predict(
  dgm = NULL,
  forestsearch_args = list(),
  n,
  c1 = NULL,
  c2 = NULL,
  family = NULL,
  consistency_method = c("resample", "split"),
  pconsistency = NULL,
  draws = 2e+05,
  seed = NULL,
  ...
)
```

Arguments

dgm	An object of class "glm_dgm"; used only when family = NULL.
-----	---

forestsearch_args

	Named list of forestsearch arguments. Supplies the default <code>c1</code> (<code>effect.threshold</code>) and <code>c2</code> (<code>consistency.threshold</code>) and, when <code>family = NULL</code> , the cut specification and floors for fs_oc_family_enumerate .
<code>n</code>	Integer. Trial size. Overrides any size implied by the arguments; sets the size floor and the standard-error scale of an enumerated family and converts expected prevalence to expected subjects.
<code>c1</code>	Numeric. Screening floor on the full-sample effect – the <code>c1</code> of forestsearch 's <code>effect.threshold</code> (legacy <code>hr.threshold</code>). Default <code>forestsearch_args\$effect.threshold</code> ; an explicit value overrides.
<code>c2</code>	Numeric. Consistency floor on each half-sample effect – the <code>c2</code> of forestsearch 's <code>consistency.threshold</code> (legacy <code>hr.consistency</code>). An effect threshold, not a proportion; the proportion is <code>pconsistency</code> below. Default <code>forestsearch_args\$consistency.threshold</code> ; an explicit value overrides.
<code>family</code>	<code>NULL</code> (enumerate with fs_oc_family_enumerate) or an <code>fs_oc_family</code> object used as-is.
<code>consistency_method</code>	"resample" (the package's production closed-form screen, the default) or "split" (the analytic document's single-split gate). See Details.
<code>pconsistency</code>	Numeric in (0, 1). Consistency-rate threshold for the "resample" gate – the <code>p</code> of forestsearch 's <code>pconsistency.threshold</code> : the fraction of splits that must clear <code>c2</code> , not an effect threshold. Default <code>forestsearch_args\$pconsistency.threshold</code> , itself defaulting to <code>forestsearch()</code> 's default (0.90); an explicit value overrides. Ignored by "split".
<code>draws</code>	Integer. Number of Monte-Carlo draws.
<code>seed</code>	Integer or <code>NULL</code> . Passed to <code>set.seed()</code> before the draws when non- <code>NULL</code> .
<code>...</code>	Passed to fs_oc_family_enumerate when <code>family = NULL</code> (<code>max_M</code> , <code>verbose</code>).

Details

The full-sample fitted effects of the M candidates are taken as $N(\beta_g, S_g)$ with $S_g = \rho \circ (se_g se'_g)$, $\rho_{ij} = P(g_i \cap g_j) / \sqrt{P(g_i)P(g_j)}$. Two independent half-sample draws $W1, W2$, each with covariance $2 * S_g$, are generated through [fs_sym_root](#); the full-sample effect is $Bhat = (W1 + W2) / 2$.

The two gates. Family declaration is any candidate declaring; the selected rule is the effect maximiser among declaring candidates (`maxeffCons`).

"resample" – the package's production screen The consistency stage of [forestsearch](#) (default `consistency_method = "resample"`) represents the random 50/50 split's half effects as $\hat{\beta} \pm D$ and computes the consistency rate in closed form as $2\Phi((\hat{\beta} - c_2)/\sigma_D) - 1$, a candidate passing when that rate is at least `pconsistency` (`R/consistency_resample.R`; `R/subgroup_consistency_helpers.R` drops a candidate when `p.consistency < pconsistency.threshold`). Inverting: $\hat{\beta} \geq c_2 + z_p \sigma_D$ with $z_p = \Phi^{-1}((1 + p)/2)$. σ_D is `sqrt(sum(dfbeta[i, treat]^2))`, the sandwich standard error of the subgroup treatment coefficient; in the draw model above the analogous quantity is `se_g`, and on a simulated MD40 trial the two agree within a few percent with no prevalence trend, so the wrapper identifies $\sigma_D = se_g$. The gate is then a single threshold on the full-sample draw, $(Bhat \geq c1) \& (Bhat - c2 \geq z_p * se_g)$, and only $Bhat \sim N(\beta_a_g, S_g)$ is drawn – one matrix, not two.

"split" – **the analytic document's historical gate** One half-sample pair $W1, W2$ stands in for the consistency rate: $(W1 + W2 \geq 2 * c1) \& (W1 \geq c2) \& (W2 \geq c2)$, the screening floor $c1$ on the full-sample effect and the consistency floor $c2$ on each half. Kept bit-identical to the document's worked-predictions chunk.

All expectations are selection-weighted population functionals of the family, conditional on declaration. Monte-Carlo standard errors are given for the proportions $\sqrt{p(1-p)/\text{draws}}$.

Value

An object of class `c("fs_oc_predict", "list")`:

`det_rate, det_rate_se` Family declaration rate and its MC standard error.

`P1, P1_se` Per-candidate declaration probability.

`p_sel, p_sel_se` Per-candidate selection probability (unconditional).

`sel_c` Selection distribution given declaration.

`EnH` Expected selected-subgroup size in subjects, $n * \text{sum}(\text{sel_c} * P_g)$.

`Esens, Espec, Eppv, Enpv` Expected classification metrics of the selected rule against Q .

`EbetaH` Expected oriented true effect on the selected rule.

`Enaive_bias` Expected naive minus true effect on the selected rule.

`mass_below` Selection mass on rules whose true mean is below $c1$.

`M, lab` Family size and labels.

`settings` $n, c1, c2, \text{consistency_method}, \text{pconsistency}$ (NA for "split"), `draws`.

`seed` The seed used.

`family` The family object.

See Also

[fs_oc_family_enumerate](#), [fs_sym_root](#)

Examples

```
# A hand-built three-candidate family
piQ <- 0.34
fam <- structure(list(
  lab = c("Q", "P", "D"), Pg = c(piQ, 0.45, 0.31),
  PQg = c(1, 0.28 / 0.45, 1), sens_g = c(1, 0.28 / piQ, 0.31 / piQ),
  spec_g = c(1, 1 - 0.17 / (1 - piQ), 1),
  ovl = matrix(c(piQ, 0.28, 0.31, 0.28, 0.45, 0.28 * 0.31 / piQ,
    0.31, 0.28 * 0.31 / piQ, 0.31), 3, 3),
  M = 3L, PQ = piQ), class = c("fs_oc_family", "list"))
fam$beta_g <- 26 + 14 * fam$PQg
fam$se_g <- 13.7 * sqrt(2) * sqrt(piQ / fam$Pg)
fs_oc_predict(family = fam, n = 500, c1 = 30, c2 = 10,
  consistency_method = "resample", draws = 2e4, seed = 1)
fs_oc_predict(family = fam, n = 500, c1 = 30, c2 = 10,
  consistency_method = "split", draws = 2e4, seed = 1)
```

fs_plot_bias_coverage *Bias-vs-coverage display (three panels)*

Description

Draws the standard bias-vs-coverage display from a stacked `fs_sim_bias_coverage()` table carrying a cell column: (1) one-sided coverage against b with Gaussian-reference curves $\Phi(z_1 * r - b)$ for the given curves values of r ; (2) two-sided coverage against b with curves $\Phi(z_2 * r - b) - \Phi(-z_2 * r - b)$; (3) observed coverage against the Gaussian reference at each point's own (b, r) , with the diagonal. Estimators are distinguished by marker shape, the nominal level by a dashed line, and cells by text labels.

Usage

```
fs_plot_bias_coverage(
  tbl,
  labels = TRUE,
  curves = c(1, 1.25, 1.5, 2),
  level = 0.95,
  side = c("lower", "upper")
)
```

Arguments

<code>tbl</code>	Stacked output of <code>fs_sim_bias_coverage()</code> with an added cell column (one factor level per simulation cell).
<code>labels</code>	Draw cell labels next to the points (default TRUE).
<code>curves</code>	Gaussian-reference r values for panels 1-2 (default <code>c(1, 1.25, 1.5, 2)</code>).
<code>level</code>	Nominal level for the reference lines (default 0.95).
<code>side</code>	"lower" (default) or "upper": which one-sided bound the table's <code>cov1</code> was computed for (pass the same value given to <code>fs_sim_bias_coverage()</code>). Only panel 1's reference curves and title change: $\Phi(z_1 * r + b)$ under "upper".

Value

A patchwork object of the three ggplot panels.

fs_rule_covariate_pairs

Covariate signatures of the realized rules

Description

Collapses realized rules to the **set of covariates** they name, discarding the cut values. A rules-by-frequency table lists every distinct sg_def string and so is dominated by cut noise – a 500-replicate survival run produced 583 rows, in which {age <= 46} & {nodes <= 5} and {nodes <= 3} & {age <= 46} are separate entries despite being the same pairing. This reports how often each covariate *combination* is selected, which is the structural question.

Usage

```
fs_rule_covariate_pairs(x, true_covariates = NULL, detected_only = TRUE)
```

Arguments

x	A results data frame carrying sg_def, a bundle with results, a path to an .rds, or a bare character vector of rules.
true_covariates	Character vector. The true-region covariates, used to classify each signature as "exact" (the signature is exactly this set), "partial" (overlaps it) or "none". NULL skips the classification.
detected_only	Logical. Restrict to rows with detected == 1 when a results frame is supplied.

Value

A data frame ordered by frequency: covariates, k, n, share, n_rules, and match when true_covariates is given.

What a signature is

The covariate names in the rule, de-duplicated and sorted, joined by " & ". Sorting is what makes the two orderings above collapse together. Two consequences worth knowing:

- A rule naming one covariate twice — a band such as !{nodes <= 3} & {nodes <= 5} — has signature "nodes", of size 1. The k column gives the number of distinct covariates, and n_rules the number of distinct rule strings that collapsed into the row, so bands are visible rather than hidden.
- Logically equivalent spellings of the same condition (!{meno} versus {meno == 0}) collapse together, which a raw rule table also splits.

fs_scale_se	<i>Estimator Standard Deviation at a Given Trial Size</i>
-------------	---

Description

Converts a [fs_dgm_scale](#) object into the sampling standard deviation of the within-region difference-in-means estimator at trial size n .

Usage

```
fs_scale_se(scale, n, region = NULL, jensen = FALSE)
```

Arguments

scale	An object of class "fs_dgm_scale".
n	Integer. Trial size.
region	Character. Region name, matching a row of scale\$regions. Default NULL returns every region.
jensen	Logical. If TRUE, inflate by the exact factor accounting for the random region size $n_g \sim \text{Bin}(n, P(g))$ and random arm split $n_1 \mid n_g \sim \text{Bin}(n_g, p)$, returning the unconditional standard deviation. Default FALSE, which returns the idealised (fixed-count) value.

Value

Named numeric vector of standard deviations.

See Also

[fs_dgm_scale](#)

fs_sim_bias_coverage	<i>Bias and coverage summary for one simulation cell</i>
----------------------	--

Description

From a combined bundle's per-replicate results (the recorder columns of the fb_mr_field template family: nv_H_*, mr_H_*, fld_H_*, betaHhat_H, detected, and their Hc twins), computes, per estimator over the detected replicates: the retained bias on the log scale, the empirical SD of the log estimate, the mean reported SE, their ratios $b = \text{bias_log} / \text{sd_emp}$ and $r = \text{se_mean} / \text{sd_emp}$, the observed one- and two-sided coverage of the target with Wilson intervals, and the Gaussian reference coverages at (b, r).

Usage

```
fs_sim_bias_coverage(
  results,
  block = c("H", "Hc"),
  estimators = c("naive", "mr", "fld"),
  level = 0.95,
  target = "betaHhat",
  side = c("lower", "upper"),
  scale = c("log", "identity")
)
```

Arguments

results	Data frame of per-replicate records from a combined bundle (<code>readRDS(...)\$results</code>).
block	"H" (harm, default) or "Hc" (complement). The "fld" estimator is dropped with a message when the block's field columns are absent (fld_Hc_* exist only for bundles run with <code>field_complement = TRUE</code>).
estimators	Subset of <code>c("naive", "mr", "fld", "mr_w", "mr_wf")</code> (default the first three). "mr_w" / "mr_wf" are MR (IJ) with the winner-only / winner-floor SE (<code>mr_<block>_se_w</code> / <code>_se_wf</code> and their <code>_lo_w/_hi_w</code> / <code>_lo_wf/_hi_wf</code> bounds), available only on bundles that recorded them.
level	Coverage level (default 0.95).
target	"betaHhat" (default; the per-replicate conditional target <code>betaHhat_<block></code>), "oracle" (the per-replicate oracle estimate <code>or_<block>_est</code>), or a numeric scalar on the HR scale (e.g. a CDE or marginal truth value from the bundle's truth element).
side	"lower" (default; one-sided coverage of the lower bound, the harm-claim orientation) or "upper" (one-sided coverage of the upper bound, the benefit-claim orientation). Two-sided quantities are unaffected.
scale	"log" (default) or "identity". The working scale of the estimate columns: "log" for ratio measures (HR/OR/RR; bias, SD and the normal-based one-sided bound $\exp(\log \text{ est } \mp z * SE)$ are computed on the log scale, and non-positive estimates are dropped), "identity" for difference measures (MD/RD; bias, SD and the bound $\text{est } \mp z * SE$ on the natural scale, no positivity guard). Add-only: the default reproduces the previous output exactly. Under "identity" the <code>bias_log</code> column holds the identity-scale bias (the column name is kept so stacked tables and <code>fs_plot_bias_coverage()</code> read either scale unchanged).

Details

Conventions: the one-sided lower bound is the stored `fld_<block>_lo1s` for the field row and $\exp(\log \text{ est } - qnorm(level) * SE)$ for naive and MR (IJ); two-sided coverage uses each estimator's stored bounds; the field estimator is the shrunk-field `est2` with the two-sided Lambda-quantile interval. Under `side = "upper"` (the complement's benefit-claim orientation) the one-sided coverage is `truth <= upper bound`, with the field's stored `fld_<block>_up1s` and $\exp(\log \text{ est } + qnorm(level) * SE)$

for naive and MR (IJ), and the Gaussian reference is $\Phi(z_{\text{level}} * r + b)$ – a positive retained bias helps an upper bound where it hurts a lower one.

Value

Data frame, one row per estimator: estimator, n, bias_log, sd_emp, se_mean, b, r, cov1, cov1_wilson_lo, cov1_wilson_hi, cov2, cov2_wilson_lo, cov2_wilson_hi, cov1_ref, cov2_ref.

fs_sym_root	<i>Symmetric Square Root of a Covariance Matrix</i>
-------------	---

Description

Computes the symmetric matrix square root of $\text{scale} * S$ after symmetrising S as $(S + t(S)) / 2$: with eigendecomposition $S = V D V'$, the returned matrix is $R = V D^{1/2} V'$ (eigenvalues scaled by scale and clamped at zero before the square root), so that $R R' = \text{scale} \cdot S$.

Usage

```
fs_sym_root(S, scale = 2)
```

Arguments

- S Numeric square matrix, a covariance (or any matrix whose symmetrisation $(S + t(S)) / 2$ is intended; the function symmetrises before decomposing). May be rank-deficient; eigenvalues that are slightly negative from numerical noise are clamped to zero.
- scale Numeric scalar multiplying S before the root is taken, so that the returned root R satisfies $R R' = \text{scale} \cdot S$. Default 2, the half-sample covariance convention of the analytic documents.

Value

A symmetric numeric matrix R of the same dimension as S , satisfying $R \%*\% t(R)$ equal to scale times the symmetrised S (up to the zero-clamp on negative eigenvalues).

Why the symmetric root and not $V D^{(1/2)}$

Both roots satisfy $R R' = \text{scale} \cdot S$, but the asymmetric root $V D^{1/2}$ depends on the eigenvector basis, which is not a continuous function of its input: signs flip and vectors rotate freely inside degenerate subspaces – which a candidate covariance built from complement pairs has exactly, since the complement constraints make it rank-deficient by construction. A pinned RNG seed fixes the standard-normal draws Z ; it cannot fix the basis Z is multiplied by, so the asymmetric root is non-reproducible across arithmetically equivalent routes to the same matrix. The symmetric root $V D^{1/2} V'$ is a matrix function of the symmetrised input alone – basis-free and continuous – so samples built from it are continuous, reproducible functions of the inputs.

Examples

```
S <- crossprod(matrix(rnorm(25), 5)) # symmetric positive definite
R <- fs_sym_root(S, scale = 2)
max(abs(R %*% t(R) - 2 * S))          # ~ 1e-14
isSymmetric(R)

# Rank-deficient input: the root exists and stays symmetric
X <- matrix(rnorm(15), 5, 3)
R2 <- fs_sym_root(tcrossprod(X), scale = 1)
max(abs(R2 %*% t(R2) - tcrossprod(X)))
```

fs_to_guohe

Run Guo and He Algorithm 3 on a forestsearch maxeff fit

Description

Bridge from a `forestsearch(sg_focus = "maxeff")` fit to the Guo and He (2021) Algorithm 3 de-biasing estimator `guohe_algorithm3()`. This is the "post-hoc identified subgroups" application of Guo and He: `forestsearch` enumerates a large candidate family and selects the argmax; Guo and He supplies a de-biased effect and one-sided bound for the selected subgroup, correcting the winner's-curse optimism of that selection. The candidate family is materialized from the fit with `fs_export_maxeff_family()` and, by default, verified against `forestsearch`'s own selected-subgroup membership with `fs_assert_membership()` before any de-biasing runs.

Usage

```
fs_to_guohe(
  fit,
  time = "time_months",
  event = "status",
  treatment = "hormon",
  B = 2000L,
  r = 0.03,
  level = 0.05,
  seed = NULL,
  min_events = 5L,
  parallel = FALSE,
  verify = TRUE
)
```

Arguments

<code>fit</code>	A <code>forestsearch</code> object fitted with <code>sg_focus = "maxeff"</code> .
<code>time, event, treatment</code>	Names of the time, event, and 0/1 treatment columns on <code>fit\$df.est</code> . The defaults match the GBSG analyses ("time_months", "status", "hormon"); set them to the names used in the <code>forestsearch</code> call.

B	Number of bootstrap resamples passed to <code>guohe_algorithm3()</code> .
r	Shrinkage tuning parameter, strictly between 0 and 0.5; see <code>guohe_algorithm3()</code> and <code>guohe_adaptive_r()</code> .
level	One-sided level; the two-sided interval has coverage $1 - \text{level}$.
seed	Optional integer for reproducible resampling.
min_events	Minimum events for a candidate to be estimable.
parallel	Logical; passed to <code>guohe_algorithm3()</code> . Requires a <code>future::plan()</code> to have been set by the caller.
verify	Logical; if TRUE (default), run <code>fs_assert_membership()</code> and stop on any mismatch between the reconstructed and the original selected-subgroup membership.

Details

The de-biasing engine is an independent implementation of the published method, validated by reproducing the simulation study of Section 5 of Guo and He (2021).

The Guo and He selection is `orient = +1` (the maximum-hazard-ratio, i.e. harm, subgroup), so its `argmax` coincides with the forestsearch `maxeff` selection. If the two nevertheless disagree (possible when candidates are dropped as non-estimable), a warning is issued and the returned Guo and He objects refer to the Guo and He `argmax`.

Value

A list with elements `gh` (the full "guohe_a3" object), `export` (the `fs_export_maxeff_family()` result), `selected` (the Guo and He `argmax` candidate name), and the convenience scalars `naive_hr`, `debiased_hr`, and `bound_hr` (hazard-ratio scale).

References

Guo, X. and He, X. (2021). Inference on Selected Subgroups in Clinical Trials. *Journal of the American Statistical Association*, **116**(535), 1498–1506. doi:10.1080/01621459.2020.1740096

See Also

`fs_export_maxeff_family()`, `fs_assert_membership()`, `guohe_algorithm3()`, `guohe_adaptive_r()`

Examples

```
set.seed(11)
n <- 200
df <- data.frame(id = seq_len(n), treat = rbinom(n, 1, 0.5),
                 age = round(rnorm(n, 55, 12)),
                 bm = factor(rbinom(n, 1, 0.5)))
harm <- df$age <= 50 & df$bm == "1"
tt <- rexp(n, 0.05 * exp(log(2.5) * df$treat * harm))
df$time <- pmin(tt, 60)
df$event <- as.integer(tt <= 60)
fit <- forestsearch(
```



```

df.analysis = df,
outcome.name = "time", event.name = "event",
treat.name = "treat", id.name = "id",
confounders.name = c("age", "bm"),
sg_focus = "maxeff", use_grf = FALSE, use_lasso = FALSE,
fs.splits = 50, n.min = 30, d0.min = 5, d1.min = 5,
hr.threshold = 1.1, hr.consistency = 1.0,
pconsistency.threshold = 0.5, maxk = 2,
details = FALSE, plot.sg = FALSE,
parallel_args = list(plan = "sequential", workers = 1L,
                      show_message = FALSE)
)
gh <- fs_to_guohe(fit, time = "time", event = "event",
                 treatment = "treat", B = 50, seed = 3)
gh$naive_hr
gh$debiased_hr
gh$bound_hr

```

generate_aft_dgm_flex *Generate Synthetic Survival Data using AFT Model with Flexible Subgroups*

Description

Creates a data generating mechanism (DGM) for survival data using an Accelerated Failure Time (AFT) model with Weibull distribution. Supports flexible subgroup definitions and treatment-subgroup interactions.

Usage

```

generate_aft_dgm_flex(
  data,
  continuous_vars,
  factor_vars,
  continuous_vars_cens = NULL,
  factor_vars_cens = NULL,
  set_beta_spec = list(set_var = NULL, beta_var = NULL),
  outcome_var,
  event_var,
  treatment_var = NULL,
  subgroup_vars = NULL,
  subgroup_cuts = NULL,
  draw_treatment = FALSE,
  model = "alt",
  k_treat = 1,
  k_inter = 1,
  n_super = 5000,

```

```

select_censoring = TRUE,
cens_type = "weibull",
cens_params = list(),
cens_intercept_only = FALSE,
seed = 8316951,
verbose = TRUE,
standardize = FALSE,
spline_spec = NULL
)

```

Arguments

<code>data</code>	A data.frame containing the input dataset to base the simulation on
<code>continuous_vars</code>	Character vector of continuous variable names to be standardized and included as covariates
<code>factor_vars</code>	Character vector of factor/categorical variable names to be converted to dummy variables (largest value as reference)
<code>continuous_vars_cens</code>	Character vector of continuous variable names to be used for censoring model. If NULL, uses same as <code>continuous_vars</code> . Default NULL
<code>factor_vars_cens</code>	Character vector of factor variable names to be used for censoring model. If NULL, uses same as <code>factor_vars</code> . Default NULL
<code>set_beta_spec</code>	List with elements 'set_var' and 'beta_var' for manually setting specific beta coefficients. Default list(set_var = NULL, beta_var = NULL)
<code>outcome_var</code>	Character string specifying the name of the outcome/time variable
<code>event_var</code>	Character string specifying the name of the event/status variable (1 = event, 0 = censored)
<code>treatment_var</code>	Character string specifying the name of the treatment variable. If NULL, treatment will be randomly simulated with 50/50 allocation
<code>subgroup_vars</code>	Character vector of variable names defining the subgroup. Default is NULL (no subgroups)
<code>subgroup_cuts</code>	Named list of cutpoint specifications for subgroup variables. See Details section for flexible specification options
<code>draw_treatment</code>	Logical indicating whether to redraw treatment assignment in simulation. Default is FALSE (use original assignments)
<code>model</code>	Character string: "alt" for alternative model with subgroup effects, "null" for null model without subgroup effects. Default is "alt"
<code>k_treat</code>	Numeric treatment effect modifier. Values >1 increase treatment effect, <1 decrease it. Default is 1 (no modification)
<code>k_inter</code>	Numeric interaction effect modifier for treatment-subgroup interaction. Default is 1 (no modification)
<code>n_super</code>	Integer specifying size of super population to generate. Default is 5000

select_censoring	Logical. If TRUE (default), fits the censoring distribution to the observed censoring times in data using survreg with AIC-based selection among Weibull and log-normal models (with and without covariates). If FALSE, no model is fitted; the censoring distribution is specified entirely by cens_params. Default TRUE.
cens_type	Character string specifying censoring distribution type: "weibull" or "uniform". Controls which parametric family is considered when select_censoring = TRUE, and determines the required structure of cens_params when select_censoring = FALSE. Default "weibull".
cens_params	Named list of censoring distribution parameters. Interpretation depends on select_censoring and cens_type: select_censoring = TRUE Ignored; all parameters are estimated from data. select_censoring = FALSE, cens_type = "uniform" Must supply min and max. If either is absent, defaults to $0.5 * \min(y)$ and $1.5 * \max(y)$ with a message. select_censoring = FALSE, cens_type = "weibull" Must supply mu (log-scale location) and tau (scale). Optionally supply type ("weibull" or "lognormal"); defaults to "weibull". Censoring is treated as intercept-only (no covariate or treatment dependence): $\text{lin_pred_cens}_0 = \text{lin_pred_cens}_1 = \mu$. Default list().
cens_intercept_only	Logical. Honored only in force-fit mode (select_censoring = FALSE with empty cens_params). If TRUE, fits $\text{Surv}(y, 1 - \text{event}) \sim 1$ with no treat or covariate dependence. If FALSE (default), the censoring formula is determined by the censoring covariate vectors: - Both *_cens = NULL (default): inherit from outcome model; formula is $\sim \text{treat} + \text{zcens}_{<\text{outcome covariates}>}$. - Both *_cens = character(0): treat-only; formula is $\sim \text{treat}$. - One or both *_cens non-empty: formula is $\sim \text{treat} + \text{zcens}_{<\text{supplied covariates}>}$. Setting cens_intercept_only = TRUE with select_censoring = TRUE or with analytical cens_params (mu, tau supplied) raises an error.
seed	Integer random seed for reproducibility. Default is 8316951
verbose	Logical indicating whether to print diagnostic information during execution. Default is TRUE
standardize	Logical indicating whether to standardize continuous variables. Default is FALSE
spline_spec	List specifying spline configuration for treatment effect. Must include 'var' (variable name), 'knot', 'zeta', and 'log_hrs' (vector of length 3). Default NULL (no spline)

Details

Subgroup Cutpoint Specifications:

The subgroup_cuts parameter accepts multiple flexible specifications:

Fixed Value:

```

subgroup_cuts = list(er = 20) # er <= 20
Quantile-based:
subgroup_cuts = list(
  er = list(type = "quantile", value = 0.25) # er <= 25th percentile
)
Function-based:
subgroup_cuts = list(
  er = list(type = "function", fun = median) # er <= median
)
Range:
subgroup_cuts = list(
  age = list(type = "range", min = 40, max = 60) # 40 <= age <= 60
)
Greater than:
subgroup_cuts = list(
  nodes = list(type = "greater", quantile = 0.75) # nodes > 75th percentile
)
Multiple values (for categorical):
subgroup_cuts = list(
  grade = list(type = "multiple", values = c(2, 3)) # grade in (2, 3)
)
Custom function:
subgroup_cuts = list(
  er = list(
    type = "custom",
    fun = function(x) x <= quantile(x, 0.3) | x >= quantile(x, 0.9)
  )
)

```

Model Structure:

The AFT model with Weibull distribution is specified as:

$$\log(T) = \mu + \gamma'X + \sigma\epsilon$$

Where:

- T is the survival time
- μ is the intercept
- γ contains the covariate effects
- X includes treatment, covariates, and treatment x subgroup interaction
- σ is the scale parameter
- ϵ follows an extreme value distribution

Interaction Term:

The model creates a SINGLE interaction term representing the treatment effect modification when ALL subgroup conditions are simultaneously satisfied. This is not multiple separate interactions but one combined indicator.

Value

An object of class "aft_dgm_flex" (a list) with components:

`df_super` Data frame containing the super-population, including covariates, treatment, counterfactual linear predictors, and subgroup indicator `flag_harm`.

`df_source` Data frame with every source patient exactly once (the prepared trial data), carrying the same counterfactual outcome and censoring linear predictors as `df_super`. Consumed by `simulate_from_dgm(baseline = "fixed")` to hold baseline covariates – and hence subgroup composition – fixed across simulated trials.

`model_params` List with AFT parameters `mu`, `tau`, `gamma`, `b0`, the fitted censoring model, and optional `spline_info`.

`subgroup_info` List describing the true subgroup: `vars`, `cuts`, `definitions`, `size`, `proportion`.

`hazard_ratios` List of true HR/AHR/CDE values on the super-population (see [compute_dgm_cde](#)).

`analysis_vars` Named list of column roles (continuous, factor, covariates, treatment, outcome, event).

`model_type` "null" or "alt".

`n_super` Super-population size.

`seed` Seed used for super-population generation.

Author(s)

Your Name

References

Leon, L.F., et al. (2024). Statistics in Medicine.

Kalbfleisch, J.D. and Prentice, R.L. (2002). The Statistical Analysis of Failure Time Data (2nd ed.). Wiley.

Examples

```
## Not run:
df <- survival::gbsg
dgm <- generate_aft_dgm_flex(
  data          = df,
  outcome_var   = "rfstime",
  event_var     = "status",
  treatment_var = "hormon",
  continuous_vars = c("age", "size", "nodes", "pgr", "er"),
  factor_vars   = "meno",
  model         = "null",
  verbose       = FALSE
)
str(dgm)

## End(Not run)
```

`generate_bootstrap_synthetic`*Generate Synthetic Data using Bootstrap with Perturbation*

Description

Generate Synthetic Data using Bootstrap with Perturbation

Usage

```
generate_bootstrap_synthetic(  
  data,  
  continuous_vars,  
  cat_vars,  
  n = NULL,  
  seed = 123,  
  noise_level = 0.1,  
  id_var = NULL,  
  cat_flip_prob = NULL,  
  preserve_bounds = TRUE,  
  ordinal_vars = NULL  
)
```

Arguments

<code>data</code>	Original dataset to bootstrap from
<code>continuous_vars</code>	Character vector of continuous variable names
<code>cat_vars</code>	Character vector of categorical variable names
<code>n</code>	Number of synthetic observations to generate (default: same as original)
<code>seed</code>	Random seed for reproducibility
<code>noise_level</code>	Noise level for perturbation (0 to 1, default 0.1)
<code>id_var</code>	Optional name of ID variable to regenerate (will be numbered 1:n)
<code>cat_flip_prob</code>	Probability of flipping categorical values (default: noise_level/2)
<code>preserve_bounds</code>	Logical: should continuous variables stay within original bounds? (default: TRUE)
<code>ordinal_vars</code>	Optional character vector of ordinal categorical variables (these will be perturbed to adjacent values rather than randomly flipped)

Value

A data frame with synthetic data

Examples

```
# Example 1: Using with GBSG dataset
synth_gbsg <- generate_bootstrap_synthetic(
  data = survival::gbsg,
  continuous_vars = c("age", "size", "nodes", "pgr", "er", "rfstime"),
  cat_vars = c("meno", "hormon", "status"),
  ordinal_vars = c("grade"),
  id_var = "pid",
  n = 1000,
  seed = 123,
  noise_level = 0.15
)

# Example 2: Using with any dataset
my_data <- data.frame(
  id = 1:100,
  height = rnorm(100, 170, 10),
  weight = rnorm(100, 70, 15),
  age = sample(20:80, 100, replace = TRUE),
  gender = sample(c("M", "F"), 100, replace = TRUE),
  education = sample(1:5, 100, replace = TRUE),
  smoker = sample(0:1, 100, replace = TRUE)
)

synth_data <- generate_bootstrap_synthetic(
  data = my_data,
  continuous_vars = c("height", "weight", "age"),
  cat_vars = c("gender", "smoker"),
  ordinal_vars = c("education"),
  id_var = "id",
  n = 150,
  seed = 456
)
```

`generate_bootstrap_with_noise`*Generate Bootstrap Sample with Added Noise*

Description

Creates a bootstrap sample from a dataset with controlled noise added to both continuous and categorical variables. This function is useful for generating synthetic datasets that maintain the general structure of the original data while introducing controlled variation.

Usage

```
generate_bootstrap_with_noise(
  data,
```

```

n = NULL,
continuous_vars = NULL,
cat_vars = NULL,
id_var = "pid",
seed = 123,
noise_level = 0.1
)

```

Arguments

<code>data</code>	A data frame containing the original dataset to bootstrap from.
<code>n</code>	Integer. Number of observations in the output dataset. If NULL (default), uses the same number of rows as the input data.
<code>continuous_vars</code>	Character vector of column names to treat as continuous variables. If NULL (default), automatically detects numeric columns.
<code>cat_vars</code>	Character vector of column names to treat as categorical variables. If NULL (default), automatically detects factors, logical columns, and numeric columns with 10 or fewer unique values.
<code>id_var</code>	Character string specifying the name of the ID variable column. This column will be reset to sequential values (1:n) in the output. Default is "pid".
<code>seed</code>	Integer. Random seed for reproducibility. Default is 123.
<code>noise_level</code>	Numeric between 0 and 1. Controls the amount of noise added. For continuous variables, this is multiplied by the standard deviation to determine noise magnitude. For categorical variables, this is divided by 2 to determine the probability of value changes. Default is 0.1.

Details

The function performs the following operations:

Bootstrap Sampling: Samples `n` observations with replacement from the original dataset.

Continuous Variable Noise:

- Adds Gaussian noise with standard deviation = original SD × noise_level
- Constrains values to remain within original variable bounds
- Preserves integer type for variables that appear to be integers

Categorical Variable Perturbation:

- Changes values with probability = noise_level / 2
- Binary variables: flips to opposite value
- Multi-level unordered: randomly selects from other levels
- Ordered factors: weights selection toward adjacent levels
- Preserves factor levels and ordering from original data

Value

A data frame with the same structure as the input data, containing bootstrap sampled observations with added noise.

Note

- The function assumes that categorical variables with numeric encoding should maintain their numeric type unless they are factors in the input
- Missing values (NA) are handled appropriately in calculations but are not imputed
- For ordered factors or variables named "grade", the perturbation favors transitions to adjacent levels over distant levels

See Also

[sample](#) for bootstrap sampling, [rnorm](#) for noise generation

Examples

```
## Not run:
# Load example dataset
data(gbsg, package = "survival")

# Basic usage with automatic variable detection
synthetic_data <- generate_bootstrap_with_noise(
  data = gbsg,
  seed = 123
)

# Specify variables explicitly
synthetic_data <- generate_bootstrap_with_noise(
  data = gbsg,
  n = 1000,
  continuous_vars = c("age", "size", "nodes", "pgr", "er", "rfstime"),
  cat_vars = c("meno", "grade", "hormon", "status"),
  id_var = "pid",
  seed = 456,
  noise_level = 0.15
)

# Create multiple synthetic datasets
synthetic_list <- lapply(1:10, function(i) {
  generate_bootstrap_with_noise(data = gbsg, seed = i)
})

## End(Not run)
```

```
generate_detection_curve
```

Generate Detection Probability Curve

Description

Computes detection probability across a range of hazard ratios to create a power-like curve for subgroup detection.

Usage

```
generate_detection_curve(
  theta_range = c(0.5, 3),
  n_points = 50L,
  n_sg,
  prop_cens = 0.3,
  hr_threshold = 1.25,
  hr_consistency = 1,
  include_reference = TRUE,
  method = "cubature",
  verbose = TRUE
)
```

Arguments

<code>theta_range</code>	Numeric vector of length 2. Range of HR values to evaluate. Default: <code>c(0.5, 3.0)</code>
<code>n_points</code>	Integer. Number of points to evaluate. Default: 50
<code>n_sg</code>	Integer. Subgroup sample size.
<code>prop_cens</code>	Numeric. Proportion censored (0-1). Default: 0.3
<code>hr_threshold</code>	Numeric. HR threshold for detection. Default: 1.25
<code>hr_consistency</code>	Numeric. HR consistency threshold. Default: 1.0
<code>include_reference</code>	Logical. Include reference HR values (0.5, 0.75, 1.0). Default: TRUE
<code>method</code>	Character. Integration method. Default: "cubature"
<code>verbose</code>	Logical. Print progress. Default: TRUE

Value

A data.frame with columns:

<code>theta</code>	Hazard ratio values
<code>probability</code>	Detection probability
<code>n_sg</code>	Subgroup size (repeated)
<code>prop_cens</code>	Censoring proportion (repeated)
<code>hr_threshold</code>	Detection threshold (repeated)

Examples

```
## Not run:
# Generate detection curve
curve_data <- generate_detection_curve(
  n_sg = 60,
  prop_cens = 0.2,
  hr_threshold = 1.25
)

# Plot
plot_detection_curve(curve_data)

## End(Not run)
```

`generate_gbsg_bootstrap_general`*Generate Synthetic GBSG Data using Generalized Bootstrap*

Description

Generate Synthetic GBSG Data using Generalized Bootstrap

Usage

```
generate_gbsg_bootstrap_general(n = 686, seed = 123, noise_level = 0.1)
```

Arguments

<code>n</code>	Number of observations
<code>seed</code>	Random seed
<code>noise_level</code>	Noise level for perturbation

Value

Synthetic GBSG dataset

Examples

```
## Not run:
df_synth <- generate_gbsg_bootstrap_general(n = 200, seed = 42)
dim(df_synth)
names(df_synth)

## End(Not run)
```

generate_glm_dgm

*Generate GLM-Based Data Generating Mechanism***Description**

Creates a data generating mechanism (DGM) for non-survival outcomes using generalised linear models.

Usage

```
generate_glm_dgm(
  data,
  factor_vars,
  continuous_vars = NULL,
  outcome_var,
  treatment_var,
  outcome_type = c("binary", "continuous", "count"),
  effect_measure = NULL,
  offset_var = NULL,
  subgroup_vars = NULL,
  subgroup_cuts = NULL,
  model = c("alt", "null"),
  k_treat = 1,
  k_inter = 0,
  adverse_outcome = FALSE,
  n_super = 5000L,
  seed = 8316951L,
  verbose = FALSE,
  k_random_noise = 0L,
  noise_seed = 20260807L
)
```

Arguments

<code>data</code>	A data frame containing the source dataset.
<code>factor_vars</code>	Character vector of factor/categorical variable names to include as covariates. These are the candidate confounders for ForestSearch (e.g., <code>paste0("z", 1:12)</code>).
<code>continuous_vars</code>	Character vector of continuous variable names to include as prognostic covariates in the baseline GLM. These improve the model fit but do not participate in subgroup definition. Consistent with generate_aft_dgm_flex , which accepts both <code>continuous_vars</code> and <code>factor_vars</code> . Default NULL (factor-only model, backward compatible).
<code>outcome_var</code>	Character string naming the outcome variable.
<code>treatment_var</code>	Character string naming the treatment variable (must be 0/1 coded).
<code>outcome_type</code>	Character. One of "binary", "continuous", or "count".

effect_measure	Character. Effect measure for the GLM. Default is NULL, which selects "OR" for binary, "MD" for continuous, "IRR" for count outcomes.
offset_var	Character or NULL. Name of the exposure / follow-up time column for count outcomes with an offset. Required when effect_measure is "IRR" or "IRD". The offset enters the Poisson GLM as <code>offset(log(offset_var))</code> . Default NULL.
subgroup_vars	Character vector of variable names defining the subgroup. Default NULL (no subgroup).
subgroup_cuts	Named list of cutpoint specifications for subgroup variables. Uses the same flexible format as generate_aft_dgm_flex : fixed values, quantiles, or functions.
model	Character. "alt" for alternative hypothesis with treatment-subgroup interaction, "null" for uniform treatment effect. Default "alt".
k_treat	Numeric. Scaling factor for the fitted treatment coefficient on the linear predictor scale. $k_{treat} = 1$ (default) preserves the fitted treatment effect as-is. $k_{treat} > 1$ amplifies the treatment main effect, useful when the source dataset has a weak treatment effect and a stronger ITT is desired for simulation. $k_{treat} < 1$ attenuates the effect. Analogous to the k_{treat} parameter in generate_aft_dgm_flex . Note: the null DGM (<code>model = "null"</code>) represents a <i>homogeneous</i> treatment effect (no HTE), not a zero treatment effect. k_{treat} scales the magnitude of that homogeneous effect.
k_inter	Numeric. Direct additive shift on the linear predictor for Q members under treatment. The interpretation depends on outcome_type: binary Shift on the log-odds scale. Positive increases $P(Y=1)$ for Q under treatment. continuous Shift on the mean scale (identity link). Positive increases $E(Y)$ for Q under treatment. count Shift on the log-rate scale. Positive increases $E(Y)$ for Q under treatment (multiplicative: rate multiplied by $\exp(k_{inter})$). Default 0 (no interaction, equivalent to <code>model = "null"</code>).
adverse_outcome	Logical. If TRUE and outcome_type = "binary", the interaction shift k_{inter} is negated internally so that a positive k_{inter} consistently amplifies the treatment contrast on the beneficial (1 - Y) scale. The flag is stored in the returned object for downstream use by calibrate_glm_interaction and run_simulation_analysis . Default FALSE.
n_super	Integer. Size of the super-population. Default 5000L.
seed	Integer. Random seed for super-population sampling. Default 8316951L.
verbose	Logical. Print diagnostic information. Default FALSE.
k_random_noise	Integer. Number of standard-normal noise columns (noise1, noise2, ...) drawn once onto the super-population at construction. Default 0L (no noise columns; output identical to previous versions).
noise_seed	Integer. Seed for the population-level noise draw, used only when $k_{random_noise} > 0$. Default 20260807L, the committed population-noise seed (commit 2a4787bc).

Details

Fits a GLM to the source data, creates a super-population with individual-level potential outcomes, and stores true subgroup effects. The returned object can be passed to [simulate_from_glm_dgm](#) for simulation studies, and is compatible with [run_simulation_analysis](#) (which dispatches on class).

Noise columns requested via `k_random_noise` are inert $N(0,1)$ population attributes intended as candidate confounders; the builder never puts them in the outcome model. They are drawn once onto `df_super` after sampling, so simulated trials inherit them by row resampling.

Value

An object of class `c("glm_dgm", "list")` with:

`df_super` Super-population data frame with potential outcome columns: `p0`, `p1` (binary probabilities), or `mu0`, `mu1` (conditional means for continuous / count outcomes), and `flag_harm`.

`df_source` The observed analysis frame in observed order (every source patient exactly once), carrying the same potential-outcome columns as `df_super` computed by the identical prediction calls, plus `flag_harm` and the discretised covariates. This is the frozen panel consumed by `simulate_from_glm_dgm(baseline = "fixed")`.

`hazard_ratios` Named list with `harm_subgroup`, `no_harm_subgroup`, and `overall` – effect estimates on the scale determined by `effect_measure`. Field name retained for compatibility with `get_dgm_hr()` and reporting functions.

`outcome_type` Character: "binary", "continuous", or "count".

`effect_measure` Character: the effect measure used.

`model_params` List with fitted coefficients, family, `k_inter`, residual SD (continuous), and offset variable name (count).

`subgroup_info` List with subgroup definition, size, and proportion.

`model_type` Character: "alt" or "null".

`noise_names` Character vector of noise column names (`character(0)` when `k_random_noise = 0`).

`noise_scheme` "population" when noise was drawn, otherwise "none".

`noise_seed` The seed used for the noise draw, or `NA_integer_` when none was drawn.

See Also

[simulate_from_glm_dgm](#), [calibrate_glm_interaction](#), [generate_aft_dgm_flex](#)

Examples

```
## Not run:
library(speff2trial)
actg <- subset(ACTG175, arms %in% c(1, 3))
actg$treat <- 1L - as.integer(actg$arms == 1)
actg$y <- as.integer(actg$cd420 > actg$cd40)
actg$z1 <- as.factor(ifelse(actg$age > 34, 1L, 0L))
actg$z2 <- as.factor(ifelse(actg$preanti <= 745, 1L, 0L))
```

```

dgm <- generate_glm_dgm(
  data      = actg,
  factor_vars = c("z1", "z2"),
  outcome_var = "y",
  treatment_var = "treat",
  outcome_type = "binary",
  subgroup_vars = c("z1", "z2"),
  subgroup_cuts = list(z1 = 1L, z2 = 1L),
  model        = "alt",
  k_inter      = 1.0, # additive log-odds shift for Q under treatment
  verbose      = TRUE
)

## End(Not run)

```

get_combinations_info *Get all combinations of subgroup factors up to maxk*

Description

Generates all possible combinations of subgroup factors up to a specified maximum size.

Usage

```
get_combinations_info(L, maxk)
```

Arguments

L	Integer. Number of subgroup factors.
maxk	Integer. Maximum number of factors in a combination.

Value

List with max_count (total combinations) and indices_list (indices for each k).

Examples

```

# L=6 single-factor subgroups, combinations up to 2 factors
info <- get_combinations_info(L = 6, maxk = 2)
info$max_count # total number of combinations
length(info$indices_list)

```

get_conf_force	<i>Get forced cut expressions for variables</i>
----------------	---

Description

For each variable in `conf.force.names`, returns cut expressions if continuous.

Usage

```
get_conf_force(df, conf.force.names, cont.cutoff = 4)
```

Arguments

- `df` Data frame.
- `conf.force.names` Character vector of variable names.
- `cont.cutoff` Integer. Cutoff for continuous.

Value

Character vector of cut expressions.

Examples

```
df <- data.frame(age = c(45, 60, 35), size = c(20, 35, 15), grade = c(1, 2, 3))
get_conf_force(df, conf.force.names = c("age", "size"))
```

get_conf_force_jq	<i>Get J-quantile cut expressions for variables</i>
-------------------	---

Description

For each named variable in `conf_jcuts`, returns `J` deferred cut expressions of the form "`X <= qj(X, k, J + 1)`" via `cut_var_jq()`. Variables that are not continuous (per `cont.cutoff`) are skipped with a warning. This is the `cut_var_jq/J`-quantile analog of `get_conf_force()`.

Usage

```
get_conf_force_jq(df, conf_jcuts, cont.cutoff = 4)
```

Arguments

- `df` Data frame.
- `conf_jcuts` Named list of positive integers. Each name is a continuous-variable column in `df`; each value `J` is the number of binary cut expressions to emit for that variable. The cuts are placed at the $(k/(J+1))$ -th empirical quantiles for $k = 1, \dots, J$, partitioning the variable's range into $J + 1$ non-overlapping intervals.
- `cont.cutoff` Integer. Cutoff for continuous determination (passed through to `is.continuous()`).

Details

Conflicts with other forced-cut mechanisms (defaultcut_names, conf.cont_medians_force) are validated upstream by get_FSdata(); this helper performs only structural checks.

Value

Character vector of cut expressions (length sum(J_v) over continuous v), or character(0) if none.

Examples

```
df <- data.frame(age = c(45, 60, 35, 50, 70, 25, 80, 55))
# 5 binary cuts at the 1/6, 2/6, ..., 5/6 quantiles of age:
get_conf_force_jq(df, conf_jcuts = list(age = 5))
```

get_covs_in

Get indicator vector for selected subgroup factors

Description

Returns a vector indicating which factors are included in a subgroup combination.

Usage

```
get_covs_in(
  kk,
  maxk,
  L,
  counts_1factor,
  index_1factor,
  counts_2factor = NULL,
  index_2factor = NULL,
  counts_3factor = NULL,
  index_3factor = NULL
)
```

Arguments

kk	Integer. Index of the combination.
maxk	Integer. Maximum number of factors in a combination.
L	Integer. Number of subgroup factors.
counts_1factor	Integer. Number of single-factor combinations.
index_1factor	Matrix of indices for single-factor combinations.
counts_2factor	Integer. Number of two-factor combinations.
index_2factor	Matrix of indices for two-factor combinations.
counts_3factor	Integer. Number of three-factor combinations.
index_3factor	Matrix of indices for three-factor combinations.

Value

Numeric vector indicating selected factors (1 = included, 0 = not included).

Examples

```
# Build index matrices directly (L=4, maxk=2)
L <- 4
idx1 <- matrix(seq_len(L), ncol = 1)      # single-factor indices
idx2 <- t(utils::combn(L, 2))             # two-factor indices
# Combination 3 falls in the single-factor block (kk=3 -> factor 3)
get_covs_in(kk = 3, maxk = 2, L = L,
             counts_1factor = nrow(idx1),
             index_1factor  = idx1,
             counts_2factor = nrow(idx2),
             index_2factor  = idx2)
```

get_Cox_sg	<i>Fit Cox Model for Subgroup</i>
------------	-----------------------------------

Description

Fits a Cox model for a subgroup and returns estimate and standard error.

Usage

```
get_Cox_sg(
  df_sg,
  cox.formula,
  est.loghr = TRUE,
  cox_initial = log(1),
  treat.name = NULL
)
```

Arguments

df_sg	Data frame for subgroup.
cox.formula	Cox model formula.
est.loghr	Logical. Is estimate on log(HR) scale?
cox_initial	Optional pre-fitted Cox model object to use instead of fitting a new model. Default NULL
treat.name	Character or NULL. Name of the treatment term whose coefficient is the effect of interest. When the formula is adjusted for covariates the fit has multiple coefficients; supplying treat.name extracts only that term's estimate and SE. NULL (default) preserves the historical behaviour of returning every coefficient (correct for the treatment-only formula).

Details

Function is utilized throughout codebase

Value

List with estimate and standard error.

Examples

```
## Not run:
library(survival)
df <- data.frame(
  tte = gbsg$rfstime / 30.4375,
  event = gbsg$status,
  treat = gbsg$hormon
)
formula <- build_cox_formula("tte", "event", "treat")
result <- get_Cox_sg(df, cox.formula = formula)
exp(result$est_obs) # hazard ratio

## End(Not run)
```

get_dfpred	<i>Generate Prediction Dataset with Subgroup Treatment Recommendation</i>
------------	---

Description

Creates a prediction dataset with a treatment recommendation flag based on the subgroup definition. Supports both label expressions (e.g., " $\{er \leq 0\}$ ") and bare column names (e.g., "q3.1").

Usage

```
get_dfpred(df.predict, sg.harm, version = 1)
```

Arguments

df.predict	Data frame for prediction (test or validation set).
sg.harm	Character vector of subgroup-defining labels. Values may be wrapped in braces and optionally negated, e.g. " $\{er \leq 0\}$ " or " $!\{size \leq 35\}$ ". Plain column names (e.g., "q3.1") are treated as binary indicators that must equal 1.
version	Integer; encoding version (maintained for backward compatibility). Default: 1.

Details

Each element of `sg.harm` is processed as follows:

1. Outer braces and leading `!` are stripped.
2. If the result matches `"var op value"` (where `op` is one of `<=`, `<`, `>=`, `>`, `==`, `!=`), the comparison is executed directly on `df.predict[[var]]`.
3. Otherwise the expression is treated as a column name and membership is `df.predict[[name]] == 1`.

Value

Data frame with treatment recommendation flag (`treat.recommend`): 0 for harm subgroup, 1 for complement.

See Also

[evaluate_comparison](#) for the operator-dispatch logic, [forestsearch](#) for the main analysis function.

Examples

```
## Not run:
# With brace-wrapped label expressions
sg <- c("{er <= 0}", "{size <= 35}")
df_out <- get_dfpred(df.predict = test_data, sg.harm = sg)

# With negation
sg_neg <- c("{er <= 0}", "!("{size <= 35}")")
df_neg <- get_dfpred(df.predict = test_data, sg.harm = sg_neg)

# With bare column names (binary indicators)
sg_col <- c("q1.1", "q3.1")
df_col <- get_dfpred(df.predict = encoded_data, sg.harm = sg_col)

## End(Not run)
```

get_dfRes

Bootstrap Confidence Interval and Bias Correction Results

Description

Calculates confidence intervals and bias-corrected estimates for bootstrap results.

Usage

```
get_dfRes(
  Hobs,
  seHobs,
  H1_adj,
  H2_adj = NULL,
  ystar,
  cov_method = "standard",
  cov_trim = 0,
  est.scale = "hr",
  est.loghr = TRUE
)
```

Arguments

Hobs	Numeric. Observed estimate.
seHobs	Numeric. Standard error of observed estimate.
H1_adj	Numeric. Bias-corrected estimate 1.
H2_adj	Numeric. Bias-corrected estimate 2 (optional).
ystar	Matrix of bootstrap samples.
cov_method	Character. Covariance method ("standard" or "nocorrect").
cov_trim	Numeric. Trimming proportion for covariance (default: 0.0).
est.scale	Character. "hr" or "1/hr".
est.loghr	Logical. Is estimate on log(HR) scale?

Value

Data.table with confidence intervals and estimates.

Examples

```
## Not run:
# get_dfRes is called internally by forestsearch_bootstrap_dofuture().
# To use directly, supply bootstrap sample matrix from bootstrap_ystar():
# ystar <- bootstrap_ystar(df, nb_boots = 100, seed = 42L)
# res <- get_dfRes(Hobs = log(1.5), seHobs = 0.3,
#                  H1_adj = 0.05, H2_adj = 0.03,
#                  ystar = ystar)

## End(Not run)
```

get_FSdata

*ForestSearch Data Preparation and Feature Selection***Description**

Prepares a dataset for ForestSearch, including options for LASSO-based dimension reduction, GRF cuts, forced cuts, and flexible cut strategies. Returns a list with the processed data, subgroup factor names, cut expressions, and LASSO selection results.

Usage

```
get_FSdata(
  df.analysis,
  use_lasso = FALSE,
  use_grf = FALSE,
  grf_cuts = NULL,
  confounders.name,
  cont.cutoff = 4,
  conf_force = NULL,
  conf.cont_medians = NULL,
  conf.cont_medians_force = NULL,
  conf.cont_jcuts = NULL,
  dina_cuts = NULL,
  collapse_cuts = TRUE,
  collapse_cuts_args = list(),
  replace_med_grf = TRUE,
  defaultcut_names = NULL,
  cut_type = "default",
  exclude_cuts = NULL,
  outcome.name = "tte",
  event.name = "event",
  details = TRUE,
  outcome_type = "survival",
  offset.name = NULL
)
```

Arguments

<code>df.analysis</code>	Data frame containing the data.
<code>use_lasso</code>	Logical. Whether to use LASSO for dimension reduction.
<code>use_grf</code>	Logical. Whether to use GRF cuts.
<code>grf_cuts</code>	Character vector of GRF cut expressions.
<code>confounders.name</code>	Character vector of confounder variable names.
<code>cont.cutoff</code>	Integer. Cutoff for continuous variable determination.

<code>conf_force</code>	Character vector of forced cut expressions.
<code>conf.cont_medians</code>	Character vector of continuous confounders to cut at median.
<code>conf.cont_medians_force</code>	Character vector of additional continuous confounders to force median cut.
<code>conf.cont_jcuts</code>	Named list of positive integers, one per continuous confounder for which J-quantile cuts are desired. For a variable X listed as $X = J$, the default cut set (<code>cut_var()</code>: mean, median, Q1, Q3) is replaced by J binary cut points at the $(k/(J+1))$-th quantiles of X for $k = 1, \dots, J$, defining $J+1$ non-overlapping intervals $[\min(X), c_1), [c_1, c_2), \dots, [c_J, \max(X)].$ Variables not listed here retain default behaviour. J-quantile cuts are unconditional (not filtered by LASSO), matching <code>defaultcut_names</code> semantics. Names must be in <code>confounders.name</code> and must not overlap with <code>defaultcut_names</code> or <code>conf.cont_medians_force</code>.
<code>dina_cuts</code>	Character vector of DINA cut expressions (optional), merged into the candidate pool exactly like <code>grf_cuts</code> .
<code>collapse_cuts</code>	Logical. If TRUE, collapse near-redundant continuous candidate cuts after the full pool is assembled and resolved to literal numerics. Cuts on the same variable and operator whose thresholds lie within a per-variable standard-error band are merged to a single rounded-centroid threshold, subject to a membership safety check; see collapse_redundant_cuts . Categorical, indicator, bare-name and equality cuts are untouched. Default TRUE; pass <code>collapse_cuts = FALSE</code> to recover the un-coarsened candidate pool used by earlier package versions.
<code>collapse_cuts_args</code>	List of overrides for the coarsening, merged onto the defaults <code>list(c = 1.0, tol = 0.05, digits = 0L)</code> : <code>c</code> is the band multiplier (<code>band = c * sd(x)/sqrt(n)</code>), <code>tol</code> the membership safety tolerance (fraction of n when < 1 , absolute count when ≥ 1), and <code>digits</code> the rounding for the representative threshold. Ignored when <code>collapse_cuts = FALSE</code> .
<code>replace_med_grf</code>	Logical. If TRUE, removes median cuts that overlap with GRF cuts.
<code>defaultcut_names</code>	Character vector of confounders to force default cuts.
<code>cut_type</code>	Character. "default" or "median" for cut strategy.
<code>exclude_cuts</code>	Character vector of cut expressions to exclude.
<code>outcome.name</code>	Character. Name of outcome variable.
<code>event.name</code>	Character. Name of event indicator variable.
<code>details</code>	Logical. If TRUE, prints details during execution.
<code>outcome_type</code>	Character. One of "survival" (default), "binary", or "continuous".
<code>offset.name</code>	Character or NULL. Name of the follow-up time column for rate-based measures (IRR, IRD).

Value

- A list with components:
 - df Data frame with derived subgroup factor columns.
 - confs_names Character vector of factor column names added to df.
 - confs Named character vector of cut expressions (continuous cuts plus categorical indicators).
 - lassokeep Character vector of confounders retained by LASSO screening (empty if use_lasso = FALSE).
 - lassoomit Character vector of confounders dropped by LASSO screening (empty if use_lasso = FALSE).

Examples

```
library(survival)
df <- survival::gbsg
df$grade3 <- as.integer(df$grade == "3")
fs_data <- get_FSdata(df.analysis = df,
  confounders.name = c("age", "meno", "size", "grade3", "nodes", "pgr", "er"),
  outcome.name = "rfstime", event.name = "status",
  use_lasso = FALSE, use_grf = FALSE)
names(fs_data)
```

get_leff	<i>Get L_{eff} at a Given Sample Size</i>
----------	---

Description

Convenience function: loads calibration and computes $L_{\text{eff}}(N) = C \cdot (N/n_{\text{min}})^\alpha$.

Usage

```
get_leff(N, outcome_type, output_dir = NULL)
```

Arguments

- N Integer. Sample size.
- outcome_type Character. Outcome type to load.
- output_dir Character. Default: NULL (auto).

Value

Numeric L_{eff} value.

get_split_hr_fast	<i>Fast Cox Model HR Estimation</i>
-------------------	-------------------------------------

Description

Fits a minimal Cox model to estimate hazard ratio with reduced overhead. Disables robust variance, model matrix storage, and other extras for speed.

Usage

```
get_split_hr_fast(df, cox_init = 0, adjust_covariates = NULL)
```

Arguments

df	data.frame or data.table with Y, Event, Treat columns.
cox_init	Numeric. Initial value for coefficient (default 0). Ignored when adjust_covariates is supplied (the default coxph starting values are used so the init length matches the number of model coefficients).
adjust_covariates	Character vector or NULL. Additional terms appended to the model right-hand side after Treat. Terms are pasted verbatim, so "strata(x1)" produces a stratified Cox model. The referenced columns must be present in df. NULL (default) fits the unadjusted $\text{Surv}(Y, \text{Event}) \sim \text{Treat}$ model.

Value

Numeric. Estimated hazard ratio for Treat, or NA if model fails.

Examples

```
set.seed(42)
df <- data.frame(
  Y      = rexp(80),
  Event  = rbinom(80, 1, 0.6),
  Treat  = rep(0:1, 40)
)
get_split_hr_fast(df)
```

```
get_subgroup_membership
```

Get subgroup membership vector

Description

Returns a vector indicating subgroup membership (1 if all selected factors are present, 0 otherwise).

Usage

```
get_subgroup_membership(zz, covs.in)
```

Arguments

`zz` Matrix or data frame of subgroup factor indicators.
`covs.in` Numeric vector indicating which factors are selected (1 = included).

Value

Numeric vector of subgroup membership (1/0).

Examples

```
zz <- matrix(c(1, 0, 1, 1, 0, 1), nrow = 3,
              dimnames = list(NULL, c("er_le0", "grade3")))
get_subgroup_membership(zz, covs.in = c(1, 1)) # both factors required
get_subgroup_membership(zz, covs.in = c(1, 0)) # first factor only
```

```
gg_forest                      ggplot2 / patchwork forest plot
```

Description

Creates a publication-quality forest plot using ggplot2 for the CI panel and patchwork to assemble label and annotation columns alongside it. Unlike forestploter, `fig.height` maps directly to row density — `row_height = fig.height / n_rows` with no hidden scaling.

Usage

```
gg_forest(
  subgroups,
  est,
  lo,
  hi,
  cat_vec = NULL,
  cat_colours = NULL,
```

```

  annot = NULL,
  ref_line = 1,
  vert_lines = NULL,
  ref_col = "firebrick",
  ref_lty = "dashed",
  vert_col = "grey50",
  vert_lty = "dotted",
  xlim = NULL,
  ticks_at = NULL,
  tick_labels = NULL,
  xlog = TRUE,
  xlab = "Hazard Ratio",
  title = NULL,
  subtitle = NULL,
  footnote = NULL,
  point_size = 2.5,
  line_size = 0.8,
  point_shape = 21,
  base_size = 11,
  widths = NULL,
  row_expand = 0.6,
  clip_marker = c("none", "arrow")
)

```

Arguments

subgroups	Character vector of subgroup names (displayed top to bottom).
est	Numeric vector of point estimates (median HR or similar).
lo	Numeric vector of lower bounds (e.g. 1st percentile ECI).
hi	Numeric vector of upper bounds (e.g. 99th percentile ECI).
cat_vec	Optional character vector of category labels (one per row). Used to colour CI lines and label text.
cat_colours	Optional named character vector mapping category labels to colours. Defaults to grey for all rows.
annot	Optional named list of character vectors, one per annotation column. Names become column headers. Each vector must match length(subgroups).
ref_line	Numeric. X position of the primary reference line (default 1). Drawn as a dashed red line.
vert_lines	Numeric vector. X positions of secondary vertical lines (default NULL). Drawn as dotted grey lines.
ref_col	Colour of the primary reference line (default "firebrick").
ref_lty	Line type of the primary reference line (default "dashed").
vert_col	Colour of secondary vertical lines (default "grey50").
vert_lty	Line type of secondary vertical lines (default "dotted").
xlim	Numeric vector length 2. X-axis limits for the CI panel.

ticks_at	Numeric vector. X-axis tick positions.
tick_labels	Character vector. Custom tick labels (default: <code>as.character(ticks_at)</code>).
xlog	Logical. If TRUE (default), x-axis on log scale.
xlab	Character. X-axis label (default "Hazard Ratio").
title	Character. Overall plot title (default NULL).
subtitle	Character. Plot subtitle (default NULL).
footnote	Character. Footnote appended below the CI panel (default NULL).
point_size	Numeric. Size of point estimate symbol (default 2.5).
line_size	Numeric. Line width of CI segments (default 0.8).
point_shape	Integer. pch for point estimates (default 21, filled circle).
base_size	Numeric. ggplot2 base font size in pt (default 11). Controls all text — increase to make the plot larger; no other knob needed.
widths	Numeric vector. Relative patchwork column widths: <code>c(label, ci, annot_1, annot_2, ...)</code> . Default: <code>c(3.5, 5, rep(1, n_annot))</code> .
row_expand	Numeric. Extra space above and below row range on y-axis, in row units (default 0.6).
clip_marker	Character. How to mark whiskers whose 1st or 99th percentile falls outside <code>xlim</code> . One of "none" (default — the whisker simply reaches the axis edge) or "arrow" (a small arrowhead is drawn at the clipped boundary, pointing outward, indicating the distribution extends beyond the visible range).

Value

A patchwork object. Render with `print()` or `plot()`. Control dimensions entirely via knitr chunk options `fig.width` / `fig.height`: `row height = fig.height / n_rows`.

Examples

```
## Not run:
# Recommended fig.height: n_rows * 0.45 + 1.5 (for title/axis overhead)
# e.g. 20 rows -> fig.height = 20 * 0.45 + 1.5 = 10.5

## End(Not run)
```

glm_cs_fit

Fit GLM with Spline Treatment-Biomarker Interaction

Description

Estimates treatment effects as a function of a continuous covariate using a generalized linear model with natural cubic splines. This is the GLM analog of `cox_cs_fit` for binary, continuous, and count outcomes.

Usage

```
glm_cs_fit(
  df,
  outcome_name = "event",
  treat_name = "treat",
  z_name = "bm",
  family = "binomial",
  offset_name = NULL,
  alpha = 0.05,
  spline_df = 3,
  z_by = 0.05,
  z_quantile = 0.95,
  z_max = Inf,
  z_window = 0,
  show_plot = FALSE,
  plot_params = NULL,
  truebeta_name = NULL,
  verbose = TRUE
)

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

Arguments

<code>df</code>	Data frame containing outcome data.
<code>outcome_name</code>	Character. Name of the outcome variable. Default: "event".
<code>treat_name</code>	Character. Name of the treatment variable (0/1). Default: "treat".
<code>z_name</code>	Character. Name of the continuous biomarker covariate. Default: "bm".
<code>family</code>	Character or family object. One of "binomial" (default), "poisson", or "gaussian". Determines the link function and the scale of the treatment-effect profile: "binomial" uses logit link (profile is log-OR); "poisson" uses log link (profile is log-IRR, requires <code>offset_name</code>); "gaussian" uses identity link (profile is mean difference).
<code>offset_name</code>	Character or NULL. Name of the offset variable (e.g., log follow-up time for Poisson models). The offset is applied as <code>log(offset_var)</code> internally. Required when <code>family = "poisson"</code> .
<code>alpha</code>	Numeric. Significance level for confidence intervals (two-sided). Default: 0.05 (95 percent CI).
<code>spline_df</code>	Integer. Degrees of freedom for the natural spline. Default: 3.
<code>z_by</code>	Numeric. Increment for the biomarker prediction grid. Default: 0.05.
<code>z_quantile</code>	Numeric (0–1). Upper quantile for the prediction grid. Default: 0.95 (5th to 95th percentile).
<code>z_max</code>	Numeric. Maximum z value for predictions. Default: Inf (no truncation).

<code>z_window</code>	Numeric. Half-width for counting observations near each grid point. Default: 0.0.
<code>show_plot</code>	Logical. Display a base-R diagnostic plot. Default: FALSE.
<code>plot_params</code>	List. Optional plot parameter overrides (see cox_cs_fit for format).
<code>truebeta_name</code>	Character or NULL. Name of a column containing true log-effect values (for simulation validation).
<code>verbose</code>	Logical. Print diagnostic information. Default: TRUE.
<code>x</code>	A <code>glm_cs_fit</code> object.
<code>...</code>	Additional arguments (unused).

Details

Model Structure:

For binary outcomes (`family = "binomial"`):

$$\text{logit}(P(Y = 1|Z, A)) = \beta_0 A + f(Z) + g(Z) \cdot A$$

For count outcomes with offset (`family = "poisson"`):

$$\log(E[Y|Z, A]) = \beta_0 A + f(Z) + g(Z) \cdot A + \log(t)$$

For continuous outcomes (`family = "gaussian"`):

$$E[Y|Z, A] = \beta_0 A + f(Z) + g(Z) \cdot A$$

The treatment-effect profile $\beta(Z) = \beta_0 + g(Z)$ is estimated on the link scale (log-OR, log-IRR, or identity) with pointwise delta-method confidence intervals.

Value

A list of class `"glm_cs_fit"` containing:

- `z_profile`** Numeric vector. Biomarker grid values.
- `loghr_est`** Numeric vector. Point estimates of the log treatment effect (log-OR, log-IRR, or MD) at each grid point.
- `loghr_lower`** Numeric vector. Lower confidence bounds.
- `loghr_upper`** Numeric vector. Upper confidence bounds.
- `se_loghr`** Numeric vector. Standard errors (delta method).
- `counts_profile`** Integer vector. Observation counts near each grid point.
- `glm_primary`** Numeric. Overall treatment effect from the no-interaction GLM.
- `model_fit`** The fitted `glm` object.
- `spline_basis`** The natural spline basis object.
- `family`** Character. The GLM family used.
- `effect_label`** Character. Human-readable label for the effect scale (e.g., "log(OR)", "log(IRR)", "Mean Difference").

lrt_pvalue Numeric. P-value from the likelihood ratio test comparing the interaction model to the main-effects-only model.

alpha Numeric. Significance level used.

ci_level Numeric. Confidence level (1 - alpha).

Invisibly returns x.

See Also

[cox_cs_fit](#) for the survival analog, [glm_effect_profile](#) for the extended interface with sandwich SEs and overdispersion correction.

Examples

```
## Not run:
# Binary outcome: log-OR profile over a biomarker
set.seed(42)
df <- data.frame(
  event = rbinom(500, 1, 0.3),
  treat = rbinom(500, 1, 0.5),
  bm    = rnorm(500, 0, 1)
)
result <- glm_cs_fit(df, outcome_name = "event", z_name = "bm",
                    family = "binomial")

# Poisson outcome with offset: log-IRR profile
df$ftime <- rexp(500, 0.01) + 1
result_irr <- glm_cs_fit(df, outcome_name = "event", z_name = "bm",
                       family = "poisson", offset_name = "ftime")

## End(Not run)
```

glm_effect_profile	<i>GLM Treatment-Effect Profile for Continuous Biomarker Interaction</i>
--------------------	--

Description

Estimates a treatment effect profile across a continuous biomarker using a generalized linear model with natural cubic spline interactions. Supports binary, continuous, and count or rate outcomes. Standard errors are computed via the delta method applied to the full model covariance, with optional sandwich variance estimation for modified-Poisson models.

Usage

```
glm_effect_profile(
  df,
  outcome.name,
  treat.name = "treat",
```

```

  z.name = "bm",
  effect_measure = c("log_OR", "log_RR", "log_IRR", "RD", "MD"),
  offset.name = NULL,
  overdispersion = c("none", "quasi", "negbin"),
  spline_df = 3L,
  z_by = 0.05,
  z_quantile = 0.95,
  alpha = 0.05,
  conf.level = NULL,
  strata.name = NULL,
  show_plot = FALSE,
  verbose = FALSE
)

```

Arguments

<code>df</code>	Data frame containing the analysis data.
<code>outcome.name</code>	Character. Name of the outcome variable. Binary outcomes must be coded 0/1; count outcomes must be non-negative integers; continuous outcomes may be any numeric.
<code>treat.name</code>	Character. Name of the binary treatment indicator (0 = control, 1 = treated). Default: "treat".
<code>z.name</code>	Character. Name of the continuous biomarker (effect modifier). Default: "bm".
<code>effect_measure</code>	Character. Effect measure to estimate on the link scale. Allowed values: "log_OR" (log odds ratio), "log_RR" (log relative risk, modified Poisson), "log_IRR" (log incidence rate ratio, requires offset), "RD" (risk difference), "MD" (mean difference). Default: "log_OR".
<code>offset.name</code>	Character or NULL. Column name for person-time or exposure. When provided, <code>log(offset)</code> enters the linear predictor; required for <code>effect_measure = "log_IRR"</code> . Default: NULL.
<code>overdispersion</code>	Character. Overdispersion correction for Poisson family models. "none" uses standard Poisson (model-based SEs or sandwich if <code>effect_measure = "log_RR"</code>); "quasi" uses quasi-Poisson; "negbin" uses negative binomial via MASS: <code>glm.nb</code> (requires MASS). Default: "none".
<code>spline_df</code>	Integer. Degrees of freedom for natural spline basis. Default: 3.
<code>z_by</code>	Numeric. Increment for the biomarker prediction grid. Default: 0.05.
<code>z_quantile</code>	Numeric. Upper quantile of the biomarker used as the grid endpoint (avoids extrapolation into sparse tails). Default: 0.95.
<code>alpha</code>	Numeric. Two-sided significance level for confidence intervals. Default: 0.05.
<code>conf.level</code>	Numeric or NULL. Confidence level; overrides alpha when supplied. Default: NULL.
<code>strata.name</code>	Character or NULL. Optional stratification variable entered as a factor covariate. Default: NULL.
<code>show_plot</code>	Logical. Display a base-R diagnostic plot of the estimated treatment effect profile. Default: FALSE.
<code>verbose</code>	Logical. Print diagnostic messages. Default: FALSE.

Value

An object of class "glm_effect_profile": a named list containing `z_profile` (biomarker grid), `est` (point estimates on the link scale), `lower` and `upper` (confidence bounds), `se` (standard errors), `effect_measure`, `family_used`, `overdispersion`, `dispersion` (quasi or negbin dispersion estimate, else NA), `model_fit`, `spline_basis`, and `alpha`.

See Also

[glm_cs_fit](#) for a simpler family-based interface, [cox_cs_fit](#) for survival outcomes, [grf.subg.harm.glm](#) for GRF-based subgroup identification.

<code>grf.subg.eval</code>	<i>Evaluates the performance of GRF-identified subgroups, including hazard ratios, bias, and predictive values. This function is typically used in simulation studies to assess the performance of the GRF subgroup identification method.</i>
----------------------------	--

Description

Evaluates the performance of GRF-identified subgroups, including hazard ratios, bias, and predictive values. This function is typically used in simulation studies to assess the performance of the GRF subgroup identification method.

Usage

```
grf.subg.eval(
  df,
  grf.est,
  dgm,
  cox.formula.sim,
  cox.formula.adj.sim,
  analysis = "GRF",
  frac.tau = 1
)
```

Arguments

<code>df</code>	Data frame containing the analysis data.
<code>grf.est</code>	List. Output from <code>grf.subg.harm.survival</code> .
<code>dgm</code>	List. Data-generating mechanism (truth) for simulation.
<code>cox.formula.sim</code>	Formula for unadjusted Cox model.
<code>cox.formula.adj.sim</code>	Formula for adjusted Cox model.
<code>analysis</code>	Character. Analysis label (default: "GRF").
<code>frac.tau</code>	Numeric. Fraction of tau for GRF horizon (default: 1.0).

Value

A data frame with evaluation metrics.

Examples

```
## Not run:
# grf.subg.eval() is called internally to evaluate GRF subgroup quality.
# See grf.subg.harm.survival() for the standard entry point.

## End(Not run)
```

grf.subg.harm.glm

GRF Subgroup Identification for GLM Outcomes

Description

Identifies treatment effect subgroups for binary, continuous, and count outcomes using Generalized Random Forests (GRF) for candidate factor screening, followed by exhaustive subgroup enumeration with GLM-based effect estimation.

Usage

```
grf.subg.harm.glm(
  data,
  confounders.name,
  outcome.name,
  treat.name = "treat",
  id.name = "id",
  outcome_type = c("binary", "continuous", "count"),
  effect_measure = NULL,
  offset.name = NULL,
  overdispersion = c("none", "quasi", "negbin"),
  grf_count_transform = c("log", "identity"),
  n.min = 60L,
  dmin.grf = 0,
  frac.tau = 0.5,
  maxdepth = 2L,
  RCT = TRUE,
  sg.criterion = c("mDiff", "Nsg"),
  conf.level = 0.95,
  seedit = 8316951L,
  return_selected_cuts_only = FALSE,
  adverse_outcome = FALSE,
  tune_grf = FALSE,
  grf_selection = c("tree", "frontier"),
  frontier_rule = c("effMaxSG", "eff", "maxSG", "minSG", "effMinSG"),
  effect_neighborhood = 0.1,
```

```

    selection_rule = c("neighborhood", "pareto", "both"),
    details = FALSE,
    verbose = FALSE
  )

```

Arguments

<code>data</code>	Data frame containing the analysis data.
<code>confounders.name</code>	Character vector of potential effect modifier (confounder) names. These are the candidate split variables for GRF and the subgroup enumeration.
<code>outcome.name</code>	Character. Name of the outcome column.
<code>treat.name</code>	Character. Name of the binary treatment indicator (0/1). Default: "treat".
<code>id.name</code>	Character. Name of the subject ID column. Default: "id".
<code>outcome_type</code>	Character. Type of outcome. One of "binary" (outcome is 0/1; use <code>effect_measure</code> to select log-OR, log-RR, or RD), "continuous" (estimates mean difference), or "count" (count or rate outcome; use <code>effect_measure = "log_IRR"</code> with <code>offset.name</code> for incidence rate ratios). Default: "binary".
<code>effect_measure</code>	Character. Effect measure to estimate. One of "log_OR" (log odds ratio, binary/logistic), "log_RR" (log relative risk, binary/modified Poisson), "RD" (risk difference, binary/identity link), "MD" (mean difference, continuous/Gaussian), or "log_IRR" (log incidence rate ratio, count/Poisson with offset). Default is "log_OR" for binary, "MD" for continuous, and "log_IRR" for count.
<code>offset.name</code>	Character or NULL. Name of the person-time (exposure) column. When supplied, <code>log(offset)</code> enters the linear predictor of the GLM as an offset term. Required when <code>outcome_type = "count"</code> and <code>effect_measure = "log_IRR"</code> . Ignored for binary and continuous outcomes. Default: NULL.
<code>overdispersion</code>	Character. Overdispersion correction for count models. One of "none" (standard Poisson, model-based SEs), "quasi" (quasi-Poisson, dispersion-corrected SEs), or "negbin" (negative binomial via MASS::glm.nb()). Ignored for binary and continuous outcomes. Default: "none".
<code>grf_count_transform</code>	Character. Transformation applied to the count outcome <i>before</i> passing it to <code>grf::causal_forest()</code> for factor screening. Has no effect on binary or continuous outcomes. "log" (default) applies $Y^* = \log(Y + 0.5)$, which prevents $\log(0)$ when zero counts are present and approximates variance stabilisation for Poisson data. "identity" passes raw counts $Y^* = Y$ directly, appropriate when counts are large and rarely zero. Both options affect only GRF factor screening; all subgroup effect estimates use the correctly specified GLM regardless. Default: "log".
<code>n.min</code>	Integer. Minimum subgroup sample size for a valid split. Default: 60.
<code>dmin.grf</code>	Numeric. Minimum absolute CATE magnitude (on the doubly robust score scale) required for the identified subgroup to be declared valid. The score equals $-CATE(H)$, so a positive threshold requires the subgroup CATE to be sufficiently negative (treatment hurts). Default: 0.

frac.tau	Numeric. Fraction of the sample used for the GRF horizon (time-horizon analogue for non-survival outcomes). Default: 0.5.
maxdepth	Integer. Maximum depth for GRF policy trees. Default: 2.
RCT	Logical. Whether data come from a randomised trial. When TRUE, propensity scores are fixed at 0.5. Default: TRUE.
sg.criterion	Character. Subgroup selection criterion. One of "mDiff" (maximum score difference) or "Nsg" (largest valid subgroup). Default: "mDiff".
conf.level	Numeric. Confidence level for subgroup effect estimates. Default: 0.95.
seedit	Integer. Random seed for GRF. Default: 8316951.
return_selected_cuts_only	Logical. If TRUE, returns only cuts from the tree depth that identified the selected subgroup. If FALSE (default), returns all cuts from all fitted trees. Matches the interface of grf.subg.harm.survival . Default: FALSE.
adverse_outcome	Logical. If TRUE, a higher outcome value indicates a worse result (e.g., adverse event count), so the "harm" subgroup is where treatment increases the outcome. If FALSE (default), higher values are better. Default: FALSE.
tune_grf	Logical. If TRUE, enables cross-validated hyperparameter tuning for the causal forest via <code>tune.parameters = "all"</code> , which tunes <code>min.node.size</code> , <code>mtry</code> , <code>sample.fraction</code> , <code>alpha</code> , and <code>imbalance.penalty</code> . If FALSE (default), all GRF parameters remain at their defaults, preserving existing behavior. Most impactful for observational data where nuisance models benefit from tuning; defaults are near-optimal for RCTs (Dandl et al., 2024).
grf_selection	Character, one of "tree" (default) or "frontier". "tree" uses the GRF policy tree (standard behavior). "frontier" is an experimental alternative that, on the same doubly-robust scores, enumerates single-covariate thresholds (and depth-2 covariate-pair conjunctions when <code>maxdepth</code> $\geq$ 2), takes the Pareto frontier of (harm-effect, size), and selects one subgroup under <code>frontier_rule</code> . The selected subgroup is always a single conjunction (never the disjunctive union the tree path can return). Provided for comparison and exploration, not as a recommended default.
frontier_rule	Character, one of "effMaxSG" (default), "eff", "maxSG", "minSG", or "effMinSG"; the rule applied to the frontier when <code>grf_selection = "frontier"</code> . See grf.subg.harm.survival .
effect_neighborhood	Numeric in (0, 1); relative neighborhood for the "effMaxSG" / "effMinSG" rules. Default 0.10. Used only when <code>grf_selection = "frontier"</code> .
selection_rule	Character, one of "neighborhood" (default), "pareto", "both"; the inclusion band for the eff*SG frontier rules. Inert for the other rules.
details	Logical. Print GRF diagnostic information. Default: FALSE.
verbose	Logical. Print progress messages. Default: FALSE.

Details

For count / rate outcomes, the GRF screening step applies a variance-stabilising $\log(Y + 0.5)$ transformation so that `grf::causal_forest()` can be applied without dedicated count-specific

infrastructure. All subgroup effect estimates are computed from correctly specified Poisson, quasi-Poisson, or negative-binomial GLMs via `create_glm_row`.

GRF Screening for Count Outcomes:

`grf::causal_forest()` is designed for continuous outcomes. For count data, this function transforms Y before passing it to GRF according to the `grf_count_transform` argument:

"log" (default) Applies $Y_{\text{star}} = \log(Y + 0.5)$. The +0.5 shift avoids $\log(0)$ for zero-count observations; the log scale approximates variance stabilisation for Poisson data and is recommended when any counts are zero or small.

"identity" Passes raw counts $Y_{\text{star}} = Y$ directly. Appropriate when counts are large, rarely zero, and the raw scale is near-continuous (e.g., daily hospitalisation volumes in a large centre).

In both cases the transformation applies *only* to GRF factor screening and policy-tree cut generation – it determines *which variables* and *which cut-points* are candidate subgroup definitions. The actual effect estimates in each subgroup are always computed from a correctly specified Poisson (or quasi-Poisson / negative-binomial) GLM via `create_glm_row`, regardless of `grf_count_transform`. Alternatively, setting `use_grf = FALSE` in the parent `forestsearch` call routes to LASSO-based factor selection, which supports `family = "poisson"` natively via `glmnet`.

Offset and Person-Time:

When `offset.name` is supplied, $\log(\text{exposure}_i)$ enters every GLM fit in the subgroup enumeration loop, so that subgroup effects are on the incidence-rate-ratio scale adjusted for differential follow-up.

Value

A list of class `"grf_glm_result"` containing:

sg.harm.id Character vector of cut expressions defining the identified subgroup (e.g., `c("v1=1", "v3=1")`), length equal to the depth of the selected policy tree. NULL if no subgroup was found. **Not** a per-subject membership indicator; see the *Field naming collision* section below.

data The input data frame with a `treat.recommend` column added (0 = in harm/questionable subgroup, 1 = complement).

tree.cuts Named list of GRF policy-tree split points.

grf_varimp Named numeric vector of GRF variable importances.

outcome.name Character: name of the outcome column.

treat.name Character: name of the treatment column.

id.name Character: name of the subject-identifier column.

confounders.name Character vector of covariate column names.

effect_measure Character: the effect measure used.

outcome_type Character: the outcome type.

overdispersion Character: the overdispersion correction applied.

offset.name Character or NULL: the offset variable name.

grf_count_transform Character: the transform argument as matched ("log" or "identity").

grf_y_transform Character: transformation actually applied to Y for GRF screening (" $\log(Y + 0.5)$ ", "identity", or "none").

Field naming collision with forestsearch / subgroup.consistency

The field name `sg.harm.id` has **different semantics** on this object versus on the `grp.consistency` list returned by [subgroup.consistency](#) (and nested inside the [forestsearch](#) result):

Object	<code>sg.harm.id</code> contains	Length / type
<code>grf.subg.harm.glm()</code> result (this function)	character vector of cut expressions	character, length = depth of selected tr
<code>subgroup.consistency()</code> result	per-subject 0/1 membership indicator	integer, length <code>nrow(df)</code>

Practical consequence. On this object, pasting `paste(obj$sg.harm.id, collapse = " & ")` correctly renders the identified subgroup. On a `grp.consistency` list the same expression silently concatenates a long 0/1 indicator and produces output like `"0 & 0 & 1 & 0 & ..."`. When writing code that must handle both object types, dispatch on class first, or use the top-level `sg.harm` field that the [forestsearch](#) result exposes.

This naming collision is a documented CRAN-stable API for v0.1.x and v0.2.x; it is expected to be resolved via a deprecation cycle in a future minor release.

See Also

[grf.subg.harm.survival](#) for time-to-event outcomes, [create_glm_row](#) for per-subgroup GLM estimation, [glm_effect_profile](#) for continuous biomarker profiles.

Examples

```
## Not run:
# Simulate count data with heterogeneous treatment effect
set.seed(123)
n <- 600
z1 <- rbinom(n, 1, 0.4) # binary subgroup factor
df_count <- data.frame(
  id      = seq_len(n),
  treat   = rbinom(n, 1, 0.5),
  v1      = as.factor(z1),
  v2      = as.factor(rbinom(n, 1, 0.5)),
  person_time = runif(n, 0.5, 2.0),
  age     = rnorm(n, 60, 10)
)
# Generate outcome: higher event rate for treated in z1 = 1 subgroup
log_rate <- -1.5 +
  0.8 * df_count$treat * (df_count$v1 == 1) -
  0.3 * df_count$treat * (df_count$v1 == 0) +
  log(df_count$person_time)
df_count$events <- rpois(n, lambda = exp(log_rate))

# Run GRF subgroup identification for count outcome (log transform, default)
grf_glm <- grf.subg.harm.glm(
  data      = df_count,
  confounders.name = c("v1", "v2", "age"),
  outcome.name   = "events",
  treat.name     = "treat",
```

```

    id.name           = "id",
    outcome_type      = "count",
    effect_measure     = "log_IRR",
    offset.name       = "person_time",
    overdispersion     = "quasi",
    grf_count_transform = "log",      # default: log(Y + 0.5)
    n.min             = 60,
    verbose            = TRUE
  )
  print(grf_glm$sg.harm.id)

# Alternative: identity transform (large counts, no zeros)
grf_glm_id <- grf.subg.harm.glm(
  data           = df_count,
  confounders.name = c("v1", "v2", "age"),
  outcome.name    = "events",
  treat.name      = "treat",
  id.name         = "id",
  outcome_type    = "count",
  effect_measure  = "log_IRR",
  offset.name     = "person_time",
  overdispersion  = "none",
  grf_count_transform = "identity", # raw counts passed to GRF
  n.min          = 60
)

## End(Not run)

```

grf.subg.harm.survival

GRF Subgroup Identification for Survival Data

Description

Identifies subgroups with differential treatment effect using generalized random forests (GRF) and policy trees. This function uses causal survival forests to identify heterogeneous treatment effects and policy trees to create interpretable subgroup definitions.

Usage

```

grf.subg.harm.survival(
  data,
  confounders.name,
  outcome.name,
  event.name,
  id.name,
  treat.name,
  frac.tau = 1,

```

```

n.min = 60,
dmin.grf = 0,
RCT = TRUE,
details = FALSE,
sg.criterion = "mDiff",
maxdepth = 2,
seedit = 8316951,
return_selected_cuts_only = FALSE,
tune_grf = FALSE,
grf_selection = c("tree", "frontier"),
frontier_rule = c("effMaxSG", "eff", "maxSG", "minSG", "effMinSG"),
effect_neighborhood = 0.1,
selection_rule = c("neighborhood", "pareto", "both")
)

```

Arguments

<code>data</code>	Data frame containing the analysis data.
<code>confounders.name</code>	Character vector of confounder variable names.
<code>outcome.name</code>	Character. Name of outcome variable (e.g., time-to-event).
<code>event.name</code>	Character. Name of event indicator variable (0/1).
<code>id.name</code>	Character. Name of ID variable.
<code>treat.name</code>	Character. Name of treatment group variable (0/1).
<code>frac.tau</code>	Numeric. Fraction of tau for GRF horizon (default: 1.0).
<code>n.min</code>	Integer. Minimum subgroup size (default: 60).
<code>dmin.grf</code>	Numeric. Minimum difference in subgroup mean (default: 0.0).
<code>RCT</code>	Logical. Is the data from a randomized controlled trial? (default: TRUE)
<code>details</code>	Logical. Print details during execution (default: FALSE).
<code>sg.criterion</code>	Character. Subgroup selection criterion ("mDiff" or "Nsg").
<code>maxdepth</code>	Integer. Maximum tree depth (1, 2, or 3; default: 2).
<code>seedit</code>	Integer. Random seed (default: 8316951).
<code>return_selected_cuts_only</code>	Logical. If TRUE, returns only cuts from the tree depth that identified the selected subgroup meeting <code>dmin.grf</code> . If FALSE (default), returns all cuts from all fitted trees (depths 1 through <code>maxdepth</code>).
<code>tune_grf</code>	Logical. If TRUE, enables cross-validated hyperparameter tuning via <code>tune.parameters = "all"</code> in the causal survival forest. Default FALSE.
<code>grf_selection</code>	Character, one of "tree" (default) or "frontier". "tree" uses the GRF policy tree (standard behavior). "frontier" is an experimental alternative that, on the same doubly-robust scores, enumerates single-covariate thresholds (and depth-2 covariate-pair conjunctions when <code>maxdepth</code> $\geq$ 2), takes the Pareto frontier of (harm-effect, size), and selects one subgroup under <code>frontier_rule</code> . The selected subgroup is always a single conjunction. In small benchmarks the policy tree matched or beat the frontier on harm recovery, so "frontier" is provided for comparison and exploration, not as a recommended default.

frontier_rule Character, one of "effMaxSG" (default), "eff", "maxSG", "minSG", or "effMinSG"; the rule applied to the frontier when `grf_selection = "frontier"`. Semantics match `dina_subgroup()`'s `sg_focus`: "eff" takes the maximum harm-effect candidate; "maxSG"/"minSG" the largest/smallest eligible candidate (effect $\geq$ `dmin.grf`); "effMaxSG"/"effMinSG" the largest/smallest candidate within `effect_neighborhood` (relative) of the max harm-effect.

effect_neighborhood Numeric in (0, 1); relative neighborhood for the "effMaxSG" / "effMinSG" rules. Default 0.10. Used only when `grf_selection = "frontier"`.

selection_rule Character, one of "neighborhood" (default), "pareto", "both"; the inclusion band for the eff*SG frontier rules, matching `forestsearch()` and `dina_subgroup()`. Inert for the other rules.

Details

The `return_selected_cuts_only` parameter controls which cuts are returned:

FALSE (default) Returns all cuts from all fitted trees (depths 1 to `maxdepth`). This provides the full set of candidate splits for downstream exploration and is the original behavior for backward compatibility.

TRUE Returns only cuts from the tree at the depth that identified the "winning" subgroup meeting the `dmin.grf` criterion. This is useful when you want a focused set of cuts associated with the selected subgroup, reducing noise from non-selected trees.

When `return_selected_cuts_only = TRUE` and no subgroup meets the criteria, `tree.cuts` will be empty (`character(0)`).

Value

A list with GRF results, including:

<code>data</code>	Original data with added treatment recommendation flags
<code>grf.gsub</code>	Selected subgroup information
<code>sg.harm.id</code>	Character vector of cut expressions defining the identified subgroup (length = depth of the selected policy tree), or NULL if no subgroup was found. Not a per-subject membership indicator; see the <i>Field naming collision</i> section below.
<code>tree.cuts</code>	Cut expressions - either all cuts from all trees (if <code>return_selected_cuts_only = FALSE</code>) or only cuts from the selected tree depth (if <code>return_selected_cuts_only = TRUE</code>)
<code>tree.names</code>	Unique variable names used in cuts
<code>tree</code>	Selected policy tree object
<code>tau.rmst</code>	Time horizon used for RMST
<code>harm.any</code>	All subgroups with positive treatment effect difference
<code>selected_depth</code>	Depth of the tree that identified the subgroup (when found)
<code>return_selected_cuts_only</code>	Logical indicating which cut extraction mode was used

Additional tree-specific cuts and objects (`tree1`, `tree2`, `tree3`) based on `maxdepth`

Field naming collision with forestsearch / subgroup.consistency

The field name `sg.harm.id` has **different semantics** on this object versus on the `grp.consistency` list returned by `subgroup.consistency` (and nested inside the `forestsearch` result):

Object	<code>sg.harm.id</code> contains	Length / type
<code>grf.subg.harm.survival()</code> result (this function)	character vector of cut expressions	character, length = depth of selection
<code>subgroup.consistency()</code> result	per-subject 0/1 membership indicator	integer, length <code>nrow(data)</code>

Practical consequence. On this object, pasting `paste(obj$sg.harm.id, collapse = " & ")` correctly renders the identified subgroup. On a `grp.consistency` list the same expression silently concatenates a long 0/1 indicator and produces output like `"0 & 0 & 1 & 0 & ..."`. When writing code that must handle both object types, dispatch on class first, or use the top-level `sg.harm` field that the `forestsearch` result exposes.

This naming collision is a documented CRAN-stable API for v0.1.x and v0.2.x; it is expected to be resolved via a deprecation cycle in a future minor release.

Examples

```
## Not run:
# Return all cuts (default behavior)
result_all <- grf.subg.harm.survival(
  data = trial_data,
  confounders.name = c("age", "biomarker", "region"),
  outcome.name = "tte",
  event.name = "event",
  id.name = "id",
  treat.name = "treat",
  dmin.grf = 0.1,
  maxdepth = 2
)
result_all$tree.cuts
# Returns cuts from both depth 1 and depth 2 trees

# Return only cuts from the selected tree
result_selected <- grf.subg.harm.survival(
  data = trial_data,
  confounders.name = c("age", "biomarker", "region"),
  outcome.name = "tte",
  event.name = "event",
  id.name = "id",
  treat.name = "treat",
  dmin.grf = 0.1,
  maxdepth = 2,
  return_selected_cuts_only = TRUE
)
result_selected$tree.cuts
# Returns cuts only from the depth that identified the winning subgroup

## End(Not run)
```

guohe_adaptive_r	<i>Cross-validated choice of the shrinkage tuning parameter r</i>
------------------	--

Description

Implements Algorithm 2 of Guo and He (2021), Section 2.5: a v -fold cross-validated choice of the shrinkage tuning parameter r used by `guohe_algorithm3()`. For each candidate r and each fold, the bias-reduced estimator is computed on the training folds and compared, on the held-out fold, against every candidate subgroup's effect estimate with its sampling variance subtracted; the r minimising the resulting CV objective is selected.

Usage

```
guohe_adaptive_r(
  data,
  outcome = c("survival", "binary", "continuous"),
  treatment,
  candidates,
  time = NULL,
  event = NULL,
  y = NULL,
  orient = -1,
  r_grid = c(0.03, 0.1, 0.2, 0.3, 0.4, 0.45),
  v = 5L,
  B = 200L,
  level = 0.05,
  seed = NULL,
  min_events = 5L,
  refit = TRUE,
  adjust_covariates = NULL,
  fast = NULL,
  parallel = FALSE
)
```

Arguments

<code>data</code>	A data frame.
<code>outcome</code>	One of "survival", "binary", "continuous".
<code>treatment</code>	Name of the 0/1 treatment column.
<code>candidates</code>	Character vector of names of 0/1 membership columns forming the enumerated family.
<code>time, event</code>	Survival only: names of the time and event columns.
<code>y</code>	Binary or continuous only: name of the outcome column.
<code>orient</code>	Either -1 or +1; sign orienting the effect of interest.
<code>r_grid</code>	Numeric vector of candidate values in (0, 0.5).

<code>v</code>	Number of cross-validation folds.
<code>B</code>	Number of bootstrap resamples.
<code>level</code>	One-sided level; the two-sided interval has coverage $1 - \text{level}$.
<code>seed</code>	Optional integer for reproducible resampling.
<code>min_events</code>	Minimum events (survival) or minimum per-arm count for a candidate to be estimable.
<code>refit</code>	Logical; if TRUE, also return a full-data fit at the selected <code>r</code> .
<code>adjust_covariates</code>	Optional character vector of covariate names for the within-subgroup model, matching the argument of the same name in <code>forestsearch()</code> 's <code>.estimate_or()</code> / <code>.make_lm_estimator()</code> . Default NULL is the unadjusted model. Supplying it DISABLES the closed-form fast path, because an adjusted coefficient is a partial regression coefficient, not a 2x2 log-odds-ratio or a difference of arm means. NOTE: <code>forestsearch()</code> has a SECOND adjustment axis, <code>ps_adjust_method</code> ("iptw" weighted fitting, "dr_gcomp" Bang-Robins G-computation). Those modes are not implemented in this comparator, and the closed forms would be invalid under them too – IPTW fits a weighted logistic, and DR reports a marginal log-OR from G-computed probabilities; neither equals the 2x2 statistic. If the comparator is extended to those modes, the <code>fast_ok</code> gate must be extended with them.
<code>fast</code>	Logical or NULL. Closed-form evaluation of the within-subgroup effect, available only for UNADJUSTED binary and continuous outcomes. These are EXACT, not approximations: with a single binary covariate the logistic model is saturated, so its MLE is the 2x2 log-odds-ratio, and the Gaussian coefficient is the difference of arm means. NULL (default) auto-enables it whenever it is valid. TRUE errors if the configuration makes it invalid (survival, or any <code>adjust_covariates</code>), rather than silently falling back.
<code>parallel</code>	Logical; if TRUE, distribute the bootstrap model fitting with <code>future::apply</code> . Requires a <code>future::plan()</code> to have been set by the caller. Results are identical to <code>parallel = FALSE</code> for the same seed, because resample indices are drawn serially before any distribution. Do NOT enable this inside an already-parallel outer loop.

Details

This is an independent implementation of the published method, validated by reproducing the simulation study of Section 5 of Guo and He (2021).

Value

An object of class "guohe_ar": the selected `r` (`r_hat`), the CV objective over the grid (`objective`), the per-candidate objective at the selected `r` (`per_candidate`), bookkeeping fields (`r_grid`, `v`, `B`, `n`, `n_candidates`, `fast`, `outcome`, `orient`, `level`, `adjust_covariates`), and, when `refit = TRUE`, the refitted full-data `guohe_algorithm3()` result (`fit`).

References

Guo, X. and He, X. (2021). Inference on Selected Subgroups in Clinical Trials. *Journal of the American Statistical Association*, **116**(535), 1498–1506. doi:10.1080/01621459.2020.1740096

Examples

```
set.seed(1)
n <- 160
df <- data.frame(
  treat = rbinom(n, 1, 0.5),
  y      = rbinom(n, 1, 0.3),
  g1     = rbinom(n, 1, 0.5),
  g2     = rbinom(n, 1, 0.5),
  g3     = rbinom(n, 1, 0.5)
)
cv <- guohe_adaptive_r(
  df, outcome = "binary", treatment = "treat",
  candidates = c("g1", "g2", "g3"), y = "y",
  orient = +1, r_grid = c(0.05, 0.25), v = 2, B = 25, seed = 2
)
cv$r_hat
```

guohe_algorithm3

De-biased inference for the best subgroup over a large enumerated family

Description

Finite realization of Algorithm 3 of Guo and He (2021): the candidate family is supplied as materialized 0/1 membership columns, the most extreme oriented effect is selected, and its selection optimism is removed by a pair (case) bootstrap with deterministic shrinkage offsets computed against the global maximum. Survival, binary, and continuous outcomes are supported.

Usage

```
guohe_algorithm3(
  data,
  outcome = c("survival", "binary", "continuous"),
  treatment,
  candidates,
  time = NULL,
  event = NULL,
  y = NULL,
  orient = -1,
  B = 2000L,
  r = 0.03,
  level = 0.05,
```

```

    seed = NULL,
    min_events = 5L,
    diagnostics = TRUE,
    parallel = FALSE,
    adjust_covariates = NULL,
    fast = NULL
  )

```

Arguments

<code>data</code>	A data frame.
<code>outcome</code>	One of "survival", "binary", "continuous".
<code>treatment</code>	Name of the 0/1 treatment column.
<code>candidates</code>	Character vector of names of 0/1 membership columns forming the enumerated family.
<code>time, event</code>	Survival only: names of the time and event columns.
<code>y</code>	Binary or continuous only: name of the outcome column.
<code>orient</code>	Either -1 or +1; sign orienting the effect of interest.
<code>B</code>	Number of bootstrap resamples.
<code>r</code>	Shrinkage tuning parameter, strictly between 0 and 0.5. Guo and He's Algorithm 2 gives a cross-validated choice; a fixed small value is used here.
<code>level</code>	One-sided level; the two-sided interval has coverage $1 - \text{level}$.
<code>seed</code>	Optional integer for reproducible resampling.
<code>min_events</code>	Minimum events (survival) or minimum per-arm count for a candidate to be estimable.
<code>diagnostics</code>	Logical; if TRUE, record per-resample estimability.
<code>parallel</code>	Logical; if TRUE, distribute the bootstrap model fitting with <code>future::apply</code> . Requires a <code>future::plan()</code> to have been set by the caller. Results are identical to <code>parallel = FALSE</code> for the same seed, because resample indices are drawn serially before any distribution. Do NOT enable this inside an already-parallel outer loop.
<code>adjust_covariates</code>	Optional character vector of covariate names for the within-subgroup model, matching the argument of the same name in <code>forestsearch()</code>'s <code>.estimate_or()</code> / <code>.make_lm_estimator()</code>. Default NULL is the unadjusted model. Supplying it DISABLES the closed-form fast path, because an adjusted coefficient is a partial regression coefficient, not a 2x2 log-odds-ratio or a difference of arm means. NOTE: <code>forestsearch()</code> has a SECOND adjustment axis, <code>ps_adjust_method</code> ("iptw" weighted fitting, "dr_gcomp" Bang-Robins G-computation). Those modes are not implemented in this comparator, and the closed forms would be invalid under them too – IPTW fits a weighted logistic, and DR reports a marginal log-OR from G-computed probabilities; neither equals the 2x2 statistic. If the comparator is extended to those modes, the <code>fast_ok</code> gate must be extended with them.

fast Logical or NULL. Closed-form evaluation of the within-subgroup effect, available only for UNADJUSTED binary and continuous outcomes. These are EXACT, not approximations: with a single binary covariate the logistic model is saturated, so its MLE is the 2x2 log-odds-ratio, and the Gaussian coefficient is the difference of arm means. NULL (default) auto-enables it whenever it is valid. TRUE errors if the configuration makes it invalid (survival, or any `adjust_covariates`), rather than silently falling back.

Details

Membership is computed once on the full data and carried through the bootstrap; cut points are NOT re-derived per resample. This is required by the method (the index set is held fixed) and is what makes the comparison against multiplier resampling like-for-like; regenerating the family per resample is the full-bootstrap object instead.

The oriented score is $\text{orient} * b$, with b the within-subgroup treatment coefficient on the natural scale. $\text{orient} = -1$ treats a protective ratio (HR or OR below 1) as the effect of interest, matching Guo and He's MONET1 convention; $\text{orient} = +1$ treats a harmful ratio as the effect of interest, which is the setting for a forest-search harm subgroup.

This is an independent implementation of the published method, validated by reproducing the simulation study of Section 5 of Guo and He (2021).

Value

An object of class "guohe_a3": a list with the selected candidate (`selected`), the naive and de-biased effects on the reporting scale (`naive`, `debiased`), the one-sided bound of Guo and He's Algorithm 3 (`bound_one_sided`, with its conservative side in `bound_side`), the two-sided quantile-inversion interval (`ci_two_sided`), the heuristic Wald interval (`ci_wald`), and bookkeeping fields (`n`, `B`, `B_requested`, `r`, `level`, `n_candidates`, `n_estimable`, `fast`, `lean_cox`, `adjust_covariates`, and, when `diagnostics = TRUE`, the per-candidate resample `drop_rate`).

References

Guo, X. and He, X. (2021). Inference on Selected Subgroups in Clinical Trials. *Journal of the American Statistical Association*, **116**(535), 1498–1506. doi:10.1080/01621459.2020.1740096

Examples

```
set.seed(1)
n <- 160
df <- data.frame(
  treat = rbinom(n, 1, 0.5),
  y      = rbinom(n, 1, 0.3),
  g1     = rbinom(n, 1, 0.5),
  g2     = rbinom(n, 1, 0.5),
  g3     = rbinom(n, 1, 0.5)
)
fit <- guohe_algorithm3(
  df, outcome = "binary", treatment = "treat",
  candidates = c("g1", "g2", "g3"), y = "y",
  orient = +1, B = 50, seed = 2
```

)
fit

hrCI_format	<i>Format Hazard Ratio and Confidence Interval</i>
-------------	--

Description

Formats a hazard ratio and confidence interval for display.

Usage

```
hrCI_format(hrest)
```

Arguments

hrest Numeric vector with HR, lower, and upper confidence limits.

Value

Character string formatted as \"HR (lower, upper)\".

Examples

```
hrCI_format(c(1.45, 0.98, 2.14))
```

inject_mrct_structure	<i>Build an aft_dgm_flex with synthetic regions and paper-style effect structure.</i>
-----------------------	---

Description

Build an aft_dgm_flex with synthetic regions and paper-style effect structure.

Usage

```
inject_mrct_structure(  
  seed_data,  
  outcome_var,  
  event_var,  
  treatment_var,  
  continuous_vars,  
  factor_vars,  
  x_pred,  
  spline_spec,
```



```

    region = list(),
    x3 = NULL,
    region_treat_loghr = NULL,
    n_super = 5000L,
    expand = c("copula", "bootstrap"),
    continuous_vars_cens = NULL,
    factor_vars_cens = NULL,
    cens_type = "weibull",
    seed = 8316951L,
    verbose = FALSE
  )

```

Arguments

seed_data	seed trial data.frame (one row per patient)
outcome_var, event_var, treatment_var	column names in seed_data
continuous_vars, factor_vars	covariates entering the AFT model (factor_vars must be 0/1 numeric or 2-level factors for clean expansion)
x_pred	region-associated effect modifier (X2), a continuous_var
spline_spec	list(knot, zeta, log_hrs) on the NATURAL scale of x_pred: true log-HR is piece-wise linear through (0, log_hrs[1]), (knot, log_hrs[2]), (zeta, log_hrs[3]). log_hrs constant => no heterogeneity (paper's beta1 = 0).
region	list(name = "region", prevalence = 0.2, or_pred = 10, x1_vars = character(0), or_x1 = 1, loghr = 0, scale = "rank") – loghr is the prognostic log-HR of Region (no TE modification; 0 for the paper's DGM; -log(5) reproduces the Feb-2026 deck's strongly prognostic region); scale = "rank" (ECDF) or "min-max" (the paper's [0, 1] scaling); optional band = list(cut, or) (and optional var, default x_pred) adds log(or) * 1(var <= cut) to the region logit, see make_region_model()
x3	NULL, or list(vars = c(...), cuts = list(...), loghr = ...) for a balanced effect modifier: cuts as in generate_aft_dgm_flex (use fixed values), loghr = log-HR change (treatment x subgroup).
region_treat_loghr	NULL, or a log-HR for a direct treat x Region interaction (the residual Region-U-TE pathway). Mutually exclusive with x3 (the package supports one interaction term).
n_super	size of the expanded super-population
expand	"copula" or "bootstrap"
continuous_vars_cens, factor_vars_cens, cens_type	passed through
seed	RNG seed
verbose	logical

Value

aft_dgm_flex with df_super = expanded population (+ z_<region>), and a \$mrct element documenting the injected structure.

interpret_estimation_table

Generate Narrative Interpretation of Estimation Properties

Description

Produces a templated text summary of the estimation properties table, automatically populating numerical results from the simulation output. Useful for reproducible vignettes where interpretation paragraphs should update when simulations are re-run.

Usage

```
interpret_estimation_table(
  results,
  dgm,
  analysis_method = "FSlg",
  n_sims = NULL,
  n_boots = 300,
  digits = 2,
  scenario = NULL,
  cat = TRUE,
  subgroup_notation = c("harm", "benefit"),
  trim_threshold = 1000,
  trim_fraction = 0.01
)
```

Arguments

results	Data frame of simulation results (same as for build_estimation_table).
dgm	DGM object with true parameter values.
analysis_method	Character. Which analysis method to summarise. Default: "FSlg".
n_sims	Integer. Total number of simulations (for detection rate). If NULL (default), derived from nrow(results) after filtering to the analysis method.
n_boots	Integer. Number of bootstraps (for narrative). Default: 300.
digits	Integer. Decimal places for reported values. Default: 2.
scenario	Character. One of "null" or "alt" (default). Controls the interpretive framing: • "null": emphasises false-positive rate and selection bias • "alt": emphasises power, bias relative to true effect If NULL, inferred from the DGM (hr_H_true == hr_Hc_true implies null).

cat	Logical. If TRUE (default), prints the paragraph via <code>cat()</code> . If FALSE, returns it invisibly as a character string (useful for programmatic insertion into Rmd via <code>results = "asis"</code>).
subgroup_notation	Character. "harm" (default) labels subgroups as H/Hc; "benefit" labels as G/Gc for benefit-search analyses (treatment switching).
trim_threshold	Numeric or NULL. When the raw mean of an estimate vector exceeds this absolute value, the narrative reports averages and SDs computed on the central $1 - 2 * \text{trim_fraction}$ of estimates and adds a closing note disclosing the trimming. Set NULL to disable. Default: 1000 (matches <code>build_estimation_table</code>).
trim_fraction	Numeric in $(0, 0.5)$. Fraction of estimates to trim from each tail when trimming triggers. Default: 0.01. Has no effect when <code>trim_threshold = NULL</code> .

Value

Invisibly returns the interpretation as a character string.

See Also

[build_estimation_table](#), [format_oc_results](#), [get_dgm_hr](#)

Examples

```
## Not run:
# In a vignette chunk with results = "asis":
interpret_estimation_table(results_null, dgm_null, scenario = "null")

# Capture for further processing:
txt <- interpret_estimation_table(results_alt, dgm_alt, cat = FALSE)

## End(Not run)
```

interpret_search_config

Interpret and Diagnose Search Configuration

Description

Generates a human-readable diagnostic of how ForestSearch will interpret the user's parameter combination. Prints via `message()` and returns an invisible list of diagnostics.

Usage

```
interpret_search_config(
  outcome_type,
  effect_measure,
  adverse_outcome,
  effect_threshold,
  consistency_threshold,
  use_lasso,
  use_grf,
  outcome.name,
  event.name,
  treat.name,
  offset.name = NULL,
  subgroup_method = "consistency",
  quiet = FALSE
)
```

Arguments

outcome_type	Character. One of "survival", "binary", "continuous", "count".
effect_measure	Character. "HR", "OR", "RR", "RD", "MD", "IRR", "IRD".
adverse_outcome	Logical. As resolved by forestsearch().
effect_threshold	Numeric. Resolved screening threshold (on log scale for ratio measures, identity for others).
consistency_threshold	Numeric. Resolved consistency threshold.
use_lasso	Logical.
use_grf	Logical. Whether GRF candidate-cut generation is on; see forestsearch .
outcome.name	Character. Name of outcome column.
event.name	Character. Name of event column.
treat.name	Character. Name of treatment column.
offset.name	Character or NULL.
subgroup_method	Character, the resolved subgroup_method. "dina" and "grf" return before the consistency stage and never read the consistency threshold, so the echoed value is annotated as unused on those paths. Display only; nothing is computed from it. Default "consistency".
quiet	Logical. If TRUE, suppress output.

Value

Invisible list with diagnostic fields.

lasso_selection	<i>LASSO selection for Cox model</i>
-----------------	--------------------------------------

Description

Performs LASSO variable selection using Cox regression.

Usage

```
lasso_selection(  
  df,  
  confounders.name,  
  outcome.name,  
  event.name,  
  seedit = 8316951,  
  outcome_type = "survival",  
  offset.name = NULL  
)
```

Arguments

df	Data frame.
confounders.name	Character vector of confounder names.
outcome.name	Character. Name of outcome variable.
event.name	Character. Name of event indicator variable.
seedit	Integer. Random seed.
outcome_type	Character. One of "survival" (default), "binary", "continuous", or "count".
offset.name	Character or NULL. Name of the follow-up time column for rate-based measures (IRR, IRD).

Value

List with selected, omitted variables, coefficients, lambda, and fits.

Examples

```
library(survival)  
df <- survival::gbsg  
df$grade3 <- as.integer(df$grade == "3")  
lasso_selection(df,  
  confounders.name = c("age", "meno", "size", "grade3", "nodes", "pgr", "er"),  
  outcome.name = "rfstime", event.name = "status")
```

latest_rds	<i>Find the Most Recent .rds File Matching a Pattern</i>
------------	--

Description

Scans output_dir for files matching the given prefix pattern and returns the path to the most recently modified one. Date-stamped filenames (e.g., results_20260402.rds) sort naturally, but this function uses filesystem modification time for robustness.

Usage

```
latest_rds(prefix, output_dir = "_data", verbose = TRUE)
```

Arguments

- prefix Character. Filename prefix to match (e.g., "selection_survival_frontier").
- output_dir Character. Directory to scan. Default: "_data".
- verbose Logical. Print the matched file. Default: TRUE.

Value

Full path to the most recent matching file, or NULL if no match is found.

Examples

```
## Not run:
path <- latest_rds("selection_survival_frontier")
res <- readRDS(path)

path <- latest_rds("calibration_binary_leff_grid")

## End(Not run)
```

load_leff_calibration	<i>Load L_eff Calibration Results</i>
-----------------------	---------------------------------------

Description

Reads a calibration bundle saved by save_leff_calibration(). Prints a summary and returns the full bundle.

Usage

```
load_leff_calibration(outcome_type, output_dir = NULL, verbose = TRUE)
```

Arguments

outcome_type Character. One of "binary", "survival", "count", "continuous".
output_dir Character. Directory containing .rds files. Default: "_data".
verbose Logical. Print summary on load. Default: TRUE.

Value

A list with components C, alpha, n_min, outcome_type, cal_data, dgm_description, timestamp, etc.

Examples

```
## Not run:  
cal <- load_leff_calibration("binary")  
C_prior <- cal$C  
alpha_prior <- cal$alpha  
  
## End(Not run)
```

make_effect_estimator *Make an Effect-Estimator Closure*

Description

Returns a closure function(data_slice) that estimates the within-subgroup treatment effect for the requested outcome type and effect measure. The closure is called repeatedly inside the splitting-consistency loop and the bootstrap worker, so it must be lightweight and handle small-sample edge cases gracefully.

Usage

```
make_effect_estimator(  
  outcome_type,  
  treat.name,  
  outcome.name,  
  event.name = NULL,  
  offset.name = NULL,  
  effect_measure = NULL,  
  adverse_outcome = TRUE,  
  robust_se = TRUE,  
  adjust_covariates = NULL,  
  ps_adjust_method = c("none", "iptw", "dr_gcomp"),  
  ...  
)
```

Arguments

outcome_type	Character. One of "survival", "binary", "continuous", or "count".
treat.name	Character. Name of the binary treatment column (0/1).
outcome.name	Character. Name of the outcome column.
event.name	Character or NULL. Name of the event indicator (survival only).
offset.name	Character or NULL. Name of the follow-up time column for rate-based measures (IRR, IRD).
effect_measure	Character or NULL. For binary: "OR" (default), "RR", "RD", "IRR", "IRD". For continuous: "MD" (default). For survival: "HR" (default). NULL uses the default for the outcome type, which for binary is "OR" – the same default forestsearch uses.
adverse_outcome	Logical. If TRUE (default), higher outcome values indicate harm and the effect is computed on the raw outcome. If FALSE, the outcome is beneficial (e.g., improvement) and the effect is computed on $1 - Y$ internally so that effect > 0 (or OR > 1) consistently indicates treatment harm. Only applies to binary outcomes; ignored for survival, continuous, and count.
robust_se	Logical. Use sandwich robust SE when available. Default TRUE.
adjust_covariates	Character vector or NULL. Additional covariate names to include in the outcome model formula.
ps_adjust_method	Character. Propensity score adjustment method. One of "none" (default), "iptw" (stabilized inverse probability of treatment weights), or "dr_gcomp" (Bang & Robins 2005 IPS covariate with G-computation).
...	Additional arguments (reserved for future extensions).

Value

A closure function(data_slice) returning a list with components: estimate, se, converged, n0, n1, measure, method_used.

Examples

```
## Not run:
# Binary outcome with odds ratio
fn <- make_effect_estimator("binary", "treat", "event",
  effect_measure = "OR")
result <- fn(my_data)
exp(result$estimate) # odds ratio

# Poisson rate with IPTW adjustment
fn_iptw <- make_effect_estimator("binary", "treat", "event",
  offset.name = "time", effect_measure = "IRR",
  ps_adjust_method = "iptw")

## End(Not run)
```

make_region_model	<i>Build the logistic region model and draw Region on a population.</i>
-------------------	---

Description

Build the logistic region model and draw Region on a population.

Usage

```
make_region_model(
  pop,
  x_pred = NULL,
  or_pred = 1,
  x1_vars = character(0),
  or_x1 = 1,
  prevalence = 0.2,
  scale_ref = NULL,
  scale = c("rank", "minmax"),
  band = NULL
)
```

Arguments

pop	data.frame holding x_pred and x1_vars (natural scale)
x_pred	name of the region-associated effect modifier (X2); NULL for no X2 term (regional imbalance only through x1_vars)
or_pred	odds ratio for s(x_pred) -> Region (OR = 1: none)
x1_vars	names of imbalanced-only covariates (X1); character(0) for none
or_x1	odds ratios for the X1 terms (recycled)
prevalence	target marginal P(Region = 1)
scale_ref	data.frame used to define the scaling (defaults to pop)
scale	"rank" (ECDF, default) or "minmax" (the paper's scaling)
band	NULL (default, no band term), or list(var, cut, or): adds log(or) * 1(df[[var]] <= cut) to the logit (var on its natural scale, not passed through s())

Value

list(alpha0, coefs, band, prob = function(df), draw = function(df, seed))

mrct_region_sims

*MRCT Regional Subgroup Simulation***Description**

Simulates multi-regional clinical trials and evaluates ForestSearch subgroup identification. Splits data by region into training and testing populations, identifies subgroups using ForestSearch on training data, and evaluates performance on the testing region.

Usage

```
mrct_region_sims(
  dgm,
  n_sims,
  n_sample = NULL,
  region_var = "z_regA",
  sg_focus = "minSG",
  maxk = 1,
  hr.threshold = 0.9,
  hr.consistency = 0.8,
  pconsistency.threshold = 0.9,
  confounders.name = NULL,
  conf_force = NULL,
  fs_args = list(),
  sim_args = list(rand_ratio = 1, draw_treatment = TRUE),
  analysis_time = 60,
  cens_adjust = 0,
  parallel_args = list(plan = "multisession", workers = NULL, show_message = TRUE),
  details = FALSE,
  verbose_n_sims = 2L,
  seed = NULL
)
```

Arguments

dgm	Data generating mechanism object from generate_aft_dgm_flex
n_sims	Integer. Number of simulations to run
n_sample	Integer. Sample size per simulation. If NULL (default), uses the entire super-population from dgm
region_var	Character. Name of the region indicator variable used to split data into training (region_var == 0) and testing (region_var == 1) populations. Default: "z_regA"
sg_focus	Character. Subgroup selection criterion passed to forestsearch : "minSG", "hr", or "maxSG". Default: "minSG"
maxk	Integer. Maximum number of factors in subgroup combinations (1 or 2). Default: 1

hr.threshold	Numeric. c_1 , the screening threshold on a candidate subgroup's own hazard ratio, passed to forestsearch . Default: 0.90
hr.consistency	Numeric. c_2 , an effect threshold – the minimum hazard ratio required in each half of a split. Not a proportion and not the consistency rate. Default: 0.80
pconsistency.threshold	Numeric. p^* , a proportion in $[0, 1]$ – the fraction of splits that must clear hr.consistency (c_2). Not an effect threshold. Default: 0.90
confounders.name	Character vector. Confounder variable names for ForestSearch. If NULL, automatically extracted from dgm
conf_force	Character vector. Forced cuts to consider in ForestSearch. Default: c("z_age <= 65", "z_bm <= 0", "z_bm <= 1", "z_bm <= 2", "z_bm <= 5")
fs_args	Named list. Additional arguments passed directly to forestsearch inside each simulation replicate. Use this to control parameters not exposed by mrct_region_sims (e.g., use_grf, use_lasso, cut_type, d0.min, d1.min, n.min, max_subgroups_search, use_twostage, twostage_args). Parameters already in the mrct_region_sims signature (hr.threshold, hr.consistency, pconsistency.threshold, sg_focus, maxk, confounders.name, conf_force) take precedence over values in fs_args. Default: list() (uses forestsearch defaults)
sim_args	Named list. Additional arguments passed to simulate_from_dgm inside each replicate (e.g., rand_ratio, draw_treatment). Parameters already in the mrct_region_sims signature (analysis_time, cens_adjust) take precedence. Default: list(rand_ratio = 1, draw_treatment = TRUE)
analysis_time	Numeric. Time of analysis for administrative censoring. Default: 60
cens_adjust	Numeric. Adjustment factor for censoring rate on log scale. Default: 0
parallel_args	List. Parallel processing configuration with components: • plan: "multisession", "multicore", "callr", or "sequential" • workers: Number of workers (NULL for auto-detect) • show_message: Logical for progress messages
details	Logical. Print detailed progress information. Default: FALSE
verbose_n_sims	Integer. When details = TRUE, print full ForestSearch diagnostics (including internal output) for only the first verbose_n_sims simulation replicates. Set to 0 to suppress per-sim output, or Inf to print all. Default: 2
seed	Integer. Base random seed for reproducibility. Default: NULL

Details

Simulation Process:

For each simulation:

1. Sample from super-population using [simulate_from_dgm](#)
2. Split by region_var into training and testing populations
3. Estimate HRs in ITT, training, and testing populations
4. Run [forestsearch](#) on training population
5. Apply identified subgroup to testing population

6. Calculate subgroup-specific estimates

Region Variable:

The `region_var` parameter is used ONLY for splitting data into training/testing populations. It does not imply any prognostic effect. To include prognostic confounder effects, specify them when creating the DGM using [create_dgm_for_mrct](#) or [generate_aft_dgm_flex](#).

Value

A `data.table` with simulation results containing:

sim Simulation index
n_itt ITT sample size
hr_itt ITT hazard ratio (stratified if strat variable present)
hr_ittX ITT hazard ratio stratified by region
n_train Training (non-region A) sample size
hr_train Training population hazard ratio
n_test Testing (region A) sample size
hr_test Testing population hazard ratio
any_found Indicator: 1 if subgroup identified, 0 otherwise
sg_found Character description of identified subgroup
n_sg Subgroup sample size
hr_sg Subgroup hazard ratio in testing population
POhr_sg Potential outcome hazard ratio in subgroup (testing)
prev_sg Subgroup prevalence (proportion of testing population)
n_sg_train Subgroup sample size in training population
hr_sg_train Subgroup hazard ratio in training population
POhr_sg_train Potential outcome hazard ratio in subgroup (training)
hr_sg_null Subgroup HR when found, NA otherwise

See Also

[forestsearch](#) for subgroup identification algorithm [generate_aft_dgm_flex](#) for DGM creation
[simulate_from_dgm](#) for data simulation [create_dgm_for_mrct](#) for MRCT-specific DGM wrapper
[summaryout_mrct](#) for summarizing simulation results

Examples

```
## Not run:
# Create DGM for alternative hypothesis
dgm_alt <- create_dgm_for_mrct(
  df_case = df_case,
  model_type = "alt",
  log_hrs = log(c(3, 1.25, 0.50)),
  verbose = TRUE
```

```

)

# Run simulations
results <- mrct_region_sims(
  dgm = dgm_alt,
  n_sims = 100,
  region_var = "z_regA",
  sg_focus = "minSG",
  parallel_args = list(plan = "multisession", workers = 4),
  details = TRUE
)

# Summarize results
cat("Subgroup identification rate:", mean(results$any_found) * 100, "%\n")

## End(Not run)

```

mrct_truth_by_region	<i>Truth by region on the super-population: prevalence, x_pred shift, causal AHR / CDE, and a large-sample Cox HR on UNCENSORED potential outcomes. Compare HR_cox_PO with the censored-trial HR from sim_region_metrics(): a gap between them isolates the prognostic-shift / follow-up-truncation mechanism (the Feb-2026 deck) from covariate shift (AHR itself differing by region).</i>
----------------------	--

Description

Truth by region on the super-population: prevalence, x_pred shift, causal AHR / CDE, and a large-sample Cox HR on UNCENSORED potential outcomes. Compare HR_cox_PO with the censored-trial HR from sim_region_metrics(): a gap between them isolates the prognostic-shift / follow-up-truncation mechanism (the Feb-2026 deck) from covariate shift (AHR itself differing by region).

Usage

```
mrct_truth_by_region(dgm, seed = 1L)
```

Arguments

dgm	An MRCT data-generating mechanism returned by inject_mrct_structure() . Uses its super-population df_super, the region indicator and effect-modifier names in mrct, and the AFT parameters mu and tau in model_params.
seed	Integer seed passed to set.seed() before drawing the common error term used to generate the uncensored potential outcomes. Default 1L.

Value

A data frame with one row per stratum ("ALL", "NON-REGION", "REGION") and columns stratum, n, prev, x_pred_mean, AHR, CDE and HR_cox_PO. Attribute "smd_x_pred" holds the standardised mean difference of the effect modifier between region and non-region.

mr_estimates_table	<i>Multiplier-resampling (MR) estimates table (bootstrap-table format)</i>
--------------------	--

Description

Renders the selected subgroup’s MR de-biased estimate ([forestsearch\(\)](#) with `mr_inference = TRUE`) in the same gt layout as the bias-corrected table produced by [summarize_bootstrap_results\(\)](#). The descriptive columns (sample size, medians/RMST or rates) and the naive effect do not depend on which de-biasing method was used, so they are inherited from a full-bootstrap (FB) result used as a template; the bias-corrected column is replaced with the MR values from `fs$mr_inference` – the de-biased estimate for the selected subgroup and, when MR was run with `include_complement = TRUE`, for the complement as well (otherwise the complement cell is marked, since MR de-biases the selected subgroup only).

Usage

```
mr_estimates_table(fs, boot_results, est.scale = "hr", digits = 2)
```

Arguments

- fs A [forestsearch\(\)](#) result with a non-NULL `mr_inference` element.
- boot_results A [forestsearch_bootstrap_dofuture\(\)](#) result used as the template for the descriptive columns, the naive effect, the complement row, and the subgroup footnote. Must contain `FSsg_tab` and `SG_CIs`.
- est.scale Character, "hr" or "1/hr"; passed to [format_bootstrap_table\(\)](#). Should match the scale of `boot_results`.
- digits Integer display precision (default 2), applied with the same reformatter [summarize_bootstrap_results\(\)](#) uses so the two tables align.

Details

The selected-subgroup row is matched by its bias-corrected string (`boot_results$SG_CIs$H_bc`) rather than by position, so the substitution is robust to row ordering. The MR de-biased confidence interval is a leading-order approximation of the full-bootstrap infinitesimal-jackknife interval – it comes from the IJ variance computed on the multiplier draws under the default `ci_method = "ij"`, and from the subgroup robust SE only under `ci_method = "wald"`. Either way the bias-corrected cell may differ slightly from the full bootstrap even when the naive effect matches exactly.

Value

A gt table.

See Also

[forestsearch\(\)](#) for the `mr_inference` switch and the vocabulary section; [fs_mr_inference\(\)](#) for the MR computation itself; [forestsearch_bootstrap_dofuture\(\)](#) for the full bootstrap that supplies the template; [fs_fdr_report\(\)](#) for harm-confirmation threshold sweeps; [summarize_bootstrap_results\(\)](#), [format_bootstrap_table\(\)](#)

n_pcnt	<i>Calculate n and percent</i>
--------	--------------------------------

Description

Returns count and percent for a vector relative to a denominator.

Usage

```
n_pcnt(x, denom)
```

Arguments

- x Vector of values.
- denom Denominator for percent calculation.

Value

Character string formatted as `"n (percent%)"`.

Examples

```
n_pcnt(1:30, 100)
```

pareto_frontier_table	<i>Format Pareto Frontier of Candidate Subgroups</i>
-----------------------	--

Description

Renders the post-hoc Pareto frontier on (effect, N) – both maximized – as a formatted **gt** table or returns it as a `data.table` for programmatic use. Works uniformly across survival (Cox PH) and GLM outcome types: the effect-column label and scale handling are derived from the `forestsearch` object’s `effect_measure`.

Usage

```
pareto_frontier_table(
  fs,
  format = c("gt", "data.table"),
  digits = 2L,
  digits_effect = NULL,
  digits_pcons = NULL,
  digits_ci = NULL,
  include_dominated = FALSE,
  include_factor_columns = TRUE,
  highlight_selected = TRUE,
  ci_table = NULL
)
```

Arguments

<code>fs</code>	A forestsearch object returned by forestsearch .
<code>format</code>	Character. Either "gt" (default) for a formatted gt table or "data.table" for a plain data.table with effect on the natural scale and an is_selected flag.
<code>digits</code>	Integer. Master decimal-places setting that drives all three column-specific defaults (digits_effect, digits_pcons, digits_ci) when those are left at their defaults. Default 2L. Pass digits = 3L for a tighter-precision view; pass per-column overrides below for asymmetric formatting (e.g. digits_effect = 3L with digits_ci = 2L).
<code>digits_effect</code>	Integer or NULL. Decimal places for the effect column. When NULL (default), inherits from digits. Pass an explicit integer to override.
<code>digits_pcons</code>	Integer or NULL. Decimal places for Pcons. When NULL (default), inherits from digits. Pass an explicit integer to override.
<code>digits_ci</code>	Integer or NULL. Decimal places used for the three CI columns (Naive, Split, FSBC) when ci_table is supplied. When NULL (default), inherits from digits. Pass an explicit integer to override.
<code>include_dominated</code>	Logical. If FALSE (default), the table shows only the Pareto-non-dominated subgroups (the frontier). If TRUE, the table shows all passing candidates – frontier members AND dominated candidates – with additional columns indicating frontier membership and band eligibility (when the selection rule uses one). This is the "rich post-consistency view" that complements the in-flight summary printed by forestsearch(show_candidate_summary = TRUE).
<code>include_factor_columns</code>	Logical. If TRUE (default), include the M.<factor> columns identifying each subgroup's defining cuts. If FALSE, show only summary metrics.
<code>highlight_selected</code>	Logical. If TRUE (default), the selected subgroup row is highlighted in the gt table. Ignored when format = "data.table".
<code>ci_table</code>	Optional data.table of frontier CIs from compute_frontier_cis . When supplied, three CI columns are merged into the table; when NULL (default), no CI

columns are shown. This function does not run any computation on `ci_table` – it simply displays the bounds already computed.

Details

The frontier is a diagnostic: it lists candidate subgroups that are *not dominated* on (effect, N) simultaneously. It is computed inside `sg_consistency_out` (see `compute_pareto_frontier`) and attached to the result object as `fs$grp.consistency$out_sg$pareto_frontier`. It is **not** used for selection – the selected subgroup is chosen by `sg_focus` and may or may not appear on the frontier.

Scale handling. For ratio measures stored on the log scale internally ("OR", "RR", "IRR"), the effect column is exponentiated for display. For "HR" (Cox PH, natural scale) and identity-scale measures ("RD", "IRD", "MD"), values pass through unchanged.

Selected-row identification. The selected subgroup is identified by matching the original-table row index `m` against the top row of `fs$grp.consistency$out_sg$result`. This is robust to sorting and outcome-type differences.

For `sg_focus = "hrMinSG"`, the selected subgroup may not appear on the frontier – that focus deliberately prefers small subgroups, which are typically N-dominated.

Optional confidence intervals. When `ci_table` is supplied (the `data.table` returned by `compute_frontier_cis`), three CI columns are added to the table. All three CIs are computed by `compute_frontier_cis`; this function only displays what is passed in. Pass the same `ci_table` to `plot_pareto_frontier` for consistent display across table and plot.

- Naive 95% CI – full-sample Wald CI from a Cox or GLM refit on each frontier member's data. Ignores subgroup-search selection; anti-conservative by construction.
- Split ~ 95% CI – subsample-derived approximation, computed from the empirical SD of averaged half-sample effects across the splits used by `compute_frontier_cis`.
- FSBC ~ 95% CI – bias-corrected interval following the bootstrap algorithm of *Leon2024fs* (eq 7) but treating the selected subgroup as fixed across half-jackknife replicates. The cell shows `est (lcl, ucl)` where `est` is the bias-corrected effect estimate $2\hat{\beta} - \hat{\beta}^{(h)}$.

See `compute_frontier_cis` for the algebra.

Value

Depending on format:

"gt" A `gt_tbl` object. Returns NULL invisibly (with a message) if the frontier is unavailable or empty.

"data.table" A `data.table` of the frontier with effect on the natural scale, an `is_selected` logical column, and effect column renamed to the `effect_measure` label (e.g., "HR", "OR"). Returns an empty `data.table` if the frontier is unavailable.

See Also

`compute_pareto_frontier`, `compute_frontier_cis`, `plot_pareto_frontier`, `sort_subgroups`, `forestsearch`.

Examples

```
## Not run:
# Survival example
data(gbsg, package = "survival")
fs <- forestsearch(gbsg, ...)

# Basic table (no CIs)
pareto_frontier_table(fs)
pareto_frontier_table(fs, format = "data.table")

# With CIs (compute once, display anywhere)
ci_tab <- compute_frontier_cis(fs, n_splits = 1000, seed = 1)
pareto_frontier_table(fs, ci_table = ci_tab)
plot_pareto_frontier(fs, ci_table = ci_tab) # same intervals

## End(Not run)
```

plot.forestsearch	<i>Plot ForestSearch Results</i>
-------------------	----------------------------------

Description

Dispatches to [plot_sg_results](#) for Kaplan-Meier curves, hazard-ratio forest plots, or combined panels.

Usage

```
## S3 method for class 'forestsearch'
plot(
  x,
  type = c("combined", "km", "forest", "summary"),
  outcome.name = "Y",
  event.name = "Event",
  treat.name = "Treat",
  ...
)
```

Arguments

x	A forestsearch object returned by forestsearch .
type	Character. Type of plot: "combined" KM curves + forest plot (default) "km" Kaplan-Meier survival curves only "forest" Hazard-ratio forest plot only "summary" Summary statistics panel
outcome.name	Character. Name of time-to-event column. Default: "Y".

event.name	Character. Name of event indicator column. Default: "Event".
treat.name	Character. Name of treatment column. Default: "Treat".
...	Additional arguments passed to plot_sg_results , such as <code>by.risk</code> , <code>conf.level</code> , <code>est.scale</code> , <code>sg0_name</code> , <code>sg1_name</code> , <code>treat_labels</code> , <code>colors</code> , <code>title</code> , <code>show_events</code> , <code>show_ci</code> , <code>show_logrank</code> , <code>show_hr</code> .

Value

Invisibly returns the plot result from [plot_sg_results](#).

See Also

[plot_sg_results](#) for full control over appearance, [plot_sg_weighted_km](#) for weighted KM curves, [plot_subgroup_results_forestplot](#) for publication-ready forest plots.

Examples

```
## Not run:
fs <- forestsearch(df.analysis = mydata, ...)

# Combined KM + forest plot (default)
plot(fs)

# KM curves only
plot(fs, type = "km")

# Forest plot only
plot(fs, type = "forest")

# With non-standard column names
plot(fs, type = "km",
      outcome.name = "os_months",
      event.name = "os_event",
      treat.name = "treatment")

# With custom labels
plot(fs, sg0_name = "High Risk", sg1_name = "Standard Risk",
      treat_labels = c("0" = "Placebo", "1" = "Active Drug"))

## End(Not run)
```

plot.fs_forestplot	<i>Plot Method for ForestSearch Forest Plot</i>
--------------------	---

Description

Plot Method for ForestSearch Forest Plot

Usage

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

Arguments

x An fs_forestplot object
 ... Additional arguments (ignored)

Value

The input x, invisibly.

Examples

```
## Not run:
fs <- forestsearch(gbsg,
  confounders.name = c("age", "meno", "size", "grade3", "nodes", "pgr", "er"),
  outcome.name = "rfstime", treat.name = "hormon", event.name = "status")
fp <- plot_subgroup_results_forestplot(list(fs.est = fs), gbsg,
  outcome.name = "rfstime", event.name = "status", treat.name = "hormon")
plot(fp)

## End(Not run)
```

plot.fs_sg_plot	<i>Plot Method for fs_sg_plot Objects</i>
-----------------	---

Description

Plot Method for fs_sg_plot Objects

Usage

```
## S3 method for class 'fs_sg_plot'
plot(x, which = 1, ...)
```

Arguments

x An fs_sg_plot object
 which Character or integer. Which plot to display. Default: 1 (first available)
 ... Additional arguments passed to plot functions

Value

x, invisibly. No plot is drawn: a message names the plot_sg_results() call that produces the requested plot. An error is raised when a character which does not name an available plot.

Examples

```
## Not run:
fs <- forestsearch(gbsg,
  confounders.name = c("age", "meno", "size", "grade3", "nodes", "pgr", "er"),
  outcome.name = "rfstime", treat.name = "hormon", event.name = "status")
p <- plot_subgroup(fs)
plot(p)

## End(Not run)
```

```
plot.subgroup_sims_summary
```

Forest plot of an extreme-subgroups summary

Description

Draws one of the vignettes' six forest panels from a `summary.subgroup_sims()` object via `gg_forest()`: the empirical-CI distribution of the HR point estimate or of the upper 95% bound UB(HR), for the single-variable panel, the combination + random- benchmark panel, or the high-risk panel (Pr(UB>=2) filter with the ITT row anchored first).

Usage

```
## S3 method for class 'subgroup_sims_summary'
plot(
  x,
  metric = c("hr", "ub"),
  panel = c("single", "combo", "highrisk"),
  hr_true = x$hr_true,
  base_size = 14,
  point_size = 2.2,
  line_size = 0.75,
  widths = c(3.5, 5, 1.1, 1.2, 1.2, 1.1),
  clip_marker = "arrow",
  xlim = NULL,
  ticks_at = NULL,
  xlab = NULL,
  footnote = NULL,
  ...
)
```

Arguments

<code>x</code>	A "subgroup_sims_summary" object.
<code>metric</code>	"hr" (HR point estimate) or "ub" (upper 95% bound).
<code>panel</code>	"single", "combo", or "highrisk".

hr_true	True hazard ratio for the reference/vertical line and footnotes; defaults to the value stored on x and must be non-NULL.
base_size, point_size, line_size	Passed to <code>gg_forest()</code> ; defaults match the vignettes' <code>fp_*</code> values.
widths	Column width vector passed to <code>gg_forest()</code> .
clip_marker	Passed to <code>gg_forest()</code> ; "arrow" marks whiskers clipped at the axis limits.
xlim, ticks_at, xlab, footnote	NULL (default) selects the vignette value for the chosen metric/panel.
...	Further arguments merged over the defaults and passed to <code>gg_forest()</code> .

Details

Defaults reproduce the committed vignette figures exactly: metric-specific axis limits, tick positions, reference/vertical line styling, the four annotation columns (N plus tail probabilities and the unconditional median), and the panel-specific footnotes. Any default can be overridden through the named arguments or `...`, which is merged over the assembled argument list before the `gg_forest()` call (so e.g. `ref_col = "black"` or `widths = ... in ... win`).

Summaries carrying effect metadata (the generic path of `summary.subgroup_sims()`, e.g. from `subgroup_glm()` fits) take a scale-aware branch instead: identity-scale measures (MD) plot with `xlog = FALSE` and the null line at 0, ratio-scale measures (OR) with `xlog = TRUE` and the null line at 1, and both delegate limits/ticks to `gg_forest()`'s data-driven defaults (the HR panel constants can hide binary medians and remain available via `xlim=/ticks_at=`); annotation columns and footnotes are built from the resolved labels and threshold strings (legacy literals preserved when a pair equals the legacy pair), and columns for disabled (NA) thresholds are dropped (the default widths shrink to match; an explicit widths is used as given). `hr_true` is interpreted on the fitter's estimate scale throughout. Legacy summaries produce byte-identical `gg_forest()` calls.

Value

The `gg_forest()` plot object.

See Also

`forest_height()`, `summary.subgroup_sims()`

Examples

```
## Not run:
S <- summary(sims_uniform, hr_true = 0.70)
plot(S, metric = "ub", panel = "combo")

## End(Not run)
```

plot_detection_curve *Plot Detection Probability Curve*

Description

Creates a visualization of the detection probability curve.

Usage

```
plot_detection_curve(  
  curve_data,  
  add_reference_lines = TRUE,  
  add_threshold_line = TRUE,  
  title = NULL,  
  ...  
)
```

Arguments

curve_data	A data.frame from generate_detection_curve or with columns: theta, probability.
add_reference_lines	Logical. Add horizontal reference lines at 0.05, 0.10, 0.80. Default: TRUE
add_threshold_line	Logical. Add vertical line at hr_threshold. Default: TRUE
title	Character. Plot title. Default: auto-generated
...	Additional arguments passed to plot()

Value

Invisibly returns the input data.

Examples

```
## Not run:  
curve_data <- generate_detection_curve(n_sg = 60, prop_cens = 0.2)  
plot_detection_curve(curve_data)  
  
## End(Not run)
```

plot_effect_distribution

Three-Panel Effect-Distribution Violin from Simulation Results

Description

Builds a three-panel violin plot (ITT / identified $\hat{H}$ / identified $\hat{H}^c$) of treatment-effect estimates across simulation replicates produced by `run_simulation_analysis()`. Handles both GLM endpoints (OR, RD, IRR) and survival (HR), optionally inverts effects to a benefit scale (`subgroup_notation = "benefit"`), and overlays a dotted reference line at the DGM-implied true effect in $\hat{H}$.

Usage

```
plot_effect_distribution(
  results,
  analysis_method = "FS",
  effect_measure = c("OR", "HR", "RD", "IRR", "MD"),
  subgroup_notation = c("harm", "benefit"),
  reference_effect = NA_real_,
  title = NULL,
  subtitle = NULL,
  panel_labels = NULL,
  drop_undetected = TRUE,
  trim_threshold = 1000,
  trim_fraction = 0.01,
  ylim = NULL
)
```

Arguments

<code>results</code>	data.frame returned by <code>run_simulation_analysis()</code> (optionally filtered). Must contain the columns <code>analysis</code> , <code>any.H</code> , <code>hr.it</code> , <code>hr.H.hat</code> , and <code>hr.Hc.hat</code> . The column names end in <code>.hr</code> regardless of the underlying effect measure (OR / HR / RD / IRR); this follows the package's established naming convention.
<code>analysis_method</code>	Character. Method label in the <code>analysis</code> column to plot (e.g. "FS", "GRF"). Default: "FS".
<code>effect_measure</code>	Character. One of "OR", "HR", "RD", "IRR", "MD". Used only for axis labelling and to decide the y-axis reference line (1 for ratio measures, 0 for risk / mean differences). "MD" (mean difference) is treated additively like "RD". Default: "OR".
<code>subgroup_notation</code>	Character. "harm" leaves effects on the original (switched) scale; "benefit" inverts multiplicative effects via $1/\hat{\theta}$ and flips the sign of additive effects so that larger values favour the benefit subgroup. Default: "harm".

reference_effect	Numeric or NA. Optional overlay showing the DGM-implied true effect in $\hat{H}$ (e.g. <code>1 / dgm\$hazard_ratios\$harm_subgroup</code> on a benefit scale). Drawn as a dotted darkgreen line. Default: NA.
title	Character. Plot title. Default: auto-constructed.
subtitle	Character or NULL. Plot subtitle. Default: NULL.
panel_labels	Named character vector with names "itt", "H", "Hc" and the display labels for each violin. Default: <code>c(itt = "ITT", H = "Identified H-hat", Hc = "Identified Hc-hat")</code> . When <code>subgroup_notation = "benefit"</code> , the default Hc label is adjusted to "Identified Hc-hat (benefit)".
drop_undetected	Logical. Restrict the $\hat{H}$ and $\hat{H}^c$ panels to replicates where a subgroup was identified (<code>any.H == 1</code>). Default: TRUE. The ITT panel always uses all replicates.
trim_threshold	Numeric or NULL. Trimming is applied only if any of the three groups has a raw mean exceeding this absolute value – i.e., extreme outliers that distort the violin or push annotated mean/SD into scientific notation. Default: 1000, matching build_estimation_table so the plot and the companion gt table auto-trim on the same condition. Set NULL to disable trimming entirely; set a smaller value (e.g., 100) to be more aggressive on plots where reasonable OR / HR / IRR estimates sit well below 100 and even moderate outliers visibly distort the violin axis.
trim_fraction	Numeric in (0, 0.5). Fraction of observations to trim from each tail of each group when trimming triggers. Default: 0.01 (1% of values per group is kept). Has no effect when <code>trim_threshold = NULL</code> .
ylim	Numeric vector of length 2 or NULL. Explicit y-axis limits as <code>c(lower, upper)</code> applied via coord_cartesian (data are <i>not</i> dropped, only the viewport is clipped). Useful when extreme outlier estimates stretch the axis beyond the interesting range — e.g., <code>ylim = c(0, 4)</code> for OR benefit-scale plots. When NULL (default), limits are determined automatically by ggplot2 .

Details

The MRCT analogue is [SGplot_estimates](#), which works on the four-panel `mrct_region_sims()` schema (ITT / training / testing / subgroup). This function targets the three-panel `run_simulation_analysis()` schema that does not involve a train/test split.

Value

A `ggplot2` object. Has `attr(p, "panel_data")` set to the long-format `data.table` used for plotting (for downstream summaries or diagnostic inspection). When trimming is active, also has `attr(p, "trim_info")` containing per-group diagnostics (`n_total`, `n_trimmed`, `n_flagged`, `raw_mean`, `raw_sd`, `trimmed_mean`, `trimmed_sd`, `lower_bound`, `upper_bound`).

See Also

[run_simulation_analysis](#) for the simulation pipeline, [SGplot_estimates](#) for the MRCT analogue, [build_estimation_table](#) for the tabular counterpart.

Examples

```
## Not run:
# Binary endpoint, benefit search, OR scale
true_or_benefit <- 1 / dgm_calibrated$hazard_ratios$harm_subgroup
p <- plot_effect_distribution(
  results_alt,
  analysis_method = "FS",
  effect_measure = "OR",
  subgroup_notation = "benefit",
  reference_effect = true_or_benefit,
  title = "ACTG175 Binary (HTE): OR Estimates Across Simulations",
  subtitle = sprintf("n = 1000, 50 sims | truth OR(Q) = %.2f",
    true_or_benefit)
)
print(p)

# Survival endpoint, harm search, HR scale (no inversion)
p_surv <- plot_effect_distribution(
  results_alt,
  effect_measure = "HR",
  subgroup_notation = "harm",
  reference_effect = dgm_alt$hazard_ratios$harm_subgroup
)

# Clip y-axis to focus on the interesting OR range
p_clipped <- plot_effect_distribution(
  results_alt,
  effect_measure = "OR",
  subgroup_notation = "benefit",
  reference_effect = true_or_benefit,
  ylim = c(0, 4)
)

## End(Not run)
```

plot_fs_identification_involvement

Covariate involvement across the sweep

Description

Reproduces the supplement's involvement figure: the share of detections whose rule names the anchor, the true partner, and the proxy, against the sweep dimension.

Usage

```
plot_fs_identification_involvement(x, roles = NULL, palette = NULL)
```

Arguments

x	An fs_identification object.
roles	Character vector. Which covariates to draw; defaults to the anchor/partner/proxy triple.
palette	Named character vector of colours, or NULL for the colour-blind-safe default used in the supplement.

Value

A ggplot object.

plot_fs_identification_structure

Composition of the identified subgroup by structure

Description

Reproduces the supplement's stacked-composition figure.

Usage

```
plot_fs_identification_structure(x, min_label = 0.03, palette = NULL)
```

Arguments

x	An fs_identification object.
min_label	Numeric. Shares below this are drawn without a label.
palette	Named character vector of fills, or NULL for the default.

Value

A ggplot object.

plot_fs_rule_pairs	<i>Selected-covariate-pair frequencies</i>
--------------------	--

Description

Bar-chart companion to `fs_rule_covariate_pairs()`, in the same idiom as the selected-covariate figure: horizontal bars, ordered by share, coloured by how the signature relates to the true region.

Usage

```
plot_fs_rule_pairs(
  x,
  true_covariates = NULL,
  top_n = 12L,
  detected_only = TRUE
)
```

Arguments

<code>x</code>	An <code>fs_identification</code> object, or anything <code>fs_rule_covariate_pairs()</code> accepts.
<code>true_covariates</code>	Character vector; taken from <code>x</code> when it is an <code>fs_identification</code> object.
<code>top_n</code>	Integer. Show only the most frequent <code>top_n</code> signatures; the remainder are pooled into a single "other" bar so the shares still sum to 1 and nothing is silently dropped.
<code>detected_only</code>	Passed through.

Value

A ggplot object.

plot_km_band_forestsearch	<i>Plot Kaplan-Meier Survival Difference Bands for ForestSearch Subgroups</i>
---------------------------	---

Description

Creates Kaplan-Meier survival difference band plots comparing the identified ForestSearch subgroup (`sg.harm`) and its complement against the ITT population. This function wraps `plotKM.band_subgroups()` from the `weightedsurv` package, automatically extracting subgroup definitions from ForestSearch results.

Usage

```
plot_km_band_forestsearch(
  df,
  fs.est = NULL,
  sg_cols = NULL,
  sg_labels = NULL,
  sg_colors = NULL,
  itt_color = "azure3",
  outcome.name = "tte",
  event.name = "event",
  treat.name = "treat",
  xlabel = "Time",
  ylabel = "Survival differences",
  yseq_length = 5,
  draws_band = 1000,
  tau_add = NULL,
  by_risk = 6,
  risk_cex = 0.75,
  risk_delta = 0.035,
  risk_pad = 0.015,
  ymax_pad = 0.11,
  show_legend = TRUE,
  legend_pos = "topleft",
  legend_cex = 0.75,
  ref_subgroups = NULL,
  verbose = FALSE
)
```

Arguments

<code>df</code>	Data frame. The analysis dataset containing all required variables including subgroup indicator columns.
<code>fs.est</code>	A forestsearch object containing the identified subgroup, or NULL if using pre-defined subgroup indicators.
<code>sg_cols</code>	Character vector. Names of columns in <code>df</code> containing subgroup indicators (0/1). These columns must already exist in <code>df</code> . If NULL and <code>fs.est</code> is provided, columns will be created automatically. Default: NULL
<code>sg_labels</code>	Character vector. Subsetting expressions for each subgroup, corresponding to <code>sg_cols</code> . These are passed to <code>plotKM.band_subgroups()</code> which evaluates them as R expressions (e.g., "age < 65", "er <= 0", "Qrecommend == 1"). Must be same length as <code>sg_cols</code> . Default: NULL (auto-generated as "colname == 1").
<code>sg_colors</code>	Character vector. Colors for each subgroup curve, corresponding to <code>sg_cols</code> . Must be same length as <code>sg_cols</code> . Default: NULL (uses default color palette).
<code>itt_color</code>	Character. Color for ITT population band. Default: "azure3".
<code>outcome.name</code>	Character. Name of time-to-event column. Default: "tte".
<code>event.name</code>	Character. Name of event indicator column. Default: "event".

treat.name	Character. Name of treatment column. Default: "treat".
xlabel	Character. X-axis label. Default: "Time".
ylabel	Character. Y-axis label. Default: "Survival differences".
yseq_length	Integer. Number of y-axis tick marks. Default: 5.
draws_band	Integer. Number of bootstrap draws for confidence band. Default: 1000.
tau_add	Numeric. Time horizon for the plot. If NULL, auto-calculated from data. Default: NULL.
by_risk	Numeric. Interval for risk table. Default: 6.
risk_cex	Numeric. Character expansion for risk table text. Default: 0.75.
risk_delta	Numeric. Vertical spacing for risk table. Default: 0.035.
risk_pad	Numeric. Padding for risk table. Default: 0.015.
ymax_pad	Numeric. Y-axis maximum padding. Default: 0.11.
show_legend	Logical. Whether to display the legend. Default: TRUE.
legend_pos	Character. Legend position (e.g., "topleft", "bottomright"). Default: "topleft".
legend_cex	Numeric. Character expansion for legend text. Default: 0.75.
ref_subgroups	Named list. Optional additional reference subgroups to include. Each element should be a list with: subset_expr Character. R expression to define subgroup (e.g., "age < 65") label Character. Display label (optional, defaults to subset_expr) color Character. Color for the curve The function automatically creates indicator columns from the expressions. Default: NULL.
verbose	Logical. Print diagnostic messages. Default: FALSE.

Details

This function simplifies the workflow of creating KM survival difference band plots for ForestSearch-identified subgroups. It can work in two modes:

Mode 1: With ForestSearch result (fs.est provided)

1. Extracts the subgroup definition from the ForestSearch result
2. Creates binary indicator columns (Qrecommend, Brecommend) in df
3. Generates appropriate labels from the subgroup definition
4. Calls plotKM.band_subgroups() with configured parameters

Mode 2: With pre-defined columns (sg_cols provided)

1. Uses existing indicator columns in df
2. Requires sg_labels and sg_colors to match sg_cols

The sg.harm subgroup (Qrecommend) represents patients with questionable treatment benefit (where treat.recommend == 0 in ForestSearch output). The complement (Brecommend) represents patients recommended for treatment.

Value

Invisibly returns a list containing:

- df** The modified data frame with subgroup indicators
- sg_cols** Character vector of subgroup column names used
- sg_labels** Character vector of subgroup labels used
- sg_colors** Character vector of colors used
- sg_harm_definition** The subgroup definition extracted from fs.est
- ref_subgroups** The reference subgroups list (if provided)

Subgroup Extraction

When fs.est is provided, the subgroup definition is extracted from:

- fs.est\$grp.consistency\$out_sg\$sg.harm_label - Human-readable labels
- fs.est\$sg.harm - Technical factor names (fallback)
- fs.est\$df.est\$treat.recommend - Subgroup membership indicator

Note

This function requires the **weightedSurv** package, which can be installed from CRAN: `install.packages("weightedSurv")`

See Also

[forestsearch](#) for running the subgroup analysis [plot_sg_weighted_km](#) for weighted KM plots
[plot_sg_results](#) for comprehensive subgroup visualization

Examples

```
## Not run:
# Mode 1: Using ForestSearch result (auto-creates Qrecommend/Brecommend)
# This will use labels "Qrecommend == 1" and "Brecommend == 1"
plot_km_band_forestsearch(
  df = df.analysis,
  fs.est = fs_result,
  outcome.name = "os_months",
  event.name = "os_event",
  treat.name = "treatment"
)

# Mode 1 with additional reference subgroups (auto-creates columns)
ref_sgs <- list(
  age_young = list(subset_expr = "age < 65", color = "brown"),
  age_old = list(subset_expr = "age >= 65", color = "grey")
)

plot_km_band_forestsearch(
  df = df.analysis,
  fs.est = fs_result,
```

```

    ref_subgroups = ref_sgs,
    outcome.name = "os_months",
    event.name = "os_event",
    treat.name = "treatment",
    tau_add = 60,
    by_risk = 6
)

# Mode 2: Using pre-defined subgroup columns with expression labels
# Note: sg_labels must be valid R expressions that evaluate against df
df.analysis$age_lt65 <- ifelse(df.analysis$age < 65, 1, 0)
df.analysis$age_ge65 <- ifelse(df.analysis$age >= 65, 1, 0)
df.analysis$Qrecommend <- ifelse(df.analysis$er <= 0, 1, 0)
df.analysis$Brecommend <- ifelse(df.analysis$er > 0, 1, 0)

plot_km_band_forestsearch(
  df = df.analysis,
  sg_cols = c("age_lt65", "age_ge65", "Qrecommend", "Brecommend"),
  sg_labels = c("age < 65", "age >= 65", "er <= 0", "er > 0"),
  sg_colors = c("brown", "grey", "blue", "red"),
  outcome.name = "os_months",
  event.name = "os_event",
  treat.name = "treatment",
  tau_add = 60,
  by_risk = 6
)

## End(Not run)

```

plot_pareto_combined	<i>Combined Pareto-Frontier Plot Across Configurations Sharing a Passing Set</i>
----------------------	--

Description

For two or more [forestsearch](#) fits whose consistency-passing candidate sets are **identical**, produces a single Pareto-frontier plot annotated with one S<i><combo_label> marker per configuration at each selected subgroup. If two configurations pick the same subgroup, the markers stack into a single multi-line label (e.g. "S1: hrMaxSG/pareto\nS2: hrMaxSG/both").

Usage

```

plot_pareto_combined(
  fs_list,
  combo_labels = NULL,
  ci_table_list = NULL,
  show_band = TRUE,
  xlim = NULL,

```



```

    tolerance = 1e-06,
    verbose = TRUE
  )

```

Arguments

<code>fs_list</code>	A list of forestsearch result objects. Length must be ≥ 2 .
<code>combo_labels</code>	Character vector of length <code>length(fs_list)</code> . Used in the <code>S<i>: <label></code> annotations. Default <code>NULL = auto</code> , e.g. <code>"config 1"</code> , <code>"config 2"</code> , ...
<code>ci_table_list</code>	Optional list of CI tables from compute_frontier_cis , parallel to <code>fs_list</code> . When provided and the tables match (within tolerance), the CI bars are drawn using the first table.
<code>show_band</code>	Logical. Draw the effect-band shading if applicable. Default <code>TRUE</code> .
<code>xlim</code>	Numeric vector of length 2 or <code>NULL</code> . Default <code>NULL = auto-range</code> .
<code>tolerance</code>	Numeric. Per-cell tolerance for the value equality check (hr, N, E, K). Default <code>1e-6</code> .
<code>verbose</code>	Logical. Emit a warning naming the specific equality-check failure mode when the sets don't match. Default <code>TRUE</code> .

Details

The combined plot is meaningful only when the passing sets match. When they don't, the function returns `NULL` with a warning; the side-by-side view (one panel per configuration) is the appropriate alternative.

Value

A ggplot object, or `NULL` (with a warning) if the passing sets don't satisfy the equality criterion.

Equality criterion

Two passing sets are considered equal when, after sorting by subgroup definition string:

- they have the same number of rows;
- the same set of subgroup definitions (concatenated `M.*` columns), compared as sets (order-independent);
- within each matched definition, the hr, N, E, and K values agree within tolerance.

`Pcons` is deliberately excluded from the value check. It can legitimately drift across rules because the preview sort (which depends on `selection_rule`) changes the queue order, which changes the random-split state each candidate consumes. Drift of up to ~ 0.10 between runs on the same subgroup is expected and does not indicate a real disagreement. Note also that the internal candidate id `m` can differ across configurations for the same reason; `m` is NOT used for equality.

See Also

[compare_selection_rules](#), [plot_pareto_frontier](#), [pareto_frontier_table](#).

Examples

```
## Not run:
out <- compare_selection_rules(
  df.analysis = actg_df,
  sg_focus    = c("hrMaxSG", "hrMaxSG"),
  selection_rule = c("pareto", "both"),
  ...
)

# Auto-attached by compare_selection_rules():
print(out$plot_combined)

# Or call directly:
p <- plot_pareto_combined(
  fs_list      = out$fs,
  combo_labels = out$combos$label,
  ci_table_list = out$ci_tab
)
print(p)

## End(Not run)
```

plot_pareto_frontier *Plot the Pareto Frontier of Candidate Subgroups*

Description

Produces a 2D scatter of candidate subgroups in (effect, N) space, with the Pareto frontier drawn as a step polyline and the selected subgroup highlighted. When `ci_table` is supplied (the return value of [compute_frontier_cis](#)), horizontal error bars for the split-derived 95\

Usage

```
plot_pareto_frontier(
  fs,
  ci_table = NULL,
  show_band = FALSE,
  effect_neighborhood = NULL,
  label_members = TRUE,
  point_size = 3,
  xlim = NULL,
  xlim_trim = NULL
)
```

Arguments

`fs` A forestsearch result object.

<code>ci_table</code>	Optional <code>data.table</code> from compute_frontier_cis . When supplied, horizontal error bars depict the split-derived 95% Default NULL (no error bars).
<code>show_band</code>	Logical. If TRUE and <code>sg_focus</code> is "hrMaxSG" or "hrMinSG", shade the effect-size neighborhood band on the right of the plot. Default FALSE.
<code>effect_neighborhood</code>	Numeric. Override of the band width; defaults to the value used in the original <code>forestsearch</code> call (or 0.10 if not recoverable).
<code>label_members</code>	Logical. If TRUE, label each frontier member with its short cut description. Default TRUE.
<code>point_size</code>	Numeric. Size of frontier-member points. Default 3.
<code>xlim</code>	Optional numeric vector of length 2 controlling the x-axis range, e.g. <code>c(0.5, 3)</code> . By default the axis auto-expands to include split-CI bars when supplied, which can make individual points look crowded. Pass an explicit <code>xlim</code> (or use <code>xlim_trim = TRUE</code>) to zoom in.
<code>xlim_trim</code>	Logical or NULL. If TRUE (and <code>xlim</code> is NULL), the x-axis is trimmed to the range of frontier point estimates with a small padding, ignoring CI bars. If NULL (default), auto-decides: TRUE when the effect measure is on a ratio (log) scale ("OR", "RR", "IRR"), where wide split-CI tails on the natural scale routinely dwarf the point-estimate range and force the axis to span orders of magnitude; FALSE otherwise. Pass TRUE or FALSE explicitly to override the auto behavior.

Details

The frontier polyline is drawn as a downward step from large-effect / small-N candidates to small-effect / large-N candidates. Points off the frontier (if any are present in the `fs$grp.consistency$out_sg$result` table beyond those on the frontier) are not currently plotted; this function only displays the frontier and its selected member.

For `sg_focus = "hrMinSG"`, the selected subgroup may not lie on the Pareto frontier (the focus deliberately prefers small subgroups, which are typically N-dominated). In that case the selected marker appears off the polyline.

Value

A `ggplot` object.

See Also

[pareto_frontier_table](#), [compute_frontier_cis](#), [frontier_member_flags](#).

Examples

```
## Not run:
p <- plot_pareto_frontier(fs)
print(p)

# With split-derived 95% CIs:
ci_tab <- compute_frontier_cis(fs, n_splits = 1000)
p2 <- plot_pareto_frontier(fs, ci_table = ci_tab)
```

```
print(p2)

## End(Not run)
```

plot_region_modifier	<i>True log-HR $\psi_0(x)$ along x_{pred} with region histograms (paper Fig. 3 analogue)</i>
----------------------	--

Description

True log-HR $\psi_0(x)$ along x_{pred} with region histograms (paper Fig. 3 analogue)

Usage

```
plot_region_modifier(dgm, file = NULL, main = NULL)
```

Arguments

dgm	An MRCT data-generating mechanism returned by <code>inject_mrct_structure()</code> . Uses <code>df_super</code> (the effect modifier, the region indicator and <code>loghr_po</code>) and <code>hazard_ratios\$AHR</code> for the reference line.
file	Optional character path. When supplied, the plot is written to this PNG file (900 x 800 px) and the device is closed on exit; when NULL (default), it is drawn on the current device.
main	Optional character title for the upper panel. When NULL (default), the title is "True causal log-HR along x_{pred} ".

Value

NULL, invisibly. Called for its side effect of drawing a two-panel plot.

plot_sg_distribution	<i>Plot Distribution of Identified Subgroups</i>
----------------------	--

Description

Bar chart of subgroups identified across simulations, filtered to those appearing in at least `min_pct` of the found simulations. Supports two column schemas (MRCT and `run_simulation_analysis`), top- K capping with a pooled "Other" bar, automatic threshold halving when no label clears the initial cut, and an `n_bars` attribute for adaptive figure-height computation.

Usage

```
plot_sg_distribution(
  results,
  min_pct = 5,
  title = "Distribution of Identified Subgroups",
  wrap_width = 25,
  any_col = "any_found",
  label_col = "sg_found",
  top_k = NULL,
  show_other = TRUE,
  min_floor_pct = 0.5,
  max_halvings = 0L,
  bar_label_inside = FALSE,
  placeholder_on_empty = FALSE
)
```

Arguments

results	data.frame or data.table with at least two columns: a per-replicate detection flag (0/1) and an identified-subgroup label string. Column names are configurable via any_col and label_col. The defaults ("any_found", "sg_found") match mrct_region_sims() ; pass any_col = "any.H", label_col = "sg.def" to plot run_simulation_analysis() output.
min_pct	Numeric. Minimum percentage threshold for display (0-100). Default: 5.
title	Character. Plot title.
wrap_width	Integer. Character width for wrapping long subgroup labels. Default: 25.
any_col	Character. Column name of the per-replicate detection flag. Default: "any_found".
label_col	Character. Column name of the identified-subgroup label string. Default: "sg_found".
top_k	Integer or NULL. Maximum number of bars to display; additional passing labels are pooled into a single "Other (<i>k</i> labels)" bar (see show_other). Prevents unreadable charts under diffuse null hypotheses where finds are fragmented across many singleton labels. Default: NULL (no cap, i.e. the original behaviour).
show_other	Logical. When top_k is set and extra labels are pooled, attach the "Other" bar. Default: TRUE. Set to FALSE to suppress the pooled bar entirely; its count and percent are still reported in the subtitle for honesty.
min_floor_pct	Numeric. Lower bound on automatic threshold halving. Default: 0.5.
max_halvings	Integer. Maximum number of halvings applied to min_pct when no label clears the initial threshold. Default: 0 (no halving, i.e. the original behaviour). Set to 2 to match the extended-diagnostic behaviour used in the benefit-search vignettes.
bar_label_inside	Logical. Render bar labels inside bars (hjust = 1.05) instead of outside (hjust = -0.1). Prevents small-count labels from clipping off the right edge. Default: FALSE (original outside-of-bar placement).
placeholder_on_empty	Logical. When no subgroups clear the threshold (even after halving), return a minimal "no data" placeholder plot with attr(p, "nBars") = 0L instead of

`invisible(NULL)`. Default: FALSE (the original behaviour). Set to TRUE to simplify adaptive- figure-height rendering in vignettes.

Value

A ggplot2 object, or `invisible(NULL)` when no subgroups are found and `placeholder_on_empty = FALSE`. When a plot is returned it carries two attributes:

`nBars` Integer. Number of bars in the plot (0 in the placeholder case). Use `sgdist_fig_height()` to compute an adaptive figure height for knitr.

`effective_min_pct` Numeric. The threshold in effect after any automatic halving.

See Also

[sgdist_fig_height](#) for computing an adaptive figure height from `nBars`, and [run_simulation_analysis](#) / [mrct_region_sims](#) for the upstream simulation engines.

Examples

```
## Not run:
# MRCT / mrct_region_sims() output (default column schema)
fs <- forestsearch(gbsg,
  confounders.name = c("age", "meno", "size", "grade3", "nodes", "pgr", "er"),
  outcome.name = "rfstime", treat.name = "hormon", event.name = "status")
plot_sg_distribution(fs$grp.consistency$result_new)

# run_simulation_analysis() output with top-K cap and auto-halving
fs_rows <- results_alt[results_alt$analysis == "FS", ]
p <- plot_sg_distribution(
  fs_rows,
  any_col          = "any.H",
  label_col        = "sg.def",
  top_k            = 15L,
  show_other       = FALSE,
  max_halvings     = 2L,
  bar_label_inside = TRUE,
  placeholder_on_empty = TRUE,
  title            = "HTE: Subgroups Identified (H-hat)"
)
fig_h <- sgdist_fig_height(attr(p, "nBars"))

## End(Not run)
```

plot_sg_glm_outcomes *Plot GLM Subgroup Outcomes*

Description

Creates a publication-ready visualization of identified subgroup outcomes for binary or continuous endpoints. This is the GLM counterpart to [plot_sg_weighted_km](#) for survival endpoints.

Usage

```

plot_sg_glm_outcomes(
  fs.est,
  fs_bc = NULL,
  outcome.name,
  treat.name,
  offset.name = NULL,
  effect_measure = NULL,
  outcome_type = NULL,
  adjust_covariates = NULL,
  sg0_name = "Questionable",
  sg1_name = "Recommend",
  E.name = "Treatment",
  C.name = "Control",
  bar_colors = c("steelblue", "coral"),
  show_effect = TRUE,
  show_bc = TRUE,
  show_itt = TRUE,
  conf.level = 0.95,
  title = NULL,
  subtitle = NULL,
  verbose = FALSE
)

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

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

```

Arguments

<code>fs.est</code>	A forestsearch object containing <code>df.est</code> with <code>treat.recommend</code> column.
<code>fs_bc</code>	Optional. Bootstrap results from forestsearch_bootstrap_dofuture for bias-corrected annotations.
<code>outcome.name</code>	Character. Name of outcome column.
<code>treat.name</code>	Character. Name of treatment column.
<code>offset.name</code>	Character or NULL. Name of follow-up time column for rate-based measures (IRR, IRD).
<code>effect_measure</code>	Character or NULL. Effect measure. Auto-detected from <code>fs.est\$effect_measure</code> if NULL.
<code>outcome_type</code>	Character or NULL. One of "binary", "continuous". Auto-detected from <code>fs.est\$outcome_type</code> if NULL.
<code>adjust_covariates</code>	Character vector or NULL. Baseline covariates to adjust the plotted effect model for, matching the adjustment used during search/estimation. When NULL, resolved from <code>fs.est\$args_call_all\$adjust_covariates</code> so the plot reflects

	the same (adjusted or unadjusted) scale as the fit. Columns must be present in <code>fs.est\$df.est</code> .
<code>sg0_name</code>	Character. Label for H subgroup (<code>treat.recommend == 0</code>). Default: "Questionable".
<code>sg1_name</code>	Character. Label for Hc subgroup (<code>treat.recommend == 1</code>). Default: "Recommend".
<code>E.name</code>	Character. Label for experimental/treatment arm. Default: "Treatment".
<code>C.name</code>	Character. Label for control arm. Default: "Control".
<code>bar_colors</code>	Character vector of length 2. Colors for control and treatment bars. Default: <code>c("steelblue", "coral")</code> .
<code>show_effect</code>	Logical. Annotate with effect estimate. Default TRUE.
<code>show_bc</code>	Logical. Annotate with bias-corrected estimate when <code>fs_bc</code> is provided. Default TRUE.
<code>show_itt</code>	Logical. Include an ITT (full population) panel. Default TRUE.
<code>conf.level</code>	Numeric. Confidence level for intervals. Default 0.95.
<code>title</code>	Character or NULL. Plot title.
<code>subtitle</code>	Character or NULL. Plot subtitle.
<code>verbose</code>	Logical. Print diagnostic messages. Default FALSE.
<code>x</code>	An <code>fs_binary_plot</code> object.
<code>...</code>	Additional arguments (unused).

Details

For binary outcomes, displays grouped bar charts of event rates by treatment arm within each subgroup. For continuous outcomes, displays point-and-errorbar charts of means ($\pm$ SE) by treatment arm. Both are annotated with the effect estimate and optional bias-corrected estimate from the bootstrap.

Value

A list of class "fs_binary_plot" containing:

plot A ggplot2 object.

data Data frame of per-arm summaries used for the plot.

effects Data frame of effect estimates per subgroup.

sg_definition Character. Subgroup definition label.

figure_note Character. Auto-generated figure note.

Examples

```
## Not run:
fs <- forestsearch(df, outcome_type = "binary", effect_measure = "OR",
  ...)
result <- plot_sg_glm_outcomes(fs, outcome.name = "status",
  treat.name = "hormon")
print(result)

## End(Not run)
```

plot_sg_results	<i>Plot ForestSearch Subgroup Results</i>
-----------------	---

Description

Creates comprehensive visualizations of subgroup results from ForestSearch, including Kaplan-Meier survival curves, hazard ratio comparisons, and summary statistics. This function is designed to work with the output from [forestsearch](#), specifically the `df.est` component.

Usage

```
plot_sg_results(  
  fs.est,  
  outcome.name = "Y",  
  event.name = "Event",  
  treat.name = "Treat",  
  plot_type = c("combined", "km", "forest", "summary"),  
  by.risk = NULL,  
  conf.level = 0.95,  
  est.scale = c("hr", "1/hr"),  
  sg0_name = "Questionable (H)",  
  sg1_name = "Recommend (H^c)",  
  treat_labels = c(`0` = "Control", `1` = "Treatment"),  
  colors = NULL,  
  title = NULL,  
  show_events = TRUE,  
  show_ci = TRUE,  
  show_logrank = TRUE,  
  show_hr = TRUE,  
  verbose = FALSE,  
  ...  
)
```

Arguments

<code>fs.est</code>	A forestsearch object or list containing at minimum: df.est Data frame with analysis data including <code>treat.recommend</code> sg.harm Character vector of subgroup-defining variable names grp.consistency Optional. Consistency results from <code>sg_consistency_out</code>
<code>outcome.name</code>	Character. Name of time-to-event outcome column. Default: "Y"
<code>event.name</code>	Character. Name of event indicator column (1=event, 0=censored). Default: "Event"
<code>treat.name</code>	Character. Name of treatment column (1=treatment, 0=control). Default: "Treat"
<code>plot_type</code>	Character. Type of plot to create. One of: "km" Kaplan-Meier survival curves

	"forest" Forest plot of hazard ratios
	"summary" Summary statistics panel
	"combined" All plots combined (default)
<code>by.risk</code>	Numeric. Risk interval for KM survival curves. Default: NULL (auto-calculated)
<code>conf.level</code>	Numeric. Confidence level for intervals. Default: 0.95
<code>est.scale</code>	Character. Effect scale: "hr" (hazard ratio) or "1/hr" (inverse). Default: "hr"
<code>sg0_name</code>	Character. Label for subgroup 0 (harm/questionable). Default: "Questionable (H)"
<code>sg1_name</code>	Character. Label for subgroup 1 (recommend/complement). Default: "Recommend (H^c)"
<code>treat_labels</code>	Named character vector. Labels for treatment arms. Default: <code>c("0" = "Control", "1" = "Treatment")</code>
<code>colors</code>	Named character vector. Colors for plot elements. Default: uses package defaults
<code>title</code>	Character. Main plot title. Default: auto-generated
<code>show_events</code>	Logical. Show event counts on KM curves. Default: TRUE
<code>show_ci</code>	Logical. Show confidence intervals. Default: TRUE
<code>show_logrank</code>	Logical. Show log-rank p-value. Default: TRUE
<code>show_hr</code>	Logical. Show hazard ratio annotation. Default: TRUE
<code>verbose</code>	Logical. Print diagnostic messages. Default: FALSE
<code>...</code>	Additional arguments passed to plotting functions.

Details

The function extracts subgroup membership from `fs.est$df.est$treat.recommend`:

- `treat.recommend == 0`: Harm/questionable subgroup (H)
- `treat.recommend == 1`: Recommend/complement subgroup (H^c)

For `est.scale = "1/hr"`, treatment labels and subgroup interpretation are reversed to maintain clinical interpretability.

Value

An object of class `fs_sg_plot` containing:

plots List of `ggplot2` or base R plot objects

summary Data frame of subgroup summary statistics

hr_estimates Data frame of hazard ratio estimates

call The matched call

Kaplan-Meier Plots

When `plot_type = "km"`, creates side-by-side survival curves for:

1. The identified subgroup (H) with treatment vs control
2. The complement subgroup (H^c) with treatment vs control

Forest Plot

When `plot_type = "forest"`, creates a forest plot showing hazard ratios with confidence intervals for: ITT population, H subgroup, and H^c complement.

See Also

[forestsearch](#) for running the subgroup analysis [sg_consistency_out](#) for consistency evaluation
[plot_subgroup_results_forestplot](#) for publication-ready forest plots

Examples

```
## Not run:
# Run ForestSearch analysis
fs <- forestsearch(
  df.analysis = my_data,
  outcome.name = "os_time",
  event.name = "os_event",
  treat.name = "treatment",
  confounders.name = c("age_cat", "stage", "biomarker")
)

# Create combined visualization
result <- plot_sg_results(
  fs.est = fs,
  outcome.name = "os_time",
  event.name = "os_event",
  treat.name = "treatment"
)

# View the Kaplan-Meier plots only
plot_sg_results(fs, plot_type = "km")

# Customize labels
plot_sg_results(
  fs,
  sg0_name = "High Risk",
  sg1_name = "Standard Risk",
  treat_labels = c("0" = "Placebo", "1" = "Active Drug")
)

## End(Not run)
```

plot_sg_weighted_km	<i>Plot Weighted Kaplan-Meier Curves for ForestSearch Subgroups</i>
---------------------	---

Description

Creates weighted Kaplan-Meier survival curves for the identified subgroups (H and H^c) using the `weightedsurv` package, matching the pattern used in `sg_consistency_out()`.

Usage

```
plot_sg_weighted_km(
  fs.est,
  fs_bc = NULL,
  outcome.name = "Y",
  event.name = "Event",
  treat.name = "Treat",
  by.risk = NULL,
  sg0_name = NULL,
  sg1_name = NULL,
  conf.int = TRUE,
  show.logrank = TRUE,
  show.cox = TRUE,
  show.cox.bc = TRUE,
  put.legend.lr = "topleft",
  ymax = 1.05,
  xmed.fraction = 0.65,
  hr_bc_position = "bottomright",
  hr_bc_cex = 0.725,
  title = NULL,
  verbose = FALSE
)
```

Arguments

<code>fs.est</code>	A forestsearch object containing <code>df.est</code> with <code>treat.recommend</code> column, or a data frame directly.
<code>fs_bc</code>	Optional. Bootstrap results from <code>forestsearch_bootstrap_dofuture()</code> containing bias-corrected HR estimates. If provided, bias-corrected HRs will be annotated on the plots.
<code>outcome.name</code>	Character. Name of time-to-event column. Default: "Y"
<code>event.name</code>	Character. Name of event indicator column. Default: "Event"
<code>treat.name</code>	Character. Name of treatment column. Default: "Treat"
<code>by.risk</code>	Numeric. Risk interval for plotting. Default: NULL (auto-calculated as $\max(\text{outcome})/12$)
<code>sg0_name</code>	Character. Label for H subgroup (<code>treat.recommend == 0</code>). Default: NULL (auto-extracted from forestsearch object as "H: {definition}" or "Questionable (H)" if not available)
<code>sg1_name</code>	Character. Label for Hc subgroup (<code>treat.recommend == 1</code>). Default: NULL (auto-generated as "Hc: NOT {definition}" or "Recommend (Hc)" if not available)
<code>conf.int</code>	Logical. Show confidence intervals. Default: TRUE
<code>show.logrank</code>	Logical. Show log-rank test. Default: TRUE
<code>show.cox</code>	Logical. Show unadjusted Cox HR from <code>weightedsurv</code> . Default: TRUE
<code>show.cox.bc</code>	Logical. Show bootstrap bias-corrected HR annotation (requires <code>fs_bc</code>). Default: TRUE
<code>put.legend.lr</code>	Character. Legend position. Default: "topleft"

ymax	Numeric. Max y-axis value. Default: 1.05
xmed.fraction	Numeric. Fraction for median lines. Default: 0.65
hr_bc_position	Character. Position for bias-corrected HR annotation. One of "bottomright", "bottomleft", "topright", "topleft". Default: "bottomright"
hr_bc_cex	Numeric. Character expansion factor for bias-corrected HR annotation text. Default: 0.725 (matches weightedsurv cox.cex default)
title	Character. Overall plot title. Default: NULL
verbose	Logical. Print diagnostic messages. Default: FALSE

Details

This function uses the exact same calling pattern as `plot_subgroup()` in the `ForestSearch` package. Column names are mapped internally to the standard names (Y, Event, Treat) expected by `weightedsurv`.

Subgroup definitions are automatically extracted from the `forestsearch` object if available:

- `fs$grp.consistency$out_sg$sg.harm_label` - Human-readable labels
- `fs$sg.harm` - Technical factor names (fallback)

HR display options controlled by `show.cox` and `show.cox.bc`:

- Both TRUE (default): Shows unadjusted HR from `weightedsurv` AND bias-corrected HR annotation
- `show.cox = TRUE`, `show.cox.bc = FALSE`: Shows only unadjusted HR
- `show.cox = FALSE`, `show.cox.bc = TRUE`: Shows only bias-corrected HR
- Both FALSE: Shows neither HR estimate

Value

Invisibly returns a list with subgroup data frames and counting data

Examples

```
## Not run:
# After running forestsearch - auto-extracts subgroup definition
plot_sg_weighted_km(fs.est = fs)

# With bootstrap bias-corrected estimates
plot_sg_weighted_km(fs.est = fs, fs_bc = fs_bootstrap)

# With custom column names
plot_sg_weighted_km(
  fs.est = fs,
  outcome.name = "time_months",
  event.name = "status",
  treat.name = "hormon"
)

## End(Not run)
```

plot_spline_treatment_effect
<i>Plot Spline Treatment Effect Function</i>

Description

Plot Spline Treatment Effect Function

Usage

```
plot_spline_treatment_effect(dgm_result, add_points = TRUE)
```

Arguments

dgm_result	Result object from generate_aft_dgm_flex with spline
add_points	Logical; add observed data points. Default TRUE

Value

Called for its side effect of producing a base-R plot. Returns NULL invisibly.

Examples

```
## Not run:
library(survival)
df <- survival::gbsg
dgm <- generate_aft_dgm_flex(df, outcome.name = "rfstime",
                             event.name = "status", treat.name = "hormon",
                             confounders.name = c("age", "meno", "nodes"))
plot_spline_treatment_effect(dgm)

## End(Not run)
```

plot_subgroup_effects	<i>Plot Subgroup Analysis Results</i>
-----------------------	---------------------------------------

Description

Creates diagnostic plots for subgroup treatment effects from df_super object

Usage

```
plot_subgroup_effects(
  df_super,
  z,
  hrz_crit = 0,
  log.hrs = NULL,
  ahr_empirical = NULL,
  plot_type = c("both", "profile", "ahr"),
  add_rug = TRUE,
  zpoints_by = 1,
  ...
)
```

Arguments

<code>df_super</code>	A data frame containing subgroup analysis results with columns: <code>loghr_po</code> (log hazard ratios), and optionally <code>theta_1</code> and <code>theta_0</code> (treatment-specific parameters)
<code>z</code>	Character string specifying the column name to use as the subgroup score (e.g., "z_age", "z_size", "subgroup"). Required.
<code>hrz_crit</code>	Critical hazard ratio threshold for defining optimal subgroup. Default is 1 (HR=1 on log scale is 0).
<code>log.hrs</code>	Optional vector of reference log hazard ratios to display as horizontal lines. Default is NULL.
<code>ahr_empirical</code>	Optional empirical average hazard ratio to display. If NULL, calculated from data. Default is NULL.
<code>plot_type</code>	Character string specifying plot type: "both" (default), "profile", or "ahr".
<code>add_rug</code>	Logical indicating whether to add rug plot of z values. Default is TRUE.
<code>zpoints_by</code>	Step size for z-axis grid when calculating AHR curves. Default is 1.
<code>...</code>	Additional graphical parameters passed to <code>plot()</code>

Details

The function creates up to two plots:

1. Treatment effect profile: Shows log hazard ratio as function of z
2. Average hazard ratio curve: Shows AHR for subgroups $z \geq \text{threshold}$

The "optimal" subgroup is defined as patients with $z \geq \text{cut.zero}$, where `cut.zero` is the minimum z value with favorable treatment effect ($\text{loghr} < \text{hrz_crit}$).

Value

A list containing:

<code>cut.zero</code>	The minimum z value where $\text{loghr_po} < \text{hrz_crit}$
<code>AHR_opt</code>	Average hazard ratio for optimal subgroup ($z \geq \text{cut.zero}$)

zpoints	Grid of z values used for AHR calculations
HR.zpoints	AHR for population with $z \geq$ zpoints
HRminus.zpoints	AHR for population with $z \leq$ zpoints
HR2.zpoints	Alternative AHR calculation for $z \geq$ zpoints
HRminus2.zpoints	Alternative AHR calculation for $z \leq$ zpoints

Examples

```
## Not run:
# Using z_age as the subgroup score
results <- plot_subgroup_effects(dgm_spline$df_super, z = "z_age", hrz_crit = 0)

# Using subgroup identifier
results <- plot_subgroup_effects(dgm_spline$df_super, z = "subgroup", hrz_crit = 0)

# With reference lines
results <- plot_subgroup_effects(dgm_spline$df_super, z = "z_size",
                                hrz_crit = 0,
                                log.hrs = c(-0.5, 0, 0.5))

# Only AHR plot
results <- plot_subgroup_effects(dgm_spline$df_super, z = "z_pgr",
                                plot_type = "ahr")

## End(Not run)
```

plot_subgroup_results_forestplot

Plot Subgroup Results Forest Plot

Description

Generates a comprehensive forest plot showing:

- ITT (Intent-to-Treat) population estimate
- Reference subgroups (e.g., by biomarker levels)
- Post-hoc identified subgroups with bias-corrected estimates
- Cross-validation agreement metrics as annotations

Usage

```
plot_subgroup_results_forestplot(
  fs_results,
  df_analysis,
  subgroup_list = NULL,
  outcome.name,
  event.name,
  treat.name,
  E.name = "Experimental",
  C.name = "Control",
  est.scale = "hr",
  xlog = TRUE,
  title_text = NULL,
  arrow_text = c("Favors Experimental", "Favors Control"),
  footnote_text = c("Eg 80% of training found SG: 70% of B (+) also B in CV testing"),
  xlim = c(0.25, 1.5),
  ticks_at = c(0.25, 0.7, 1, 1.5),
  show_cv_metrics = TRUE,
  cv_source = c("auto", "kfold", "oob", "both"),
  posthoc_colors = c("powderblue", "beige"),
  reference_colors = c("yellow", "powderblue"),
  ci_column_spaces = 20,
  conf.level = 0.95,
  theme = NULL,
  outcome_type = NULL,
  effect_measure = NULL,
  offset.name = NULL,
  adjust_covariates = NULL,
  extreme_ci_cap = 1.5,
  xlim_method = c("clinical", "data")
)
```

Arguments

- | | |
|----------------------------|--|
| <code>fs_results</code> | <p>List. A list containing ForestSearch analysis results with elements:</p> <ul style="list-style-type: none"> • <code>fs.est</code>: ForestSearch estimation object from forestsearch • <code>fs.bc</code>: Bootstrap bias-corrected results from forestsearch_bootstrap_dofuture • <code>fs.kfold</code>: K-fold cross-validation results from forestsearch_Kfold or forestsearch_tenfold (optional) • <code>fs.OOB</code>: Out-of-bag cross-validation results (optional, alternative to <code>fs.kfold</code>) |
| <code>df_analysis</code> | Data frame. The analysis dataset with outcome, event, and treatment variables. |
| <code>subgroup_list</code> | <p>List. Named list of subgroup definitions to include in the plot. Each element should be a list with:</p> <ul style="list-style-type: none"> • <code>subset_expr</code>: Character string for subsetting (e.g., "BM> 1") • <code>name</code>: Display name for the subgroup • <code>type</code>: Either "reference" for pre-specified or "posthoc" for identified |

<code>outcome.name</code>	Character. Name of the survival time variable.
<code>event.name</code>	Character. Name of the event indicator variable.
<code>treat.name</code>	Character. Name of the treatment variable.
<code>E.name</code>	Character. Label for experimental arm (default: "Experimental").
<code>C.name</code>	Character. Label for control arm (default: "Control").
<code>est.scale</code>	Character. Estimate scale: "hr" or "1/hr" (default: "hr").
<code>xlog</code>	Logical. If TRUE (default), the x-axis is plotted on a logarithmic scale. This is standard for hazard ratio forest plots where equal distances represent equal relative effects.
<code>title_text</code>	Character. Plot title (default: NULL).
<code>arrow_text</code>	Character vector of length 2. Arrow labels for forest plot (default: <code>c("Favors Experimental", "Favors Control")</code>).
<code>footnote_text</code>	Character vector. Footnote text for the plot explaining CV metrics (default provides CV interpretation guidance; set to NULL to omit).
<code>xlim</code>	Numeric vector of length 2. X-axis limits (default: <code>c(0.25, 1.5)</code>).
<code>ticks_at</code>	Numeric vector. X-axis tick positions (default: <code>c(0.25, 0.70, 1.0, 1.5)</code>).
<code>show_cv_metrics</code>	Logical. Whether to show cross-validation metrics (default: TRUE if <code>fs_kfold</code> or <code>fs_OOB</code> available).
<code>cv_source</code>	Character. Source for CV metrics: "auto" (default, uses both if available, otherwise whichever is present), "kfold" (use <code>fs_kfold</code> only), "oob" (use <code>fs_OOB</code> only), or "both" (explicitly use both <code>fs_kfold</code> and <code>fs_OOB</code> , with K-fold first then OOB).
<code>posthoc_colors</code>	Character vector. Colors for post-hoc subgroup rows (default: <code>c("powderblue", "beige")</code>).
<code>reference_colors</code>	Character vector. Colors for reference subgroup rows (default: <code>c("yellow", "powderblue")</code>).
<code>ci_column_spaces</code>	Integer. Number of spaces for the CI plot column width. More spaces = wider CI column (default: 20).
<code>conf.level</code>	Numeric. Confidence level for intervals (default: 0.95 for 95% CI). Used to calculate the z-multiplier as <code>qnorm(1 - (1 - conf.level)/2)</code> .
<code>theme</code>	An <code>fs_forest_theme</code> object from <code>create_forest_theme()</code> . Use this to control plot sizing (fonts, row height, CI appearance, CV annotation font size). Default: NULL (uses default theme).
<code>outcome_type</code>	Character or NULL. One of "survival" (default), "binary", or "continuous". If NULL, auto-detected from <code>fs_results\$fs.est\$outcome_type</code> .
<code>effect_measure</code>	Character or NULL. Effect measure for GLM outcomes (e.g., "OR", "RR", "IRR", "RD", "IRD", "MD"). Auto-detected from <code>fs_results\$fs.est\$effect_measure</code> if NULL.
<code>offset.name</code>	Character or NULL. Follow-up time column for rate-based measures (IRR, IRD).

<code>adjust_covariates</code>	Character vector or NULL. Baseline covariates to adjust the GLM-row effect models for, matching the adjustment used during search/estimation. When NULL, resolved from <code>fs_results\$fs.est\$args_call_all\$adjust_covariates</code> so the forest-plot CIs share the fit's scale. Applies to GLM rows only.
<code>extreme_ci_cap</code>	Numeric. Multiplier for outlier-resistant axis limits when <code>xlim_method = "data"</code> . If the most extreme CI bound exceeds twice the second-most extreme, the axis limit is capped at <code>extreme_ci_cap</code> times the second-most extreme bound. Default 1.5. Set to Inf to disable.
<code>xlim_method</code>	Character. Method for computing default axis limits for GLM measures when <code>xlim</code> is not user-supplied. "clinical" (default) uses fixed clinically meaningful ranges: HR/OR/RR/IRR = (0.25, 4.0), RD/IRD = (-0.50, 0.50), MD = data-driven. "data" computes limits from the actual estimates with outlier resistance controlled by <code>extreme_ci_cap</code> .

Details

Creates a publication-ready forest plot displaying identified subgroups with hazard ratios, bias-corrected estimates, and cross-validation metrics. This wrapper integrates ForestSearch results with the forestploter package.

ForestSearch Labeling Convention:

ForestSearch identifies subgroups based on hazard ratio thresholds:

- `sg.harm`: Contains the definition of the "harm" or "questionable" subgroup (H)
- `treat.recommend == 0`: Patient is IN the harm subgroup (H)
- `treat.recommend == 1`: Patient is in the COMPLEMENT subgroup (Hc, typically benefit)

For `est.scale = "hr"` (searching for harm):

- H (`treat.recommend=0`): Subgroup defined by `sg.harm` with elevated HR (harm/questionable)
- Hc (`treat.recommend=1`): Complement of `sg.harm` (potential benefit)

For `est.scale = "1/hr"` (searching for benefit):

- Roles are reversed: H becomes the benefit group

Value

A list containing:

plot The forestploter grob object (can be rendered with `plot()`)

data The data frame used for the forest plot

row_types Character vector of row types for styling reference

cv_metrics Cross-validation metrics text (if available)

See Also

[forestsearch](#) for running the subgroup analysis [forestsearch_bootstrap_dofuture](#) for bootstrap bias correction [forestsearch_Kfold](#) for cross-validation [create_forest_theme](#) for customizing plot appearance [render_forestplot](#) for rendering the plot

Examples

```

## Not run:
# Load ForestSearch results
load("fs_analysis_results.Rdata") # Contains fs.est, fs_bc, fs_kfold

# Define subgroups to display
subgroups <- list(
  bm_gt1 = list(
    subset_expr = "BM > 1",
    name = "BM > 1",
    type = "reference"
  ),
  bm_gt1_size_gt19 = list(
    subset_expr = "BM > 1 & tmrsize > 19",
    name = "BM > 1 & Tumor Size > 19",
    type = "posthoc"
  )
)

# Create the forest plot with default theme
result <- plot_subgroup_results_forestplot(
  fs_results = list(fs.est = fs.est, fs_bc = fs_bc, fs_kfold = fs_kfold),
  df_analysis = df_itt,
  subgroup_list = subgroups,
  outcome.name = "os_time",
  event.name = "os_event",
  treat.name = "combo",
  E.name = "Experimental+CT",
  C.name = "CT"
)

# Create with custom theme for larger plot
large_theme <- create_forest_theme(
  base_size = 14,
  row_padding = c(6, 4),
  cv_fontsize = 11
)

result <- plot_subgroup_results_forestplot(
  fs_results = list(fs.est = fs.est, fs_bc = fs_bc, fs_kfold = fs_kfold),
  df_analysis = df_itt,
  subgroup_list = subgroups,
  outcome.name = "os_time",
  event.name = "os_event",
  treat.name = "combo",
  theme = large_theme
)

# Display the plot
render_forestplot(result)

## End(Not run)

```

predict.dina	<i>Predicted DINA values for new covariate data.</i>
--------------	--

Description

Returns the estimated treatment-effect contrast $\hat{\tau}(x) = \hat{\beta}_0 + x^\top \hat{\beta}_{1:d}$, on the natural parameter scale of the fitted family (mean difference for Gaussian, log odds ratio for binomial, log mean ratio for Poisson, log hazard ratio for Cox).

Usage

```
## S3 method for class 'dina'
predict(object, newdata, ...)
```

Arguments

object	an object of class "dina".
newdata	a numeric matrix or data frame with the same number of columns as the training X and columns in the same order.
...	unused.

Value

numeric vector of predicted treatment-effect contrasts, one per row of newdata.

predict_fpr_corrected	<i>Predict Corrected Null FPR</i>
-----------------------	-----------------------------------

Description

Uses a calibrated L_{eff} model to predict the procedure-level FPR at arbitrary sample sizes and thresholds.

Usage

```
predict_fpr_corrected(
  calibration,
  theta,
  d_eff,
  c1,
  c2,
  N,
  effect_scale = "ratio"
)
```

Arguments

calibration	An "leff_calibration" object from <code>calibrate_L_eff</code> .
theta	Numeric. True treatment effect (1.0 for null).
d_eff	Numeric. Effective information in the subgroup.
c1	Numeric. Screening threshold.
c2	Numeric. Consistency threshold.
N	Integer. Total sample size.
effect_scale	Character. "ratio" or "difference".

Value

A list with components:

P1 Per-subgroup detection probability.

L_eff Effective number of candidates at this N.

fpr_corrected Corrected procedure-level FPR.

Examples

```
cal <- calibrate_L_eff(
  N = c(200, 500, 700, 1000),
  P1 = c(0.152, 0.109, 0.091, 0.071),
  sim_fpr = c(0.185, 0.220, 0.270, 0.280)
)

# Predict FPR at N = 800
d <- d_eff_binary(n_sg = round(0.30 * 800), p_event = 0.30)
predict_fpr_corrected(cal, theta = 1.0, d_eff = d,
  c1 = 1.25, c2 = 1.25, N = 800)
```

prepare_subgroup_data *Prepare subgroup data for analysis*

Description

Splits a data frame into two subgroups based on a flag and treatment scale.

Usage

```
prepare_subgroup_data(df, SG_flag, est.scale, treat.name)
```

Arguments

df	Data frame.
SG_flag	Character. Name of subgroup flag variable.
est.scale	Character. Effect scale ("hr" or "1/hr").
treat.name	Character. Name of treatment variable.

Value

List with subgroup data frames and treatment variable name.

Examples

```
df <- data.frame(
  treat = c(0, 1, 0, 1, 0),
  sg_flag = c(1, 1, 0, 0, 1)
)
result <- prepare_subgroup_data(df, SG_flag = "sg_flag",
                                est.scale = "hr", treat.name = "treat")
nrow(result$df_1)
```

print.cox_ahr_cde	<i>Print method for cox_ahr_cde objects</i>
-------------------	---

Description

Print method for cox_ahr_cde objects

Usage

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

Arguments

x	A cox_ahr_cde object from cox_ahr_cde_analysis .
...	Additional arguments (not used).

Value

Invisibly returns the input object.

Examples

```
## Not run:
library(survival)
df <- survival::gbag
df$grade3 <- as.integer(df$grade == "3")
res <- cox_ahr_cde_analysis(df,
  outcome.name = "rfstime", event.name = "status", treat.name = "hormon",
  confounders.name = c("age", "meno", "size", "grade3", "nodes", "pgr", "er"))
print(res)

## End(Not run)
```

```
print.dina_bagged      Print a bagged DINA fit.
```

Description

Extends print.dina() with bag count and per-bag cross-fitting info.

Usage

```
## S3 method for class 'dina_bagged'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

x	an object of class "dina_bagged".
digits	number of digits to show for coefficients.
...	unused.

Value

invisibly returns x.

```
print.dina_subgroup    Print a dina_subgroup result.
```

Description

Print a dina_subgroup result.

Usage

```
## S3 method for class 'dina_subgroup'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```


Arguments

x	a "dina_subgroup" object.
digits	number of digits for numeric summary.
...	unused.

Value

invisibly returns x.

```
print.dina_subgroup_bootstrap
```

Print a dina_subgroup_bootstrap result.

Description

Print a dina_subgroup_bootstrap result.

Usage

```
## S3 method for class 'dina_subgroup_bootstrap'  
print(x, digits = max(3L, getOption("digits") - 3L), max_select = 5L, ...)
```

Arguments

x	a "dina_subgroup_bootstrap" object.
digits	number of digits for numeric summary.
max_select	maximum number of rows of the selection-frequency table to display. Default 5L.
...	unused.

Value

invisibly returns x.

```
print.dina_subgroup_refit
```

Print a dina_subgroup_refit result.

Description

Print a dina_subgroup_refit result.

Usage

```
## S3 method for class 'dina_subgroup_refit'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

x	a "dina_subgroup_refit" object.
digits	number of digits for the numeric summary.
...	unused.

Value

invisibly returns x.

```
print.forestsearch
```

Print Method for forestsearch Objects

Description

Displays a concise summary of ForestSearch results including the identified subgroup definition, consistency metrics, algorithm details, and computation time.

Usage

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

Arguments

x	A forestsearch object returned by forestsearch .
...	Additional arguments (currently unused).

Value

Invisibly returns x.

Post-selection inference

When the fit carries multiplier-resampling results (`mr_inference = TRUE`, attached as `x$mr_inference`) and the field block ran (`ci_method = "field"`, the default), a further block reports: the one-sided 95% lower bound for the identified subgroup (the field); the one-sided 95% upper bound for its complement (field-s, the studentized complement field); the Bonferroni pair for a joint two-subgroup statement; the infinitesimal-jackknife (IJ) two-term two-sided interval, as a secondary summary; and the re-selection frequency $\hat{p}(\hat{H})$ as a descriptive diagnostic. Bounds are read by their location relative to a clinically meaningful effect size, not as tests at the null. With `ci_method = "ij"` only the two-sided interval is shown; without MR results the output is unchanged. The constructions and their simulation operating characteristics are described in León and Anderson (2026), [arXiv:2609.38361](#).

See Also

[summary.forestsearch](#) for detailed output, [plot.forestsearch](#) for visualization.

Examples

```
## Not run:
fs <- forestsearch(df.analysis = mydata, ...)
print(fs)
# or simply:
fs

## End(Not run)
```

```
print.forestsearch_comparison
```

Print Method for forestsearch_comparison

Description

Short summary of the comparison. Use `print(x$plot_grid)` for the side-by-side plot and `cat(x$console[[i]])` for each combo's diagnostic output.

Usage

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

Arguments

<code>x</code>	A <code>forestsearch_comparison</code> object.
<code>...</code>	Ignored.

Value

`x`, invisibly. Called for its side effect of printing the comparison summary.

```
print.fpr_calibration Print method for fpr_calibration objects
```

Description

Print method for fpr_calibration objects

Usage

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

Arguments

x	An fpr_calibration object.
...	Unused; present for S3 compatibility.

Value

The input x, invisibly.

```
print.fs_bootstrap Print Method for ForestSearch Bootstrap Results
```

Description

Prints a concise but informative summary of an fs_bootstrap object returned by [forestsearch_bootstrap_dofuture](#). Covers identification rate, agreement with the primary subgroup, top identified subgroups, and bias-corrected effect estimates.

Usage

```
## S3 method for class 'fs_bootstrap'
print(x, top_n = 3L, ...)
```

Arguments

x	An fs_bootstrap object.
top_n	Integer. Maximum number of top identified subgroups to print. Default 3.
...	Additional arguments (ignored; present for S3 consistency).

Details

The print method is deliberately richer than the corresponding `print.fs_kfold/print.fs_tenfold` methods because the bootstrap is the primary inferential machinery of the package, while the CV routines are complementary diagnostics. For the full per-factor breakdown, consistency distribution, size distribution, and GRF cut tabulation, call [summarize_bootstrap_subgroups](#).

Value

Invisibly returns x.

See Also

[forestsearch_bootstrap_dofuture](#) to produce the object. [summarize_bootstrap_subgroups](#) for full tabulations.

Examples

```
## Not run:
fs <- forestsearch(
  df.analysis = mydata,
  confounders.name = c("age", "sex", "biomarker"),
  outcome.name = "time", event.name = "status", treat.name = "treat"
)
fs_bc <- forestsearch_bootstrap_dofuture(fs, nb_boots = 500)
print(fs_bc)           # rich default summary
print(fs_bc, top_n = 5)

## End(Not run)
```

print.fs_cv_summary *Print Method for fs_cv_summary Objects*

Description

Print Method for fs_cv_summary Objects

Usage

```
## S3 method for class 'fs_cv_summary'
print(x, n = 5L, ...)
```

Arguments

x	An fs_cv_summary object from summarize_cv_results .
n	Integer. Maximum rows to print from each raw data frame (default: 5L).
...	Additional arguments (currently unused).

Value

Invisibly returns x.

print.fs_dgm_scale	<i>Print method for fs_dgm_scale objects</i>
--------------------	--

Description

Print method for fs_dgm_scale objects

Usage

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

Arguments

x	An object of class "fs_dgm_scale" from fs_dgm_scale .
...	Unused; present for S3 compatibility.

Value

The input x, invisibly.

print.fs_fdr_report	<i>Print method for fs_fdr_report objects</i>
---------------------	---

Description

Print method for fs_fdr_report objects

Usage

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

Arguments

x	An fs_fdr_report object.
...	Unused; present for S3 compatibility.

Value

The input x, invisibly.

print.fs_forestplot *Print Method for ForestSearch Forest Plot*

Description

Print Method for ForestSearch Forest Plot

Usage

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

Arguments

x	An fs_forestplot object
...	Additional arguments (ignored)

Value

The input x, invisibly.

Examples

```
## Not run:  
fs <- forestsearch(gbsg,  
  confounders.name = c("age", "meno", "size", "grade3", "nodes", "pgr", "er"),  
  outcome.name = "rfstime", treat.name = "hormon", event.name = "status")  
fp <- plot_subgroup_results_forestplot(list(fs.est = fs), gbsg,  
  outcome.name = "rfstime", event.name = "status", treat.name = "hormon")  
print(fp)  
  
## End(Not run)
```

print.fs_forest_theme *Print Method for ForestSearch Forest Theme*

Description

Print Method for ForestSearch Forest Theme

Usage

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

Arguments

x An fs_forest_theme object
... Additional arguments (ignored)

Value

The input x, invisibly.

Examples

```
theme <- create_forest_theme()
print(theme)
```

<code>print.fs_kfold</code>	<i>Print Method for K-Fold CV Results</i>
-----------------------------	---

Description

Print Method for K-Fold CV Results

Usage

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

Arguments

x An fs_kfold object
... Additional arguments (ignored)

Value

x, invisibly. Called for its side effect of printing the K-fold summary.

Examples

```
## Not run:
fs <- forestsearch(gbmg,
  confounders.name = c("age", "meno", "size", "grade3", "nodes", "pgr", "er"),
  outcome.name = "rfstime", treat.name = "hormon", event.name = "status")
oob <- forestsearch_Kfold(fs.est = fs)
print(oob)

## End(Not run)
```

print.fs_mr_oc	<i>Print method for fs_mr_oc objects</i>
----------------	--

Description

Print method for fs_mr_oc objects

Usage

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

Arguments

x	An object of class "fs_mr_oc" from fs_mr_oc_summary .
...	Unused; present for S3 compatibility.

Value

The input x, invisibly.

print.fs_sg_plot	<i>Print Method for fs_sg_plot Objects</i>
------------------	--

Description

Print Method for fs_sg_plot Objects

Usage

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

Arguments

x	An fs_sg_plot object
...	Additional arguments (unused)

Value

x, invisibly. Called for its side effect of printing the subgroup definition, summary statistics and hazard ratio estimates.

Examples

```
## Not run:
fs <- forestsearch(gbsg,
  confounders.name = c("age", "meno", "size", "grade3", "nodes", "pgr", "er"),
  outcome.name = "rfstime", treat.name = "hormon", event.name = "status")
p <- plot_subgroup(fs)
print(p)

## End(Not run)
```

print.fs_tenfold	<i>Print Method for Repeated K-Fold CV Results</i>
------------------	--

Description

Print Method for Repeated K-Fold CV Results

Usage

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

Arguments

x	An fs_tenfold object
...	Additional arguments (ignored)

Value

x, invisibly. Called for its side effect of printing the repeated K-fold summary.

Examples

```
## Not run:
fs <- forestsearch(gbsg,
  confounders.name = c("age", "meno", "size", "grade3", "nodes", "pgr", "er"),
  outcome.name = "rfstime", treat.name = "hormon", event.name = "status")
kfold <- forestsearch_tenfold(fs.est = fs, sims = 5)
print(kfold)

## End(Not run)
```

print.fs_weighted_km *Print Method for fs_weighted_km Objects*

Description

Print Method for fs_weighted_km Objects

Usage

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

Arguments

x	An fs_weighted_km object from plot_sg_weighted_km()
...	Additional arguments (unused)

Value

The input x, invisibly.

print.gbmsg_dgm *Print Method for gbmsg_dgm Objects*

Description

Print Method for gbmsg_dgm Objects

Usage

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

Arguments

x	A gbmsg_dgm object
...	Additional arguments (unused)

Value

x, invisibly. Called for its side effect of printing the data-generating mechanism summary.

Examples

```
## Not run:
dgm <- create_gbsg_dgm()
print(dgm)

## End(Not run)
```

print.glm_dgm	<i>Print method for glm_dgm objects</i>
---------------	---

Description

Print method for glm_dgm objects

Usage

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

Arguments

x	A glm_dgm object from generate_glm_dgm .
...	Unused; present for S3 compatibility.

Value

The input x, invisibly.

print.glm_effect_profile	<i>Print method for glm_effect_profile objects</i>
--------------------------	--

Description

Print method for glm_effect_profile objects

Usage

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

Arguments

x	A glm_effect_profile object.
...	Additional arguments (unused).

Value

Invisibly returns x.

print.guohe_a3	<i>Print method for guohe_a3 objects</i>
----------------	--

Description

Prints the family size, the selected candidate, the naive and de-biased effects, and the three interval objects returned by `guohe_algorithm3()`, labelled by their status (Guo-He bound / extension / heuristic).

Usage

```
## S3 method for class 'guohe_a3'
print(x, ...)
```

Arguments

x	An object of class "guohe_a3".
...	Ignored; present for method consistency.

Value

x, invisibly.

print.guohe_ar	<i>Print method for guohe_ar objects</i>
----------------	--

Description

Prints the cross-validation objective over the r grid, marks the selected value, and, when present, prints the refitted full-data `guohe_algorithm3()` result.

Usage

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

Arguments

x	An object of class "guohe_ar".
...	Ignored; present for method consistency.

Value

x, invisibly.

print.hte_test	<i>Print method for hte_test objects</i>
----------------	--

Description

Print method for hte_test objects

Usage

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

Arguments

x	An hte_test object.
...	Unused; present for S3 compatibility.

Value

The input x, invisibly.

print.leff_calibration	<i>Print method for leff_calibration objects</i>
------------------------	--

Description

Print method for leff_calibration objects

Usage

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

Arguments

x	A leff_calibration object.
...	Unused; present for S3 compatibility.

Value

The input x, invisibly.

```
print.multi_survreg_comparison
```

Print method for survreg_comparison objects

Description

Print method for survreg_comparison objects

Usage

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

Arguments

x	A survreg_comparison object
...	Additional arguments (not used)

Value

Invisibly returns the input object

Examples

```
## Not run:
library(survival)
df <- survival::gbsg
res <- compare_multiple_survreg(df, outcome.name = "rfstime",
                               event.name = "status")
print(res)

## End(Not run)
```

```
print_grf_details
```

Print detailed output for debugging

Description

Displays detailed information about the GRF analysis

Usage

```
print_grf_details(config, values, best_subgroup, sg_harm_id, tree_cuts = NULL)
```

Arguments

config	List. GRF configuration
values	Data frame. Node metrics
best_subgroup	Data frame row. Selected subgroup (or NULL)
sg_harm_id	Character. Subgroup definition (or NULL)
tree_cuts	List. Cut information

Value

Called for its side effect of printing to the console. The value is returned invisibly and is incidental: tree_cuts\$all when a subgroup, its definition and tree_cuts are all supplied, otherwise NULL.

Examples

```
## Not run:
# print_grf_details() is called internally by grf.subg.harm.survival().
# See grf.subg.harm.survival() for the standard entry point.

## End(Not run)
```

process_conf_force_expr

Process forced cut expression for a variable

Description

Evaluates a cut expression (e.g., "age <= mean(age)") and returns the expression with the value.

Usage

```
process_conf_force_expr(expr, df)
```

Arguments

expr	Character string of the cut expression.
df	Data frame.

Value

Character string with evaluated value.

Examples

```
df <- data.frame(age = c(40, 55, 70, 35), size = c(20, 30, 25, 15))
process_conf_force_expr("age <= mean(age)", df)
process_conf_force_expr("size <= median(size)", df)
```

quick_km_band_plot	<i>Quick Plot KM Bands from ForestSearch</i>
--------------------	--

Description

Convenience wrapper with sensible defaults for quick visualization.

Usage

```
quick_km_band_plot(df, fs.est, outcome.name, event.name, treat.name, ...)
```

Arguments

df	Data frame with analysis data.
fs.est	ForestSearch result object.
outcome.name	Character. Time-to-event column name.
event.name	Character. Event indicator column name.
treat.name	Character. Treatment column name.
...	Additional arguments passed to <code>plot_km_band_forestsearch()</code> , such as <code>ref_subgroups</code> , <code>tau_add</code> , <code>by_risk</code> , etc.

Value

Invisibly returns the plot result.

Examples

```
## Not run:
# Quick plot with minimal configuration
quick_km_band_plot(df_analysis, fs_result, "os_months", "os_event", "treat")

# With reference subgroups
ref_sgs <- list(
  age_young = list(subset_expr = "age < 65", color = "brown"),
  age_old = list(subset_expr = "age >= 65", color = "grey")
)
quick_km_band_plot(df_analysis, fs_result, "os_months", "os_event", "treat",
  ref_subgroups = ref_sgs, tau_add = 60)

## End(Not run)
```

render_forestplot	<i>Render ForestSearch Forest Plot</i>
-------------------	--

Description

Renders a forest plot from `plot_subgroup_results_forestplot()`.

Usage

```
render_forestplot(x, newpage = TRUE)
```

Arguments

<code>x</code>	An <code>fs_forestplot</code> object from <code>plot_subgroup_results_forestplot()</code> .
<code>newpage</code>	Logical. Call <code>grid.newpage()</code> before drawing. Default: <code>TRUE</code> .

Details

To control plot sizing, create a custom theme using `create_forest_theme()` and pass it to `plot_subgroup_results_forestplot()`.

```
my_theme <- create_forest_theme(base_size = 14, row_padding = c(6, 4))
```

```
result <- plot_subgroup_results_forestplot(..., theme = my_theme)
```

```
render_forestplot(result)
```

Value

Invisibly returns the grob object.

Examples

```
## Not run:
# For larger plot, use theme parameter in plot_subgroup_results_forestplot:
large_theme <- create_forest_theme(
  base_size = 14,
  row_padding = c(6, 4),
  cv_fontsize = 11
)

result <- plot_subgroup_results_forestplot(
  fs_results = list(fs.est = fs, fs_bc = fs_bc),
  df_analysis = df.analysis,
  outcome.name = "time",
  event.name = "status",
  treat.name = "treatment",
  theme = large_theme
)

render_forestplot(result)
```

```
## End(Not run)
```

```
render_reference_table
```

Render Reference Simulation Table as gt

Description

Converts a data frame of pre-computed reference simulation results (e.g., digitized from a published LaTeX table) into a styled gt table. This is useful for displaying published benchmark results alongside new simulation output within vignettes or reports.

Usage

```
render_reference_table(
  ref_df,
  title = "Reference Simulation Results",
  subtitle = NULL,
  bold_threshold = 0.05,
  subgroup_notation = c("harm", "benefit")
)
```

Arguments

ref_df	data.frame. Must contain a Metric column and one column per analysis method, with a Scenario column for row grouping.
title	Character. Table title.
subtitle	Character. Table subtitle. Default: NULL.
bold_threshold	Numeric. Values in any(H) rows exceeding this threshold are shown in bold. Set NULL to disable. Default: 0.05.
subgroup_notation	Character. "harm" (default) labels subgroups as H/Hc; "benefit" labels as G/Gc for benefit-search analyses (treatment switching).

Value

A gt table object.

Examples

```
## Not run:
ref <- data.frame(
  Scenario = "M1 Null: N=700",
  Metric   = "any(H)",
  FS       = 0.02,
  FS1g     = 0.03,
```

```

    GRF      = 0.25
  )
  render_reference_table(ref, title = "Reference Results")

## End(Not run)

```

reset_workers

Reset a Parallel Worker Pool

Description

Tears down the current **future** plan, forces garbage collection, optionally pauses, and spins up a fresh worker pool. Used to release worker-side memory between simulation phases (e.g. between alternative and null Monte Carlo runs) so that accumulated R-session state in long-running workers does not bloat memory or cause sporadic failures.

Usage

```

reset_workers(
  workers = NULL,
  strategy = "multisession",
  sleep = 1,
  gc = TRUE,
  quiet = FALSE
)

```

Arguments

workers	Integer. Number of workers for the new pool. Default NULL leaves the current plan's worker count unchanged (useful if a reset without resize is wanted; this requires future to be installed).
strategy	Character. The future strategy to reinstate. Default "multisession". Other reasonable choices are "multicore" (Linux/macOS only) and "sequential" (to disable parallelism entirely).
sleep	Numeric. Seconds to pause between teardown and setup. Default 1. Set to 0 to skip.
gc	Logical. Whether to force garbage collection between teardown and setup. Default TRUE.
quiet	Logical. Suppress informational message() output. Default FALSE.

Details

The worker reset sequence is:

1. `future::plan("sequential")` – dismantles the existing worker pool; R processes terminate and their memory is returned to the OS.

2. `gc(verbose = FALSE)` – reclaims main-process memory holding references to now-defunct futures.
3. `Sys.sleep(sleep)` – brief pause to let the OS reclaim sockets / pipes before the new pool is requested. A small non-zero sleep (the default 1) materially reduces multisession setup failures observed on busy Linux servers; set to 0 to skip.
4. `future::plan(strategy, workers = workers)` – starts a fresh pool.

This function is a no-op if future is not installed or if the requested strategy cannot be initialised; a warning is emitted and the previous plan is left in place.

Value

Invisibly, a list with the previous and current plan class names: `list(previous = <chr>, current = <chr>)`.

See Also

[collect_results](#) for error-tolerant collection of foreach output.

Examples

```
if (requireNamespace("future", quietly = TRUE)) {
  future::plan("multisession", workers = 2L)
  # ... run alternative simulations ...
  reset_workers(workers = 2L)
  # ... run null simulations ...
  future::plan("sequential")
}
```

rmst_calculation

RMST calculation for subgroup

Description

Calculates restricted mean survival time (RMST) for a subgroup.

Usage

```
rmst_calculation(
  df,
  tte.name = "tte",
  event.name = "event",
  treat.name = "treat"
)
```

Arguments

df	Data frame.
tte.name	Character. Name of time-to-event variable.
event.name	Character. Name of event indicator variable.
treat.name	Character. Name of treatment variable.

Value

List with tau, RMST, RMST for treatment, RMST for control.

Examples

```
## Not run:
library(survival)
df <- data.frame(
  tte = gbsg$rfstime / 30.4375,
  event = gbsg$status,
  treat = gbsg$hormon
)
rmst_calculation(df, tte.name = "tte", event.name = "event",
  treat.name = "treat")

## End(Not run)
```

run_null_calibration *Run Null Calibration Simulation*

Description

Runs ForestSearch under the null hypothesis at multiple sample sizes to calibrate L_{eff} , then returns a calibration object for predicting corrected FPR.

Usage

```
run_null_calibration(
  sim_null_fn,
  fs_args,
  d_eff_fn,
  N_values = c(200, 500, 1000),
  c1 = 1.25,
  c2 = 1,
  n_sims = 100L,
  prop_harm = 0.3,
  n_min = 60L,
  seed_base = 42L,
  verbose = TRUE
)
```

Arguments

sim_null_fn	Function. Generates one H0 dataset. Signature: <code>sim_null_fn(n, seed)</code> returning a <code>data.frame</code> suitable for <code>forestsearch()</code> .
fs_args	Named list. Arguments passed to <code>forestsearch()</code> (excluding <code>df.analysis</code> , <code>hr.threshold</code> , <code>hr.consistency</code> , and <code>seedit</code> , which are set by the wrapper).
d_eff_fn	Function. Computes d_{eff} for a subgroup. Signature: <code>d_eff_fn(n_sg)</code> returning numeric. Example: <code>function(n) d_eff_binary(n, 0.30)</code> .
N_values	Integer vector. Sample sizes to simulate. At least 2 required. Default: <code>c(200, 500, 1000)</code> .
c1	Numeric. Screening threshold for calibration. Default: 1.25.
c2	Numeric. Consistency threshold for calibration. Default: 1.0.
n_sims	Integer. Simulations per sample size. Default: 100.
prop_harm	Numeric. Expected proportion in harm subgroup (for computing d_{eff}). Default: 0.30.
n_min	Integer. Minimum subgroup size. Default: 60.
seed_base	Integer. Base seed. Default: 42.
verbose	Logical. Print progress. Default: TRUE.

Details

For each N in `N_values`, the function:

1. Generates `n_sims` datasets via `sim_null_fn()`
2. Runs `forestsearch()` on each
3. Computes the simulated FPR = proportion with any H found
4. Computes P_1 via the GLM approximation
5. Calls `calibrate_L_eff()` to fit the power-law model

`ForestSearch` is run with `parallel_args = list(plan = "sequential")` to avoid nested parallelism. The outer simulation loop is sequential. For faster execution, wrap in `foreach %dofuture%` externally.

Value

An "leff_calibration" object.

Examples

```
# Binary DGM under H0
sim_null <- function(n, seed) {
  set.seed(seed)
  data.frame(
    id = seq_len(n),
    treat = rbinom(n, 1, 0.5),
    bm1 = as.factor(rbinom(n, 1, 0.70)),
    bm2 = as.factor(rbinom(n, 1, 0.50)),
  )
}
```

```

    age = round(rnorm(n, 55, 10)),
    ecog = as.factor(sample(0:1, n, TRUE, c(0.6, 0.4))),
    progressed = rbinom(n, 1, 0.30)
  )
}

fs_args <- list(
  confounders.name = c("bm1", "bm2", "age", "ecog"),
  outcome.name = "progressed",
  event.name = "progressed",
  treat.name = "treat",
  id.name = "id",
  outcome_type = "binary",
  effect_measure = "OR",
  adverse_outcome = TRUE,
  pconsistency.threshold = 0.90,
  fs.splits = 200,
  n.min = 60,
  maxk = 2,
  use_lasso = TRUE,
  use_grf = TRUE,
  use_twostage = TRUE,
  is.RCT = TRUE,
  details = FALSE,
  plot.sg = FALSE,
  parallel_args = list(plan = "sequential",
                        workers = 1,
                        show_message = FALSE)
)

cal <- run_null_calibration(
  sim_null_fn = sim_null,
  fs_args = fs_args,
  d_eff_fn = function(n) d_eff_binary(n, 0.30),
  N_values = c(200, 500, 1000),
  n_sims = 50,
  verbose = TRUE
)
print(cal)

```

run_simulation_analysis

Run One Simulation Replicate

Description

General replacement for the legacy `run_simulation_analysis()` that was coupled to `simulate_from_gbsg_dgm()` and GBSG-specific column names. This version calls `simulate_from_dgm` and accepts explicit column-name parameters, making it applicable to any DGM built with `generate_aft_dgm_flex`.

Usage

```
run_simulation_analysis(
  sim_id,
  dgm,
  n_sample,
  analysis_time = Inf,
  cens_adjust = 0,
  max_follow = NULL,
  muC_adj = NULL,
  confounders_base = c("v1", "v2", "v3", "v4", "v5", "v6", "v7"),
  n_add_noise = 0L,
  outcome_name = "y_sim",
  event_name = "event_sim",
  treat_name = "treat_sim",
  harm_col = "flag_harm",
  run_fs = TRUE,
  run_fs_grf = TRUE,
  run_grf = TRUE,
  methods = NULL,
  fs_params = list(),
  grf_params = list(),
  cox_formula = NULL,
  cox_formula_adj = NULL,
  n_sims_total = NULL,
  seed_base = 8316951L,
  verbose = FALSE,
  verbose_n = NULL,
  debug = FALSE,
  keep = character(0),
  keep_first_n = Inf
)
```

Arguments

<code>sim_id</code>	Integer. Simulation replicate index (used as seed offset).
<code>dgm</code>	An "aft_dgm_flex" object from generate_aft_dgm_flex or setup_gbsg_dgm , or a "glm_dgm" object from generate_glm_dgm .
<code>n_sample</code>	Integer. Per-replicate sample size.
<code>analysis_time</code>	Numeric. Calendar time of analysis on the DGM time scale. Use Inf (default) for no administrative censoring — equivalent to the legacy <code>max_follow = Inf</code> .
<code>cens_adjust</code>	Numeric. Log-scale shift to censoring times passed to <code>simulate_from_dgm(cens_adjust = ...)</code> . Replaces legacy <code>muC_adj</code> . Default 0.
<code>max_follow</code>	Deprecated. Use <code>analysis_time</code> instead. If supplied, its value is forwarded to <code>analysis_time</code> with a warning. Retained for backward compatibility with legacy scripts.

muC_adj	Deprecated. Use cens_adjust instead. If supplied, its value is forwarded to cens_adjust with a warning. Retained for backward compatibility with legacy scripts.
confounders_base	Character vector of base confounder names.
n_add_noise	Integer. Number of independent $N(0,1)$ noise variables to append. Default 0L.
outcome_name	Name of the observed time column in simulated data. Default "y_sim".
event_name	Name of the event indicator column. Default "event_sim".
treat_name	Name of the treatment column. Default "treat_sim".
harm_col	Name of the true-subgroup indicator column. Default "flag_harm".
run_fs	Logical. Run ForestSearch (LASSO). Default TRUE.
run_fs_grf	Logical. Run ForestSearch (LASSO + GRF). Default TRUE.
run_grf	Logical. Run standalone GRF. Default TRUE.
methods	Optional specification of analysis arms that generalizes the run_fs / run_fs_grf / run_grf booleans. When non-NULL the booleans are ignored and every arm is run via forestsearch() (through .run_fs_analysis_gen()), so all arms — including dina-* and grf-* — inherit the bias-corrected post-selection inference. Two forms are accepted: • A character vector of canonical "<engine>-<focus>" labels, e.g. c("fs-effMaxSG", "dina-effMaxSG", "grf-tree"), expanded internally to per-arm forestsearch() overrides. Engines are "fs" (subgroup_method = "consistency"), "dina", and "grf" (grf_selection = "frontier", or "tree" for the grf-tree label). Defaults applied during expansion: selection_rule = "neighborhood"; for fs-* arms use_grf = TRUE, use_dina = TRUE, use_lasso = FALSE, and n.min = NULL (the sample-size-adaptive rule). • A named list of explicit forestsearch() argument-override lists, label = name, e.g. list("fs-effMaxSG" = list(subgroup_method = "consistency", sg_focus = "effMaxSG", use_grf = TRUE, use_dina = TRUE, use_lasso = FALSE), "grf-tree" = list(subgroup_method = "grf", grf_selection = "tree")). Each arm's overrides are merged on top of default_sim_params() overlaid with fs_params; per-arm overrides take precedence. The merge preserves an explicit n.min = NULL. Default NULL preserves the legacy boolean behavior.
fs_params	Named list of ForestSearch parameter overrides. Any element of default_sim_params() can be overridden, including parallel_args (see <i>Parallel Processing</i> section below). Passing NULL or an empty list uses defaults throughout.
grf_params	Named list of GRF parameter overrides.
cox_formula	Optional Cox formula for unadjusted ITT.
cox_formula_adj	Optional adjusted Cox formula.
n_sims_total	Integer. Total simulations (for progress messages).
seed_base	Integer. Base seed; replicate seed = seed_base + sim_id. Default 8316951L.
verbose	Logical. Print progress messages. Default FALSE.

verbose_n	Integer. If set, only print for <code>sim_id <= verbose_n</code> . Default <code>NULL</code> .
debug	Logical. Print detailed debug output. Default <code>FALSE</code> .
keep	Character vector. Optional names of heavy diagnostic objects to attach as a list in <code>attr(result, "diagnostics")</code> . Allowed values (any combination): "fs_full" The full <code>forestsearch()</code> result object (minus the <code>df.est</code> analysis data frame, which is stripped to control memory). Includes <code>sg.harm</code> , <code>grp.consistency</code> , <code>confounders.candidate</code> , <code>confounders.evaluated</code> , GRF cuts, and timing. "df.est" The analysis data frame with <code>treat.recommend</code> column attached. Heavy; use sparingly. "candidate_table" The data table of candidate subgroups evaluated (<code>fs_full\$grp.consistency\$out</code>). Medium-weight; useful for diagnosing which candidates were near-threshold. "grf_full" The full <code>grf.subg.harm.*()</code> result object. Default character(<code>0</code>) (attach nothing; current behavior).
keep_first_n	Integer. When <code>keep</code> is non-empty, attach the requested diagnostics only if <code>sim_id <= keep_first_n</code> . Default <code>Inf</code> (attach for all replicates). Use this to capture full details for, e.g., the first 20 simulations without bloating the result object for long production runs.

Value

A data table with one row per analysis method. Always-present scalar columns cover: true-subgroup size/effect from the DGM, per-method detection flag (`any.H`), estimated subgroup size (`size.H`, `size.Hc`), effect estimates (`hr.H.hat`, `hr.Hc.hat`, `hr.itt`, and their true counterparts), classification metrics against the DGM-stored subgroup (`sens`, `spec`, `ppv`, `npv`), and pairwise concordance between methods (`agree.*.*`, `kappa.*.*`). As of v0.2.0 the following ForestSearch diagnostic columns are also always populated (NA for GRF rows):

`sg.def` Character. Cut-expression string of the identified subgroup, formed as `paste(fs_full$sg.harm, collapse = " & ")`. Empty string when no subgroup was identified. Read across all replicates to diagnose which cuts the method is selecting (e.g., `table(results$sg.def)`).

`sg.n_factors` Integer. Number of factors in the identified subgroup definition.

`n_candidates_evaluated` Integer. Number of candidate subgroups actually evaluated for consistency (before or at the early-stop candidate).

`n_candidates_total` Integer. Total number of candidate subgroups available.

`n_passed` Integer. Number of candidates meeting the consistency threshold.

`consistency_algorithm` Character. "fixed" or "twostage".

`early_stop_triggered` Logical. Whether evaluation stopped early due to `stop_threshold`.

`fs_minutes` Numeric. Wall time for the FS run in minutes.

The analogous GRF row carries `sg.def` (from `grf_result$sg.harm.id`) and `grf_selected_depth`. When `keep` is non-empty and `sim_id <= keep_first_n`, the heavy objects are attached as `attr(result, "diagnostics")` — a named list with one element per keeper (`fs_full`, `grf_full`, etc.).

```
run_single_consistency_split
```

Run Single Consistency Split

Description

Performs one random 50/50 split and evaluates whether both halves meet the HR consistency threshold.

Usage

```
run_single_consistency_split(
  df.x,
  N.x,
  hr.consistency,
  cox_init = 0,
  estimator_fn = NULL,
  consistency_threshold = NULL,
  adjust_covariates = NULL
)
```

Arguments

<code>df.x</code>	data.table. Subgroup data with columns Y, Event, Treat.
<code>N.x</code>	Integer. Number of observations in subgroup.
<code>hr.consistency</code>	Numeric. c2, an effect threshold – the minimum effect required in <i>each</i> half of a split for that split to count as consistent. Not a proportion and not the consistency rate. <code>forestsearch()</code> calls this <code>consistency.threshold/hr.consistency</code> .
<code>cox_init</code>	Numeric. Initial value for Cox model (log HR).
<code>estimator_fn</code>	Closure or NULL. Effect-estimator closure from make_effect_estimator for GLM outcomes. Default NULL (uses Cox model).
<code>consistency_threshold</code>	Numeric or NULL. Threshold for GLM consistency evaluation. Default NULL (uses <code>hr.consistency</code>).
<code>adjust_covariates</code>	Character vector or NULL. Additional Cox model terms (e.g. "strata(x1)") passed to get_split_hr_fast on the survival path. Ignored on the GLM path. Default NULL (unadjusted).

Value

Numeric. 1 if both splits meet threshold, 0 if not, NA if error.

Examples

```
library(data.table)
set.seed(1)
df <- data.table(
  Y      = rexp(100),
  Event = rbinom(100, 1, 0.55),
  Treat = rep(0:1, 50)
)
run_single_consistency_split(df, N.x = 100, hr.consistency = 1.0)
```

run_subgroup_sims	<i>Run the extreme-subgroups simulation study</i>
-------------------	---

Description

Draws `n_sims` synthetic trials from a calibrated DGM via [simulate_from_dgm\(\)](#), fits `fit` in every pre-defined subgroup of every trial, and returns the raw result matrices plus full provenance. With the default `fit`, benchmarks, seeds, and skip rule, the matrices are bit-identical to those produced by the `extreme_subgroups` vignettes for the matching baseline, and the returned object is byte-compatible with their RDS payload: `saveRDS(sims, path)` yields a file the design-comparison memo loads unchanged.

Usage

```
run_subgroup_sims(
  dgm,
  subgroups,
  n_sims,
  fit = subgroup_cox(survival::Surv(y_sim, event_sim) ~ treat_sim),
  baseline = c("resample", "fixed"),
  n = NULL,
  analysis_time,
  max_entry,
  cens_adjust,
  cutpoints = list(),
  benchmarks = benchmark_spec(),
  min_n = 5L,
  workers = NULL,
  seed_base = 0L,
  rand_seed_offset = 1000000L,
  hr_true = NULL,
  k_treat = NULL,
  future_packages = c("forestsearch", "survival"),
  validate = TRUE,
  verbose = FALSE
)
```

Arguments

dgm	A DGM from generate_aft_dgm_flex() (survival) or generate_glm_dgm() (GLM). For baseline = "fixed" (survival only) it must carry df_source.
subgroups	List of subgroup definitions, each a list with character scalars id (an expression over trial columns, exposed cut-points, flag_itt, and benchmark columns), name, and grp.
n_sims	Number of simulated trials.
fit	Per-subgroup analysis function data -> c(estimate, upper); see subgroup_cox() . Default is an unstratified Cox fit on the simulated columns (the vignettes pass a grade-stratified formula); for a "glm_dgm" the default is subgroup_glm() constructed from the DGM's outcome_type / effect_measure (and, for binary, its adverse_outcome).
baseline	"resample" (random-X: draw n patients from the super-population per trial) or "fixed" (conditional-on-X: every trial is the df_source panel). Survival only; a "glm_dgm" always resamples.
n	Patients per trial. Required for "resample" and for a "glm_dgm"; must be NULL for "fixed", where N is nrow(dgm\$df_source) by construction.
analysis_time, max_entry, cens_adjust	Passed to simulate_from_dgm() (administrative censoring time, recruitment window, calibrated censoring shift). Survival-only: supplying any of them with a "glm_dgm" is an error.
cutpoints	Named list of scalars exposed as columns of each simulated trial so id expressions like "age <= age_med" resolve.
benchmarks	A benchmark_spec() , or NULL to skip benchmark generation entirely (no RNG is consumed for it).
min_n	Subgroups with fewer rows are skipped for that trial (row stays NA), default 5L as in the vignettes.
workers	Parallel workers. NULL (default) reuses a caller-established non-sequential future::plan() when one exists – no worker-pool spawn or teardown, the mode the vignettes use after their parallel-setup chunk – and otherwise creates a multisession plan on ceiling(0.90 * availableCores()), restored on exit. A numeric value forces a plan for this call (restored on exit): a fraction in (0, 1) of availableCores(), an integer count, or 1 for sequential. Results are identical under every setting (seeding is per-iteration); only wall time differs. sim_config records workers, plan_reused, and t_plan_secs.
seed_base, rand_seed_offset	Seed scheme; defaults (0L, 1e6L) reproduce the vignettes.
hr_true, k_treat	Optional provenance values stored verbatim in the result (the vignettes store 0.70 and the calibrated k_treat). hr_true is the true effect on the fitter's estimate scale: a hazard ratio for Cox fits, a mean difference for subgroup_glm() fits.
future_packages	Packages loaded on workers, default c("forestsearch", "survival") as in the vignettes.

validate	Run validate_subgroups() first (default TRUE).
verbose	Print progress lines.

Details

Per-iteration seeding is explicit (trial ss uses seed_base + ss for the DGM draw and seed_base + ss + rand_seed_offset for benchmark membership), so results are independent of the parallel schedule: any workers setting, including sequential, gives identical matrices.

GLM dispatch: when dgm inherits "glm_dgm" ([generate_glm_dgm\(\)](#)), trials are drawn by [simulate_from_glm_dgm\(\)](#) (n is required; the survival-only analysis_time / max_entry / cens_adjust must not be supplied; baseline = "fixed" uses the stored df_source panel), the default fit becomes [subgroup_glm\(\)](#) constructed from the DGM's own outcome_type and effect_measure (binary DGMs also pass their adverse_outcome; a continuous DGM resolves to the same fitter as before), and the result additionally carries the fitter's effect scale metadata with design = "glm". The sim_hrs / sim_ubs matrices then hold that fitter's (estimate, upper_bound) pairs – the field names are retained for structural compatibility, exactly as hr.threshold serves generic-threshold duty for GLM outcomes in [forestsearch\(\)](#).

Value

An object of class "subgroup_sims": a named list with design, n_sims, matrices sim_hrs / sim_ubs / sim_ns (n_sims x subgroups, columns named by subgroup name), subgroups, sim_config, cens_adjust, k_treat, hr_true, created, r_version, and forestsearch_version. For a "glm_dgm": design = "glm", cens_adjust = NULL, and an additional effect field holding the fitter's scale metadata (see [subgroup_glm\(\)](#)); survival results are constructed exactly as before, with no new fields.

See Also

[summary.subgroup_sims\(\)](#), [benchmark_spec\(\)](#), [subgroup_cox\(\)](#), [subgroup_glm\(\)](#), [validate_subgroups\(\)](#)

Examples

```
## Not run:
sims <- run_subgroup_sims(
  dgm = dgm_uniform, subgroups = subgroups, n_sims = 1000,
  fit = subgroup_cox(survival::Surv(y_sim, event_sim) ~ treat_sim +
    survival::strata(grade)),
  baseline = "fixed",
  analysis_time = 84, max_entry = 24,
  cens_adjust = cal_uniform$cens_adjust,
  cutpoints = list(age_med = median(gbsg$age)),
  hr_true = 0.70, k_treat = k_treat_uniform
)
saveRDS(sims, "results/extreme_sims_fixed_1000_payload.rds")

## End(Not run)
```

save_forestplot	<i>Save ForestSearch Forest Plot to File</i>
-----------------	--

Description

Saves a forest plot to a file (PDF, PNG, etc.) with explicit dimensions.

Usage

```
save_forestplot(x, filename, width = 12, height = 10, dpi = 300, bg = "white")
```

Arguments

x	An fs_forestplot object.
filename	Character. Output filename. Extension determines format.
width	Numeric. Plot width in inches. Default: 12.
height	Numeric. Plot height in inches. Default: 10.
dpi	Numeric. Resolution for raster formats. Default: 300.
bg	Character. Background color. Default: "white".

Value

Invisibly returns the filename.

Examples

```
## Not run:
# Create plot with custom theme
large_theme <- create_forest_theme(base_size = 14, row_padding = c(6, 4))

result <- plot_subgroup_results_forestplot(
  fs_results = list(fs.est = fs, fs.bc = fs.bc),
  df_analysis = df.analysis,
  outcome.name = "time",
  event.name = "status",
  treat.name = "treatment",
  theme = large_theme
)

# Save to file
save_forestplot(result, "forest_plot.pdf", width = 14, height = 12)

## End(Not run)
```

save_leff_calibration *Save L_eff Calibration Results*

Description

Writes a standardized .rds bundle containing the fitted L_eff parameters and all supporting data. Called at the end of calibration documents (e.g., calibration_binary_leff.qmd).

Usage

```
save_leff_calibration(
  C,
  alpha,
  n_min,
  outcome_type,
  cal_data = NULL,
  dgm_description = "",
  n_sims_per_N = NA_integer_,
  output_dir = NULL,
  extra = list()
)
```

Arguments

C	Numeric. Fitted scale parameter.
alpha	Numeric. Fitted power-law exponent.
n_min	Integer. Reference minimum subgroup size.
outcome_type	Character. One of "binary", "survival", "count", "continuous".
cal_data	Data frame with columns N, sim_fpr, P1, L_eff (the raw calibration points).
dgm_description	Character. Free-text description of the DGM used for calibration.
n_sims_per_N	Integer. Simulations per sample size.
output_dir	Character. Directory for .rds files. Required: an error is raised when it is NULL (the default). The former default was "_data".
extra	List. Any additional items to store.

Value

Invisible path to the saved file.

Examples

```
## Not run:
save_leff_calibration(
  C = 0.220, alpha = 1.298, n_min = 60,
  outcome_type = "binary",
  cal_data = cal_df,
  dgm_description = "Binary DGM, 4 confounders (bm1, bm2, age, ecog)",
  n_sims_per_N = 5000,
  output_dir = tempdir()
)

## End(Not run)
```

select_best_subgroup	Select best subgroup based on criterion
----------------------	---

Description

Identifies the optimal subgroup according to the specified criterion

Usage

```
select_best_subgroup(values, sg.criterion, dmin.grf, n.max)
```

Arguments

values	Data frame. Node metrics from policy trees
sg.criterion	Character. "mDiff" for maximum difference, "Nsg" for largest size
dmin.grf	Numeric. Minimum difference threshold
n.max	Integer. Maximum allowed subgroup size (total sample size)

Value

Data frame row with best subgroup or NULL if none found

Examples

```
vals <- data.frame(diff = c(8.5, 6.2, 3.1), Nsg = c(120, 95, 80))
select_best_subgroup(values = vals, sg.criterion = "mDiff",
  dmin.grf = 6, n.max = 500)
```

sensitivity_analysis_k_inter

Sensitivity Analysis of Hazard Ratios to k_inter

Description

Analyzes how the interaction parameter `k_inter` affects hazard ratios in different populations (overall, harm subgroup, no-harm subgroup).

Usage

```
sensitivity_analysis_k_inter(
  k_inter_range = c(-5, 5),
  n_points = 21,
  plot = TRUE,
  ...
)
```

Arguments

<code>k_inter_range</code>	Numeric vector of length 2 specifying the range of <code>k_inter</code> values to analyze. Default is <code>c(-5, 5)</code> .
<code>n_points</code>	Integer number of points to evaluate within the range. Default is 21.
<code>plot</code>	Logical indicating whether to create visualization plots. Default is TRUE.
<code>...</code>	Additional arguments passed to generate_aft_dgm_flex .

Details

This function evaluates the hazard ratios at evenly spaced points across the `k_inter` range. If `plot = TRUE`, it creates a 4-panel visualization showing:

1. Harm subgroup HR vs `k_inter`
2. All HRs (overall, harm, no-harm) vs `k_inter`
3. Ratio of HRs (harm/no-harm) showing effect modification
4. Table of key values

Value

A data.frame of class "k_inter_sensitivity" with columns:

k_inter Numeric `k_inter` value
hr_harm Numeric hazard ratio in harm subgroup
hr_no_harm Numeric hazard ratio in no-harm subgroup
hr_overall Numeric overall hazard ratio
subgroup_size Integer size of harm subgroup

Examples

```
## Not run:
# Analyze sensitivity to k_inter
sensitivity_results <- sensitivity_analysis_k_inter(
  k_inter_range = c(-2, 2),
  n_points = 11,
  data = survival::gbsg,
  continuous_vars = c("age", "er", "pgr"),
  factor_vars = c("meno", "grade"),
  outcome_var = "rfstime",
  event_var = "status",
  treatment_var = "hormon",
  subgroup_vars = c("er", "meno"),
  subgroup_cuts = list(er = 20, meno = 0),
  model = "alt",
  plot = TRUE
)

# Results show relationship between k_inter and HRs
print(sensitivity_results)

## End(Not run)
```

setup_gbsg_dgm

Set Up a GBSG-Based AFT Data Generating Mechanism

Description

Creates a GBSG-based data generating mechanism that is fully compatible with [simulate_from_dgm](#). This is the replacement for `create_gbsg_dgm()`: it accepts exactly the same arguments and produces the same numeric output, but returns an object of class "aft_dgm_flex" instead of "gbsg_dgm".

Usage

```
setup_gbsg_dgm(
  model = c("alt", "null"),
  k_treat = 1,
  k_inter = 1,
  k_z3 = 1,
  z1_quantile = 0.25,
  n_super = 5000L,
  cens_type = c("weibull", "uniform"),
  use_rand_params = FALSE,
  seed = 8316951L,
  verbose = FALSE,
  k_random_noise = 0L,
  noise_seed = 20260807L
)
```

Arguments

model	Character. Either "alt" for alternative hypothesis with heterogeneous treatment effects, or "null" for uniform treatment effect. Default: "alt"
k_treat	Numeric. Treatment effect multiplier applied to the treatment coefficient from the fitted AFT model. Values > 1 strengthen the treatment effect. Default: 1
k_inter	Numeric. Interaction effect multiplier for the treatment-subgroup interaction ($z1 * z3$). Only used when model = "alt". Higher values create more heterogeneity between HR(H) and HR(Hc). Default: 1
k_z3	Numeric. Effect multiplier for the z3 (menopausal status) coefficient. Default: 1
z1_quantile	Numeric. Quantile threshold for z1 (estrogen receptor). Observations with ER $\leq$ quantile are coded as z1 = 1. Default: 0.25
n_super	Integer. Size of super-population for empirical HR estimation. Default: 5000
cens_type	Character. Censoring distribution type: "weibull" or "uniform". Default: "weibull"
use_rand_params	Logical. If TRUE, modifies confounder coefficients using estimates from randomized subset (meno == 0). Default: FALSE
seed	Integer. Random seed for super-population generation. Default: 8316951
verbose	Logical. Print diagnostic information. Default: FALSE
k_random_noise	Integer. Number of standard-normal noise columns (noise1, noise2, ...) drawn once onto the super-population at construction. Default 0L (no noise columns; output identical to previous versions).
noise_seed	Integer. Seed for the population-level noise draw, used only when k_random_noise > 0. Default 20260807L, the committed population-noise seed (commit 2a4787bc).

Details

Internally the function calls `create_gbsg_dgm()` and then:

1. Adds a `df_super` field with column names aligned to `simulate_from_dgm()` conventions (`lin_pred_1`, `lin_pred_0`, `lin_pred_cens_1`, `lin_pred_cens_0`, `flag_harm`).
2. Adds a `model_params$tau` field (= `model_params$sigma`) and a `model_params$censoring` sub-list.
3. Sets class to `c("aft_dgm_flex", "gbsg_dgm", "list")`.

The original `df_super_rand` field is kept so that `compute_dgm_cde()` and `print.gbsg_dgm` continue to work.

Noise columns requested via `k_random_noise` are inert $N(0,1)$ population attributes intended as candidate confounders; the builder never puts them in the outcome model. They are drawn once onto the super-population (both `df_super_rand` and `df_super`), so simulated trials and evaluation frames inherit them by row resampling.

Value

An object of class `c("aft_dgm_flex", "gbsg_dgm", "list")` with all fields from `create_gbsg_dgm()` plus:

`df_super` Super-population data frame with `simulate_from_dgm()`-compatible column names.

`model_params$tau` Copy of `model_params$sigma`.

`model_params$censoring` Sub-list with type, mu, tau for the censoring model.

`noise_names` Character vector of noise column names (`character(0)` when `k_random_noise = 0`).

`noise_scheme` "population" when noise was drawn, otherwise "none".

`noise_seed` The seed used for the noise draw, or `NA_integer_` when none was drawn.

See Also

[create_gbsg_dgm](#), [simulate_from_dgm](#), [compute_dgm_cde](#)

Examples

```
## Not run:
dgm <- setup_gbsg_dgm(model = "alt", k_inter = 2, verbose = FALSE)
dgm <- compute_dgm_cde(dgm)
print(dgm)
sim <- simulate_from_dgm(dgm, n = 400, seed = 1)

## End(Not run)
```

`setup_parallel_SGcons` *Set up parallel processing for subgroup consistency*

Description

Sets up parallel processing using the specified approach and number of workers.

Usage

```
setup_parallel_SGcons(
  parallel_args = list(plan = "multisession", workers = 4, show_message = TRUE)
)
```

Arguments

`parallel_args` List with plan (character), workers (integer), and show_message (logical).

Value

None. Sets up parallel backend as side effect.

Examples

```
setup_parallel_SGcons(list(plan = "sequential", workers = 1,
                           show_message = FALSE))
future::plan(future::sequential) # reset
```

sgdist_fig_height	<i>Adaptive Figure Height for a Subgroup-Distribution Plot</i>
-------------------	--

Description

Computes a reasonable knitr fig.height (in inches) for a [plot_sg_distribution\(\)](#) panel based on the number of bars, keeping the visual quality consistent across scenarios with very different numbers of distinct subgroup labels.

Usage

```
sgdist_fig_height(
  n_bars,
  floor_in = 4,
  ceil_in = 10,
  per_bar = 0.35,
  base_in = 3
)
```

Arguments

n_bars	Integer. Number of bars in the plot, typically retrieved via attr(p, "n_bars") on a plot returned by plot_sg_distribution() .
floor_in	Numeric. Minimum height in inches. Default: 4.0.
ceil_in	Numeric. Maximum height in inches. Default: 10.0.
per_bar	Numeric. Height contribution per bar in inches. Default: 0.35.
base_in	Numeric. Base height added to per_bar * n_bars before applying the floor / ceiling. Default: 3.0.

Value

Numeric scalar, a height in inches in [floor_in, ceil_in]. Suitable for the fig.height chunk option.

See Also

[plot_sg_distribution](#)

Examples

```
## Not run:
p <- plot_sg_distribution(results, placeholder_on_empty = TRUE)
h <- sgdist_fig_height(attr(p, "n_bars"))
# Use `h` as the fig.height chunk option.

## End(Not run)
```

SGplot_estimates	<i>Violin/Boxplot Visualization of HR Estimates</i>
------------------	---

Description

Creates violin plots with embedded boxplots showing the distribution of hazard ratio estimates across simulations for different analysis populations. Supports symmetric trimming to handle extreme values that can distort the display when small subgroups produce very large HR estimates.

Usage

```
SGplot_estimates(
  df,
  label_training = "Training",
  label_testing = "Testing",
  label_itt = "ITT (stratified)",
  label_sg = "Testing (subgroup)",
  trim_fraction = NULL,
  ylim = NULL,
  show_summary = NULL,
  title = "Distribution of HR Estimates Across Simulations",
  subtitle = NULL
)
```

Arguments

df	data.frame or data.table. Simulation results from mrct_region_sims
label_training	Character. Label for training data estimates. Default: "Training"
label_testing	Character. Label for testing data estimates. Default: "Testing"
label_itt	Character. Label for ITT estimates. Default: "ITT (stratified)"
label_sg	Character. Label for subgroup estimates. Default: "Testing (subgroup)"
trim_fraction	Numeric or NULL. Fraction of observations to trim from each tail (e.g., 0.01 trims the lowest 1% When non-NULL, trimmed means and SDs are computed for each group, extreme observations are flagged, and the y-axis is clipped to the trimmed data range. Set to NULL (default) for no trimming (backward compatible).

<code>ylim</code>	Numeric vector of length 2 or NULL. Explicit y-axis limits as <code>c(lower, upper)</code> . Overrides automatic limits from trimming. Default: NULL (auto).
<code>show_summary</code>	Logical. Annotate each violin with mean (SD) below the x-axis labels. When trimming is active, displays trimmed statistics. Default: TRUE when <code>trim_fraction</code> is non-NULL, FALSE otherwise.
<code>title</code>	Character. Plot title. Default: "Distribution of HR Estimates Across Simulations".
<code>subtitle</code>	Character or NULL. Plot subtitle. When trimming is active and subtitle is NULL, an auto-generated note indicating the trim fraction and number of flagged observations is shown. Default: NULL.

Value

List with components:

dfPlot_estimates data.table formatted for plotting, with a trimmed logical column when trimming is active

plot_estimates ggplot2 object

trim_info List of per-group trimming diagnostics (NULL when no trimming). Each element contains: `n_total`, `n_trimmed`, `n_flagged`, `raw_mean`, `raw_sd`, `trimmed_mean`, `trimmed_sd`, `lower_bound`, `upper_bound`.

See Also

[mrct_region_sims](#) for generating simulation results, [summaryout_mrct](#) for tabular summaries with trimming

Examples

```
## Not run:
# Default (no trimming) - backward compatible
plot_results <- SGplot_estimates(
  results_alt,
  label_training = "Non-Region A, ITT",
  label_itt = "Overall, ITT",
  label_testing = "Region A, ITT",
  label_sg = "Region A, identified subgroup"
)
print(plot_results$plot_estimates)

# With 1% symmetric trimming
plot_results <- SGplot_estimates(
  results_alt,
  label_training = "Non-AP, ITT",
  label_itt = "Overall, ITT",
  label_testing = "AP, ITT",
  label_sg = "AP, identified subgroup",
  trim_fraction = 0.01
)
print(plot_results$plot_estimates)
```

```

# Inspect trimming diagnostics
print(plot_results$trim_info)

# With explicit y-axis limits
plot_results <- SGplot_estimates(
  results_alt,
  trim_fraction = 0.01,
  ylim = c(0.2, 3.0)
)

## End(Not run)

```

sg_consistency_out	<i>Output Subgroup Consistency Results</i>
--------------------	--

Description

Returns the top subgroup(s) and recommended treatment flags.

Usage

```

sg_consistency_out(
  df,
  result_new,
  sg_focus,
  index.Z,
  names.Z,
  details = FALSE,
  plot.sg = FALSE,
  by.risk = 12,
  confs_labels,
  is_glm = FALSE,
  selection_rule = "neighborhood",
  effect_neighborhood = 0.1,
  effect_log_scale = FALSE
)

```

Arguments

df	Data.frame. Original analysis data.
result_new	Data.table. Sorted subgroup results.
sg_focus	Character. Sorting focus criterion.
index.Z	Matrix. Subgroup factor indicators.
names.Z	Character vector. Factor column names.
details	Logical. Print details.

plot.sg Logical. Plot subgroup curves.
by.risk Numeric. Risk interval for plotting.
confs_labels Character vector. Human-readable labels.
is_glm Logical. If TRUE, suppresses Kaplan-Meier survival plots (which are not meaningful for GLM outcomes). Default FALSE.
selection_rule Character. Rule defining the candidate inclusion set for "hrMaxSG" / "hrMinSG". See [sort_subgroups](#). Default "neighborhood".
effect_neighborhood Numeric in $[0, 1)$. Relative tolerance for "hrMaxSG"/"hrMinSG" selection. See [sort_subgroups](#). Default 0.10.
effect_log_scale Logical. If TRUE, the hr column stores log-scale values and is exponentiated for the neighborhood test and for Pareto-frontier dominance. Default FALSE.

Value

List with elements:

result Sorted candidate table (top row = selected subgroup).
pareto_frontier Data.table of non-dominated candidates on (effect, N), both maximized. See [compute_pareto_frontier](#). May be NULL if computation failed.
sg.harm Factor-level cut names defining the selected subgroup.
sg.harm_label Human-readable subgroup labels.
df_flag Per-subject treatment-recommendation flags.
sg.harm.id Per-subject subgroup-membership indicator.

Examples

```
## Not run:
# sg_consistency_out is called internally by forestsearch().
# See forestsearch() for the standard entry point.
fs <- forestsearch(gbsg,
  confounders.name = c("age", "meno", "size", "grade3", "nodes", "pgr", "er"),
  outcome.name = "rfstime", treat.name = "hormon", event.name = "status")
fs$grp.consistency$out_sg

## End(Not run)
```

sg_tables

Enhanced Subgroup Summary Tables (gt output)

Description

Returns formatted summary tables for subgroups using the gt package, with search metadata and customizable decimal precision. Produces two tables: a treatment effect estimates table and an identified subgroups table, each with fully customizable titles and subtitles.

Usage

```
sg_tables(
  fs,
  which_df = "est",
  est_title = "Treatment Effect Estimates",
  est_caption = "Training data estimates",
  sg_title = "Identified Subgroups",
  sg_subtitle = NULL,
  potentialOutcome.name = NULL,
  hr_1a = NA,
  hr_0a = NA,
  ndecimals = 3,
  include_search_info = TRUE,
  subgroup_notation = NULL,
  font_size = 12
)
```

Arguments

<code>fs</code>	ForestSearch results object, or a <code>grf_glm_result</code> object from grf.subg.harm.glm . When a GRF object is supplied, only Table 1 (treatment effect estimates) is produced; Table 2 (identified subgroups with consistency) returns NULL.
<code>which_df</code>	Character. Which data frame to use ("est" or "testing").
<code>est_title</code>	Character or NULL. Main title for the estimates table (default: "Treatment Effect Estimates"). Rendered as bold markdown. Set to NULL to suppress the title and display only <code>est_caption</code> .
<code>est_caption</code>	Character. Subtitle for the estimates table (default: "Training data estimates").
<code>sg_title</code>	Character or NULL. Main title for the identified subgroups table (default: "Identified Subgroups"). Rendered as bold markdown. Set to NULL to suppress the title and display only <code>sg_subtitle</code> .
<code>sg_subtitle</code>	Character or NULL. Subtitle for the identified subgroups table. When NULL (default), an informative subtitle is auto-generated from <code>maxk</code> (e.g., "Two-factor subgroups (<code>maxk=2</code>)").
<code>potentialOutcome.name</code>	Character. Name of potential outcome variable (optional).
<code>hr_1a</code>	Character. Adjusted HR for subgroup 1 (optional).
<code>hr_0a</code>	Character. Adjusted HR for subgroup 0 (optional).
<code>ndecimals</code>	Integer. Number of decimals for formatted numbers (default: 3). Controls precision in both <code>sg10_out</code> (formatted via <code>gt::fmt_number</code>) and <code>tab_estimates</code> (threaded as <code>digits</code> to SG_tab_estimates_glm for arm summaries, Diff, effect-estimate CI, and bias-corrected CI). GLM and GRF-GLM paths only; the survival path uses its own formatting in SG_tab_estimates .
<code>include_search_info</code>	Logical. Include search metadata table (default: TRUE).

subgroup_notation Character or NULL. When set to "harm" or "benefit", overrides the default "Recommend"/"Questionable" labels with the Unicode labels from [get_sg_labels](#) (e.g., $\hat{H}$ / $\hat{H}^c$ for harm). Default NULL (use "Recommend"/"Questionable").

font_size Numeric. Font size in pixels for table text (default: 12).

Value

List with gt tables for estimates, subgroups, and optionally search info.

Examples

```
## Not run:
library(survival)
fs <- forestsearch(
  gbsg,
  confounders.name = c("age", "meno", "size", "grade3", "nodes", "pgr", "er"),
  outcome.name     = "rfstime",
  treat.name       = "hormon",
  event.name       = "status",
  fs.splits        = 50,
  use_lasso        = FALSE,
  use_grf          = FALSE
)
tabs <- sg_tables(fs)
tabs$tab_estimates

## End(Not run)
```

SG_tab_estimates	<i>Subgroup summary table estimates</i>
------------------	---

Description

Returns a summary table of subgroup estimates (HR, RMST, medians, etc.).

Usage

```
SG_tab_estimates(
  df,
  SG_flag,
  outcome.name = "tte",
  event.name   = "event",
  treat.name   = "treat",
  strata.name  = NULL,
  hr_1a = NA,
  hr_0a = NA,
  potentialOutcome.name = NULL,
  sg1_name = NULL,
```

```

    sg0_name = NULL,
    draws = 0,
    details = FALSE,
    return_medians = TRUE,
    est.scale = "hr"
  )

```

Arguments

df	Data frame.
SG_flag	Character. Subgroup flag variable.
outcome.name	Character. Name of outcome variable.
event.name	Character. Name of event indicator variable.
treat.name	Character. Name of treatment variable.
strata.name	Character. Name of strata variable (optional).
hr_1a	Character. Adjusted HR for subgroup 1 (optional).
hr_0a	Character. Adjusted HR for subgroup 0 (optional).
potentialOutcome.name	Character. Name of potential outcome variable (optional).
sg1_name	Character. Name for subgroup 1.
sg0_name	Character. Name for subgroup 0.
draws	Integer. Number of draws for resampling (optional).
details	Logical. Print details.
return_medians	Logical. Use medians or RMST.
est.scale	Character. Effect scale ("hr" or "1/hr").

Value

Data frame of subgroup summary estimates.

Examples

```

## Not run:
library(survival)
df <- data.frame(
  tte      = gbsg$rfstime / 30.4375,
  event    = gbsg$status,
  treat    = gbsg$hormon,
  treat.recommend = as.integer(gbsg$er > 0)
)
SG_tab_estimates(df, SG_flag = "ITT",
  outcome.name = "tte",
  event.name   = "event",
  treat.name   = "treat")

## End(Not run)

```

simulate_from_dgm

*Simulate Survival Data from AFT Data Generating Mechanism***Description**

Generates simulated survival data from a previously created AFT data generating mechanism (DGM). Samples from the super population and generates survival times with specified censoring.

Usage

```
simulate_from_dgm(
  dgm,
  n = NULL,
  rand_ratio = 1,
  entry_var = NULL,
  max_entry = 24,
  analysis_time = 48,
  cens_adjust = 0,
  draw_treatment = TRUE,
  seed = NULL,
  strata_rand = NULL,
  hrz_crit = NULL,
  keep_rand = FALSE,
  time_eos = NULL,
  replace = TRUE,
  baseline = c("resample", "fixed")
)
```

Arguments

dgm	An object of class "aft_dgm_flex" created by generate_aft_dgm_flex .
n	Integer specifying the sample size. If NULL (default), uses the entire super population without sampling.
rand_ratio	Numeric randomisation ratio (treatment:control). Default 1 (1:1 allocation).
entry_var	Character string naming an entry-time variable in the super population. If NULL, entry times are drawn as <code>Uniform(0, max_entry)</code> . Default NULL.
max_entry	Numeric maximum entry time for staggered entry simulation. Only used when <code>entry_var = NULL</code> . Default 24.
analysis_time	Numeric calendar time of analysis. Follow-up is <code>analysis_time - entry_time</code> . Must be on the same time scale as the DGM (i.e. the same units as <code>outcome_var</code> passed to <code>generate_aft_dgm_flex</code>). Default 48.
cens_adjust	Numeric log-scale adjustment to censoring distribution. Positive values increase censoring times; negative values decrease them. Default 0 (no adjustment).
draw_treatment	Logical. If TRUE (default), reassigns treatment according to <code>rand_ratio</code> . If FALSE, retains original treatment assignments from the super population.

seed	Integer random seed. Default NULL.
strata_rand	Character string naming a column in the sampled data for within-stratum balanced treatment allocation. If NULL, marginal allocation is used. Default NULL.
hrz_crit	Numeric log-HR threshold. If supplied, a column hrz_flag is added marking subjects with $\text{lin_pred}_1 - \text{lin_pred}_0 \geq \text{hrz_crit}$. Default NULL.
keep_rand	Logical. If TRUE, appends a rand_order column preserving the randomisation sequence. Default FALSE.
time_eos	Numeric secondary administrative censoring cutoff (end-of-study time on the DGM scale). Applied after follow_up censoring. Default NULL.
replace	Logical sampling scheme for drawing the n subjects from dgm\$df_super. TRUE (default) samples with replacement (the trial-resampling / bootstrap mode used for simulated trials). FALSE samples without replacement and so requires $n \leq \text{nrow}(\text{dgm}\$df_super)$; with $n = \text{nrow}(\text{dgm}\$df_super)$ it evaluates every super-population subject exactly once while still applying randomisation and censoring (unlike the $n = \text{NULL}$ path, which uses the pool as-is and skips both). Ignored when baseline = "fixed". Default TRUE.
baseline	Character; how baseline covariates are obtained per simulated trial. "resample" (default – the historical behaviour) draws subjects from dgm\$df_super, so covariates and therefore subgroup composition and sizes vary across trials. "fixed" uses dgm\$df_source – every source patient exactly once, in source order – so the baseline data and all covariate-defined subgroups are identical in every simulated trial; only treatment assignment, entry times, latent event times, and censoring are re-drawn per trial. Requires a DGM built by a package version that stores df_source (rebuild with generate_aft_dgm_flex() if absent). With baseline = "fixed", n must be NULL or equal $\text{nrow}(\text{dgm}\$df_source)$, and replace is ignored.

Details

Time-scale consistency:

All time parameters (analysis_time, max_entry, time_eos) must be expressed in the same units as outcome_var supplied to generate_aft_dgm_flex(). A common error is building the DGM on days (e.g. rfstime) and then passing analysis_time in months, which causes follow-up windows far shorter than the DGM event-time scale and produces universal administrative censoring (event_sim = 0 for all subjects).

Verify with: $\exp(\text{dgm}\$model_params\$mu)$ — the implied median event time should be plausible given your analysis_time.

n = NULL path:

When $n = \text{NULL}$ the entire super population is used as-is, with no staggered entry and no administrative censoring (follow_up = Inf). Treatment assignments and linear predictors already stored in dgm\$df_super are retained unchanged.

Censoring adjustment:

cens_adjust shifts the log-scale location parameter of the censoring distribution:

- $\text{cens_adjust} = \log(2)$ doubles expected censoring times.

- `cens_adjust = log(0.5)` halves expected censoring times.

Random-X vs fixed-X designs (baseline):

Under `baseline = "resample"` each simulated trial is a fresh draw of subjects, so between-trial variability reflects both covariate resampling and outcome generation – the "what if this trial had enrolled a different sample from the same population?" interpretation. Subgroup sizes fluctuate across trials.

Under `baseline = "fixed"` the covariate matrix is the observed trial data, identical in every simulated trial: subgroup definitions, memberships, and sizes are exactly those of the source data. Between-trial variability isolates the outcome-generation process (treatment assignment, latent event times, censoring, entry). This is the conditional-on-X (parametric-bootstrap-given-X) design. Note that entry times are still drawn `Uniform(0, max_entry)` per trial unless `entry_var` names a fixed column, and treatment is still re-randomised per trial unless `draw_treatment = FALSE`.

Value

A data.frame with columns:

`id` Subject identifier.

`treat` Original treatment from super population.

`treat_sim` Simulated treatment assignment.

`flag_harm` Subgroup indicator (1 = all subgroup conditions met).

`z_*` Covariate values.

`lin_pred_1, lin_pred_0` Counterfactual log-time linear predictors.

`y_sim` Observed survival time ($\min(T, C)$).

`event_sim` Event indicator (1 = event, 0 = censored).

`t_true` Latent true survival time (pre-censoring).

`c_time` Effective censoring time (post admin-censoring).

`hrz_flag` (Optional) Individual harm-zone indicator.

`rand_order` (Optional) Randomisation sequence index.

See Also

[generate_aft_dgm_flex](#), [check_censoring_dgm](#)

Examples

```
## Not run:
dgm <- setup_gbsg_dgm(model = "null", verbose = FALSE)
sim_data <- simulate_from_dgm(dgm, n = 200, seed = 42)
dim(sim_data)
head(sim_data[, c("y_sim", "event_sim", "treat_sim")])

## End(Not run)
```

simulate_from_gbsg_dgm

Simulate Trial Data from GBSG DGM

Description

Generates simulated clinical trial data from a GBSG-based data generating mechanism.

Usage

```
simulate_from_gbsg_dgm(
  dgm,
  n = NULL,
  rand_ratio = 1,
  sim_id = 1,
  max_follow = Inf,
  muC_adj = 0,
  min_cens = NULL,
  max_cens = NULL,
  draw_treatment = TRUE
)
```

Arguments

dgm	A "gbsg_dgm" object from create_gbsg_dgm
n	Integer. Sample size. If NULL, uses full super-population. Default: NULL
rand_ratio	Numeric. Randomization ratio (treatment:control). Default: 1 (1:1 randomization)
sim_id	Integer. Simulation ID used for seed offset. Default: 1
max_follow	Numeric. Administrative censoring time (months). Default: Inf (no administrative censoring)
muC_adj	Numeric. Adjustment to censoring distribution location parameter. Positive values increase censoring. Default: 0
min_cens	Numeric. Minimum censoring time for uniform censoring. Required if cens_type = "uniform"
max_cens	Numeric. Maximum censoring time for uniform censoring. Required if cens_type = "uniform"
draw_treatment	Logical. If TRUE, randomly assigns treatment. If FALSE, samples from existing treatment arms. Default: TRUE

Value

Data frame with simulated trial data including:

id Subject identifier

y.sim Observed follow-up time
event.sim Event indicator (1 = event, 0 = censored)
t.sim True event time (before censoring)
treat Treatment indicator
flag.harm Harm subgroup indicator
loghr_po Individual log hazard ratio (potential outcome)
v1-v7 Analysis factors

Examples

```
## Not run:
dgm <- create_gbsg_dgm(model = "alt", k_inter = 2.0)
sim_data <- simulate_from_gbsg_dgm(dgm, n = 500, sim_id = 1)

# Check AHR in simulated data
exp(mean(sim_data$loghr_po))

## End(Not run)
```

simulate_from_glm_dgm *Simulate Trial Data from a GLM Data Generating Mechanism*

Description

Generates a simulated clinical trial dataset by sampling from the super-population in a "glm_dgm" object. Assigns treatment and generates outcomes from the individual-level potential outcomes stored in the DGM.

Usage

```
simulate_from_glm_dgm(
  dgm,
  n = NULL,
  rand_ratio = 1,
  seed = NULL,
  draw_treatment = TRUE,
  replace = TRUE,
  baseline = c("resample", "fixed")
)
```

Arguments

dgm	An object of class "glm_dgm" created by generate_glm_dgm .
n	Integer. Sample size. If NULL (default), uses the entire super-population without resampling.
rand_ratio	Numeric. Treatment:control randomisation ratio. Default 1 (1:1 allocation).
seed	Integer. Random seed. Default NULL (no explicit seed; use the calling environment's RNG state).
draw_treatment	Logical. If TRUE (default), randomly assigns treatment. If FALSE, retains the treatment column from the super-population.
replace	Logical sampling scheme for drawing the n subjects from dgm\$df_super. TRUE (default) samples with replacement (bootstrap-style trial sampling). FALSE samples without replacement and so requires $n \leq \text{nrow}(\text{dgm}\$df_super)$; treatment is still re-randomised and the outcome re-drawn. Used to draw the full pool once ($n = \text{nrow}(\text{dgm}\$df_super)$, <code>replace = FALSE</code>) for the beta(Hhat) evaluation frame. Ignored on the $n = \text{NULL}$ path.
baseline	"resample" (default; draw the trial panel from the super-population) or "fixed" (use dgm\$df_source – the observed panel, every patient exactly once, observed order; requires a DGM built by a version that stores df_source; n must be NULL or $\text{nrow}(\text{df_source})$).

Value

A data frame with columns:

id Integer subject identifier.

y_sim Simulated outcome.

treat_sim Treatment indicator (0/1).

flag_harm True subgroup membership indicator.

Plus all covariate columns from the super-population.

For binary outcomes, y_sim is a 0/1 Bernoulli draw from the potential outcome probabilities (p_0 or p_1). There is no event_sim column – for ForestSearch, set `event.name = "y_sim"`.

See Also

[generate_glm_dgm](#), [run_simulation_analysis](#)

Examples

```
## Not run:
dgm <- generate_glm_dgm(...)
df <- simulate_from_glm_dgm(dgm, n = 500, seed = 42)
table(df$y_sim, df$treat_sim)

## End(Not run)
```

sim_region_metrics	<i>Region-consistency metrics on a simulated trial (ITT and optionally a subgroup)</i>
--------------------	--

Description

Region-consistency metrics on a simulated trial (ITT and optionally a subgroup)

Usage

```
sim_region_metrics(sim, region_col, subgroup_index = NULL, pi_preserve = 0.5)
```

Arguments

sim	Data frame of one simulated trial with columns y_sim, event_sim and treat_sim (as from simulate_from_dgm()) plus the region indicator.
region_col	Character name of the 0/1 region indicator column in sim.
subgroup_index	Optional logical vector of length nrow(sim) marking subgroup membership. When supplied, Cox HRs are also computed within the subgroup for the region and non-region strata. Default NULL.
pi_preserve	Numeric threshold for the Method 1 consistency check: the check passes when log(HR region) / log(HR overall) is at least pi_preserve. Default 0.5.

Value

A list with numeric vectors overall, non_region and region (each with elements hr, lo, hi, n, d; NA HR and limits when fewer than 10 subjects or 5 events), consistency_ratio, logical method1_pass, method2_pass and region_hr_below_1, and, when subgroup_index is supplied, region_subgroup and non_region_subgroup.

sort_subgroups	<i>Sort Subgroups by Focus (post-consistency)</i>
----------------	---

Description

Sorts a data.table of subgroup results according to the specified focus. For "hrMaxSG" and "hrMinSG", candidates are first partitioned into a *candidate inclusion set* (the "band"), then ranked within the band by sample size. The selection_rule argument controls how the band is defined:

- "neighborhood" – 1D effect-size band (default; legacy behaviour): a candidate is in-band iff its effect is within effect_neighborhood of the maximum.
- "pareto" – 2D Pareto-non-dominated set in (effect, N) space: a candidate is in-band iff no other candidate has both a larger effect and a larger N.
- "both" – intersection of the two: a candidate is in-band iff it satisfies the neighborhood test *and* is Pareto-non-dominated.

See Details.

Usage

```
sort_subgroups(
  result_new,
  sg_focus,
  selection_rule = "neighborhood",
  effect_neighborhood = 0.1,
  effect_log_scale = FALSE
)
```

Arguments

result_new A data.table of subgroup results with columns Pcons, hr, N, K.

sg_focus Character. Sorting focus. One of "hr", "maxSG", "hrMaxSG", "minSG", "hrMinSG".

selection_rule Character. Rule defining the candidate inclusion set for "hrMaxSG" / "hrMinSG". One of "neighborhood" (default; current behaviour), "pareto", or "both". Must be "neighborhood" for other sg_focus values.

effect_neighborhood Numeric in $[0, 1)$. Relative tolerance defining the effect-size neighborhood for "hrMaxSG" and "hrMinSG". A candidate is in the neighborhood iff its (natural-scale) effect is at least $(1 - \text{effect_neighborhood}) * \max(\text{effect})$. Default 0.10 (i.e., within 10% selection_rule "neighborhood" and "both"; ignored by "pareto" (and rejected if non-default with "pareto")).

effect_log_scale Logical. If TRUE, the hr column stores log-scale values (log-OR, log-IRR) and is exponentiated before applying the inclusion test. Default FALSE (Cox stores natural-scale HR).

Details**Sort keys by sg_focus:**

"hr" (-Pcons, -hr, K) – prefer high consistency and large effect.

"maxSG" (-N, -Pcons, K) – prefer large subgroups with high consistency.

"minSG" (N, -Pcons, K) – prefer small subgroups with high consistency.

"hrMaxSG" (-in_band, -N, -Pcons, -hr, K) – among candidates in the inclusion band defined by selection_rule, prefer the largest sample size.

"hrMinSG" (-in_band, N, -Pcons, -hr, K) – among candidates in the inclusion band defined by selection_rule, prefer the smallest sample size.

"hrMaxSG" and "hrMinSG" use a *lexicographic with candidate band* rule: effect size is primary, but a bounded set of candidates near (or trading off against) the maximum is accepted to optimize sample size.

Selection rules:

"neighborhood" 1D ε -band on the effect scale: $\text{in_band} = (\text{hr} \geq (1 - \text{effect_neighborhood}) * \max(\text{hr}))$. Setting effect_neighborhood = 0 reduces this to a strict max-effect filter. This is the legacy behaviour.

"pareto" 2D Pareto-non-dominated set: a candidate is in-band iff no other candidate has both $hr_j \geq hr_i$ and $N_j \geq N_i$ with at least one strict inequality. Reuses the same dominance computation as `compute_pareto_frontier()`. Removes the `effect_neighborhood` tuning parameter from the selection criterion.

"both" Intersection: in-band iff in the ϵ -band *and* Pareto-non-dominated. Strictest of the three.

Value

A sorted `data.table`. The top row is the selected subgroup under `sg_focus`; remaining rows are diagnostic.

Examples

```
library(data.table)
dt <- data.table(Pcons = c(0.92, 0.95, 0.88, 0.90),
                 hr     = c(2.5, 2.4, 2.3, 2.0),
                 N      = c(70, 100, 150, 200),
                 K      = c(1, 2, 1, 2))
# hrMaxSG with 10% neighborhood: HR 2.5, 2.4, 2.3 are in-band;
# among those, N = 150 is largest -> top row is N=150, hr=2.3.
sort_subgroups(dt, sg_focus = "hrMaxSG", effect_neighborhood = 0.10)

# Same data with selection_rule = "pareto": the dominance set in
# (hr, N) is used instead of the 1D effect band.
sort_subgroups(dt, sg_focus = "hrMaxSG", selection_rule = "pareto")
```

subgroup.consistency *Evaluate Subgroup Consistency*

Description

Evaluates candidate subgroups using split-sample consistency validation. For each candidate, repeatedly splits the data and checks whether the treatment effect direction is consistent across splits.

Usage

```
subgroup.consistency(
  df,
  hr.subgroups,
  hr.threshold = 1,
  hr.consistency = 1,
  pconsistency.threshold = 0.9,
  m1.threshold = Inf,
  n.splits = 100,
  details = FALSE,
  by.risk = 12,
  plot.sg = FALSE,
```



```

    maxk = 7,
    Lsg,
    confs_labels,
    sg_focus = "hr",
    selection_rule = "neighborhood",
    effect_neighborhood = 0.1,
    stop_Kgroups = 200,
    stop_threshold = NULL,
    show_candidate_summary = FALSE,
    max_print = 10,
    pconsistency.digits = 2,
    seed = 8316951,
    checking = FALSE,
    use_twostage = FALSE,
    twostage_args = list(),
    parallel_args = list(),
    estimator_fn = NULL,
    consistency_threshold = NULL,
    effect_label = "HR",
    effect_log_scale = FALSE,
    adjust_covariates = NULL,
    consistency_method = c("resample", "split"),
    glm_resample_spec = NULL
  )

```

Arguments

<code>df</code>	Data frame containing the analysis dataset. Must include columns for outcome (Y), event indicator (Event), and treatment (Treat).
<code>hr.subgroups</code>	Data.table of candidate subgroups from subgroup search, containing columns: HR, n, E, K, d0, d1, m0, m1, grp, and factor indicators.
<code>hr.threshold</code>	Numeric. <code>c1</code> , the screening threshold on a candidate's own effect, re-applied here to the supplied <code>hr.subgroups</code> table. On the comparison scale of the estimand (<code>forestsearch()</code> passes the already-resolved value for GLM outcomes). Default: 1.0
<code>hr.consistency</code>	Numeric. <code>c2</code> , an effect threshold – the minimum effect required in <i>each</i> half of a split for that split to count as consistent. Not a proportion and not the consistency rate; for the proportion see <code>pconsistency.threshold</code> . Default: 1.0
<code>pconsistency.threshold</code>	Numeric. <code>p*</code> , a proportion in $[0, 1]$ – the minimum fraction of splits that must be consistent, where consistency of one split is decided by <code>hr.consistency</code> (<code>c2</code>). <code>c2</code> sets the bar; <code>p*</code> counts how often it is met. Both names contain "consistency" and they are different quantities. Default: 0.9
<code>m1.threshold</code>	Numeric. Maximum <code>m1</code> threshold for filtering. Default: Inf
<code>n.splits</code>	Integer. Number of splits for consistency evaluation. Default: 100
<code>details</code>	Logical. Print progress details. Default: FALSE

by.risk	Numeric. Risk interval for KM plots. Default: 12
plot.sg	Logical. Generate subgroup plots. Default: FALSE
maxk	Integer. Maximum number of factors in subgroup. Default: 7
Lsg	List of subgroup parameters.
confs_labels	Character vector mapping factor names to labels.
sg_focus	Character. Subgroup selection criterion. One of: "hr" (or "eff") Sort by $(-Pcons, -hr, K)$; pick top. "maxeff" Sort by $(-hr, K)$; pick top. No consistency filter. Consistency (Pcons) does not enter the sort key, so it is evaluated for the SELECTED winner only (reported in the footer) and is NA for every other candidate – the effect (from the search) is carried for all, but the per-candidate consistency refits are skipped. Selection is identical to evaluating every candidate. "maxSG" Sort by $(-N, -Pcons, K)$; pick top. "minSG" Sort by $(N, -Pcons, K)$; pick top. "hrMaxSG" (or "effMaxSG") Among candidates with effect size within effect_neighborhood of the maximum, pick the one with the largest N . "hrMinSG" (or "effMinSG") Among candidates with effect size within effect_neighborhood of the maximum, pick the one with the smallest N . Default: "hr". The "eff*" forms are aliases for the "hr*" forms and read more naturally in GLM contexts (continuous MD, binary OR/RR/RD, count IRR). Both vocabularies produce identical results.
selection_rule	Character. Rule defining the candidate inclusion set for "hrMaxSG" / "hrMinSG". One of "neighborhood" (default; current behaviour), "pareto" (2D Pareto-non-dominated set in (effect, N)), or "both" (intersection of the two). Forwarded to sort_subgroups_preview (pre-consistency triage) and sort_subgroups (post-consistency selection). Must be "neighborhood" for other sg_focus values.
effect_neighborhood	Numeric in $[0, 1)$. Relative tolerance for "hrMaxSG" and "hrMinSG". A candidate is in the <i>effect-size neighborhood</i> iff its (natural-scale) effect is at least $(1 - \text{effect_neighborhood}) * \max(\text{effect})$. Default 0.10 (i.e., within 10% For GLM ratio measures (OR, IRR), the neighborhood test is applied on the natural scale (after exponentiation of hr). Ignored when sg_focus is "hr", "maxSG", or "minSG". When selection_rule = "pareto", must be left at its default (the rule does not consult it).
stop_Kgroups	Integer. Maximum number of candidates to evaluate. When the ranked pool exceeds this cap it is truncated and a warning naming both counts and the sg_focus ordering is emitted. Default: 200
stop_threshold	Numeric in $[0, 1]$ or NULL. When a candidate subgroup's consistency probability (Pcons) meets or exceeds this threshold, evaluation stops early – remaining candidates are skipped. Set to NULL to disable early stopping and evaluate all candidates up to stop_Kgroups. Default: NULL. Note: Values > 1.0 are not permitted. To disable early stopping, use stop_threshold = NULL, not a value above 1. Interaction with sg_focus:

"hr", "maxSG", "minSG" Early stopping is valid because candidates are sorted by a single criterion. The first candidate passing the threshold is optimal under that criterion.

"hrMaxSG", "hrMinSG" Should generally be NULL, because these compound criteria require comparing HR *and* size across all candidates. forestsearch() automatically resets to NULL with a warning for these.

For parallel execution, early stopping is checked after each batch completes, so some additional candidates beyond the first meeting the threshold may be evaluated. Use a smaller batch_size in parallel_args for finer-grained early stopping.

show_candidate_summary

Logical. If TRUE, prints two diagnostic tables: a **pre-consistency** candidate preview (all top-stop_Kgroups candidates entering consistency evaluation, with Frontier and InBand flags) and a **post-consistency** summary (passing candidates only, with Pcons, Frontier, InBand, and Selected flags). The two views together make the rule's filter visible end-to-end. Pcons is shown only for passing candidates (the consistency evaluator returns NULL for non-passing ones). Default: FALSE.

max_print

Integer. Maximum number of candidate rows printed in each show_candidate_summary table (forwarded to the internal print_candidate_summary()). Candidates are sorted selection-first, so the selected subgroup is always shown; any remaining candidates beyond max_print are collapsed into a trailing "... N more candidate(s) not shown" line. Affects console output only, not the returned object. Default: 10.

pconsistency.digits

Integer. Number of decimal places to which the consistency proportion is rounded **before** it is compared with pconsistency.threshold. The rounded value decides admission and is the value the selection sort orders on, so this setting affects which subgroups are admitted, not only what is displayed. At the default of 2, a threshold of p^* admits any candidate whose unrounded proportion is at or above the largest 0.01 grid point strictly below p^* plus half a step (0.005) – so $p^* = 0.99$ admits from about 0.985. Raising the setting makes admission finer-grained and produces fewer exact ties for the selection rule to break. Under consistency_method = "resample" the proportion is a continuous closed-form quantity, so the rounding is a deliberate coarsening rather than a formatting step. Default: 2

seed

Integer. Random seed for reproducible consistency splits. Default: 8316951. Set to NULL for non-reproducible random splits. The seed is used both for sequential execution (via set.seed()) and parallel execution (via future.seed).

checking

Logical. Enable additional validation checks. Default: FALSE

use_twostage

Logical. Use two-stage adaptive algorithm. Default: FALSE

twostage_args

List. Parameters for two-stage algorithm:

n.splits.screen Splits for Stage 1 screening. Default: 30

screen.threshold Consistency threshold for Stage 1. Default: auto

batch.size Splits per batch in Stage 2. Default: 20

conf.level Confidence level for early stopping. Default: 0.95

	min.valid.screen Minimum valid Stage 1 splits. Default: 10
parallel_args	List. Parallel processing configuration: plan Future plan: "multisession", "multicore", "callr", or "sequential". A sequential plan runs the consistency screen as a plain in-process loop regardless of workers: no future backend is set up and no per-batch globals resolution is paid. A parallel plan with workers = 1 is honoured as parallel (one-worker batched backend); pass plan = "sequential" to run the plain loop. Results under consistency_method = "resample" are identical on every path (the closed form consumes no RNG). Under "split" the plain loop draws from a single RNG stream seeded once, whereas the batched backend gives each candidate its own L'Ecuyer-CMRG stream indexed by global position, so a sequential plan's p.consistency values differ from a parallel plan's within Monte-Carlo noise at the stated fs.splits; invariance to worker count holds within any plan. workers Number of parallel workers on a parallel plan. batch_size Number of candidates to evaluate per batch on a genuinely parallel plan – the knob for the per-batch overhead described above. Smaller values provide finer-grained early stopping but may increase overhead. Default: When stop_threshold is set and sg_focus is "hr" or "minSG", defaults to 1 (stop immediately when first candidate passes). For other sg_focus values with stop_threshold, defaults to min(workers, n_candidates/4). When stop_threshold is NULL, defaults to workers*2 for efficiency. show_message Print parallel config messages
estimator_fn	Closure or NULL. Effect-estimator closure from make_effect_estimator for GLM outcomes. Default NULL (uses Cox model).
consistency_threshold	Numeric or NULL. Threshold for GLM consistency evaluation. Default NULL (uses hr.consistency).
effect_label	Character. Column label for the effect measure in diagnostic output (e.g., candidate subgroup table). Default "HR". Set automatically by forestsearch to the resolved effect_measure (e.g., "IRR", "OR") for GLM outcomes.
effect_log_scale	Logical. If TRUE, the HR column in hr.subgroups stores log-scale values (log-OR, log-IRR) and will be exponentiated for display in diagnostic output. Default FALSE (Cox stores natural-scale HR).
adjust_covariates	Character vector or NULL. Additional Cox model terms used when scoring survival subgroups during consistency evaluation, e.g. "strata(x1)" for a stratified model or "x2" for linear adjustment. Referenced columns must be present in df. Ignored on the GLM path. Default NULL (treatment-only, unadjusted).
consistency_method	Character. "resample" (default) for the multiplier (influence-function) approximation via consistency_resample , or "split" – the retained fallback – for the literal repeated 50/50 split-and-refit calculation, which costs fs.splits * 2 refits per candidate evaluation. On the GLM path "resample" requires

a `glm_resample_spec`; without one (or when `consistency_resample` is unavailable) it falls back to "split" with a warning. `forestsearch` builds that spec automatically; a direct caller on a GLM outcome must supply it.

`glm_resample_spec`

List or NULL. GLM resampling specification threaded from `forestsearch` when `consistency_method = "resample"` on a GLM outcome: components `outcome_type`, `effect_measure`, `treat.name`, `outcome.name`, `offset.name`, `adjust_covariates`, `adverse_outcome`, and `comparison_threshold` (the consistency threshold already on the comparison scale, i.e. `log(threshold)` for ratio measures). NULL on the survival path.

Value

A list containing:

out_sg Selected subgroup results. When non-NULL, contains `result` (sorted candidate table; top row is the selected subgroup) and `pareto_frontier` (data.table of non-dominated candidates on (effect, N), both maximized – a post-hoc diagnostic, not used for selection).

sg_focus Selection criterion used

df_flag Data frame with treatment recommendations

sg.harm Subgroup definition labels – character vector of factor-level cut names (e.g., `c("z1.1", "z2.1")`), of length equal to the number of cuts defining the subgroup. NULL if no subgroup was identified.

sg.harm.id Per-subject subgroup-membership indicator – integer vector of length `nrow(df)` with 1 if subject i is in the identified subgroup and 0 otherwise. NULL if no subgroup was identified. **Not** a character vector of cut expressions; see the *Field naming collision* section below.

algorithm "twostage" or "fixed"

n_candidates_evaluated Number of candidates actually evaluated

n_candidates_total Total candidates available

n_passed Number meeting consistency threshold

early_stop_triggered Logical indicating if early stop occurred

early_stop_candidate Index of candidate triggering early stop

stop_threshold Threshold used for early stopping

seed Random seed used for reproducibility (NULL if not set)

Field naming collision with GRF results

The field name `sg.harm.id` has **different semantics** on this object versus on the result objects returned by `grf.subg.harm.glm` and `grf.subg.harm.survival`:

Object	<code>sg.harm.id</code> contains	Length / type
<code>subgroup.consistency()</code> result (this function)	per-subject 0/1 membership indicator	integer, length <code>nrow(df)</code>
<code>grf.subg.harm.glm()</code> result	character vector of cut expressions	character, length = depth of selection
<code>grf.subg.harm.survival()</code> result	character vector of cut expressions	character, length = depth of selection

Practical consequence. Do not paste `sg.harm.id` with `paste(..., collapse = "& ")` to print "the identified subgroup" without first checking object class. For subgroup labels from a [forestsearch](#) or `subgroup.consistency()` result, use `sg.harm` (character vector of cut names) – the FS main result exposes `sg.harm` at the top level.

This naming collision is a documented CRAN-stable API for v0.1.x and v0.2.x; it is expected to be resolved via a deprecation cycle in a future minor release.

Examples

```
## Not run:
# Standard evaluation
result <- subgroup.consistency(
  df = trial_data,
  hr.subgroups = candidates,
  sg_focus = "hr",
  n.splits = 400,
  parallel_args = list(plan = "multisession", workers = 6)
)

# Show top 10 candidates before evaluation
result <- subgroup.consistency(
  df = trial_data,
  hr.subgroups = candidates,
  sg_focus = "hr",
  show_candidate_summary = TRUE, # Post-consistency summary
  n.splits = 400
)

# With early stopping and custom batch size
result <- subgroup.consistency(
  df = trial_data,
  hr.subgroups = candidates,
  sg_focus = "hr",
  stop_threshold = 0.95,
  show_candidate_summary = TRUE,
  parallel_args = list(
    plan = "multisession",
    workers = 6,
    batch_size = 2 # Check early stopping after every 2 candidates
  )
)

## End(Not run)
```

subgroup.search	<i>Subgroup Search for Treatment Effect Heterogeneity (Improved, Parallelized)</i>
-----------------	--

Description

Searches for subgroups with treatment effect heterogeneity using combinations of candidate factors. Evaluates subgroups for minimum prevalence, event counts, and hazard ratio threshold. Parallelizes the main search loop.

Usage

```
subgroup.search(  
  Y,  
  Event,  
  Treat,  
  ID = NULL,  
  Z,  
  n.min = 30,  
  d0.min = 15,  
  d1.min = 15,  
  hr.threshold = 1,  
  max.minutes = 30,  
  minp = 0.05,  
  rmin = 5,  
  details = FALSE,  
  maxk = 2,  
  parallel_workers = parallel::detectCores(),  
  estimator_fn = NULL,  
  df_analysis = NULL,  
  effect_threshold = NULL,  
  effect_measure = NULL,  
  outcome_type = NULL,  
  adjust_covariates = NULL,  
  disable_effect_floor = FALSE  
)
```

Arguments

Y	Numeric vector of outcome (e.g., time-to-event).
Event	Numeric vector of event indicators (0/1).
Treat	Numeric vector of treatment group indicators (0/1).
ID	Optional vector of subject IDs.
Z	Matrix or data frame of candidate subgroup factors (binary indicators).
n.min	Integer. Minimum subgroup size.

d0.min	Integer. Minimum per-arm filter. For survival and binary outcomes: minimum number of events in the control arm within the candidate subgroup. Ignored for continuous outcomes (only n.min applies).
d1.min	Integer. Same as d0.min for the treatment arm.
hr.threshold	Numeric. c1, the screening threshold on a candidate subgroup's own effect : a candidate is retained only if its fitted effect exceeds this value. The scale is the one the caller compares on, and it differs by path. On the survival path effect_threshold is NULL, so this value reaches the comparison unchanged and is compared against the fitted hazard ratio on the natural scale (fit_cox_for_subgroup() returns exp(coef)): 1.25 means HR >= 1.25, not log(1.25). On the GLM paths the value arrives already on the comparison scale that forestsearch() resolved from the effect measure – link scale for the ratio measures (OR, RR, IRR), identity for RD, IRD and MD – and is compared against the estimator's estimate on that same scale. This is forestsearch()'s effect_threshold/hr.threshold. It is not the per-split threshold (c2, hr.consistency) and not the consistency rate (p*, pconsistency.threshold); neither is applied here – this function performs the screening stage only.
max.minutes	Numeric. Currently inert; scheduled for deprecation in v0.3.0 . Previously intended as a wall-clock budget for the combination search, it is threaded to search_combinations_parallel() and no path consults it, so it has no effect on behaviour. Search scope is governed by maxk and the candidate-factor controls. Retained because it is part of a public signature; this description now matches forestsearch's, which has documented it as inert for some time.
minp	Numeric. Minimum prevalence rate for each factor.
rmin	Integer. Minimum required reduction in sample size when adding a factor.
details	Logical. Print details during execution.
maxk	Integer. Maximum number of factors in a subgroup.
parallel_workers	Integer. Number of parallel workers (default: all available cores).
estimator_fn	Closure or NULL. Effect-estimator closure from make_effect_estimator for GLM outcomes. Default NULL (uses Cox model).
df_analysis	Data frame or NULL. Full analysis dataset, passed to the GLM estimator for within-subgroup evaluation.
effect_threshold	Numeric or NULL. GLM effect threshold for subgroup selection. Default NULL (uses hr.threshold).
effect_measure	Character or NULL. Effect measure name (e.g., "OR", "HR", "RD"). Used for display labels in the filtering summary. Default NULL.
outcome_type	Character or NULL. One of "survival", "binary", "continuous", "count". Controls d0.min/d1.min interpretation: events per arm for binary/survival, ignored for continuous. Default NULL (inferred from estimator_fn presence).
adjust_covariates	Character vector or NULL. Additional Cox model terms (e.g. "strata(x1)" or "x2") used to adjust the candidate-search Cox scorer on the survival path. Referenced columns are sourced from df_analysis. Ignored on the GLM path. Default NULL (treatment-only).

disable_effect_floor
Logical. When TRUE, the Status-6 effect threshold (hr.threshold) is not applied, so the full estimable candidate family is retained regardless of effect size. Set by `forestsearch` when `sg_focus = "maxeff"` so the effect maximiser ranges over an unconditional family. Default FALSE.

Value

List with found subgroups, maximum HR, search time, configuration info, and filtering statistics.

Examples

```
## Not run:
library(survival)
df <- survival::gbsg
df$grade3 <- as.integer(df$grade == "3")
Z <- df[, c("age", "meno", "size", "grade3", "nodes", "pgr", "er")]
res <- subgroup.search(Y = df$rfstime, Event = df$status,
  Treat = df$hormon, Z = Z,
  hr.threshold = 1.25)

## End(Not run)
```

subgroup_cox	<i>Cox per-subgroup fit function</i>
--------------	--------------------------------------

Description

Returns a function data -> c(HR, UB) fitting formula by `survival::coxph()` and extracting the first row of `summary(fit)$conf.int`: the hazard-ratio point estimate ("exp(coef)") and the upper 95% bound ("upper .95"). Estimation failure or an empty coefficient table returns c(NA, NA). This reproduces the vignettes' `cox_sR()` exactly when called with `survival::Surv(y_sim, event_sim) ~ treat_sim + survival::strata(grade)`.

Usage

```
subgroup_cox(formula, lean = TRUE)
```

Arguments

- formula A model formula for `survival::coxph()`, evaluated against each subgroup's data.
- lean Re-home the formula environment as described above (default TRUE).

Details

Custom fitters may be passed to `run_subgroup_sims()` directly: any function of a data frame returning a length-2 numeric (estimate, upper bound) is accepted. Fitters run on parallel workers, so they should reference non-base functions by `pkg::fun` or rely on packages listed in `future_packages`.

Value

A function of one argument (data) returning `c(hr, ub)`; carries the formula in `attr(, "formula")`.

Formula environment (lean)

A formula captures the environment where it was created. In a script or a knitr chunk that is the global environment, and future's globals machinery gives formulas special treatment: a fit whose formula points at a populated environment imposes a multi-second per-future cost on parallel runs (measured ~2-4 s x ~116 futures in the Phase 2 timing diagnostic), independent of the serialized size. With `lean = TRUE` (default) the formula is re-homed into a fresh, empty environment parented on the survival namespace, cutting the costly chain while keeping `survival::Surv()`, `survival::strata()`, and base/stats symbols resolvable. Consequence: the formula must reference only data columns and functions reachable from the survival namespace (use `pkg::` prefixes for anything else). Set `lean = FALSE` to retain the calling environment – restoring arbitrary-symbol resolution at that parallel cost. Independently of `lean`, the returned closure is stripped of source references and given a minimal environment holding only the formula (keep-source installs otherwise attach ~250 KB of package-source baggage to every worker shipment).

Examples

```
fit <- subgroup_cox(survival::Surv(y_sim, event_sim) ~ treat_sim)
```

subgroup_glm

GLM per-subgroup fit function

Description

Returns a function data → `c(estimate, UB)` for non-survival outcomes by delegating estimation to the package's native effect-estimator layer: the closure from `make_effect_estimator()` (the same code the `forestsearch()` search, bootstrap, and consistency loops use), including the bit-identical unadjusted fast path (`.make_lm_estimator_fast()`) under the same routing predicate `forestsearch()` applies. The estimator's internal-scale estimate/se are mapped to the wrapper contract as `upper = estimate + qnorm(1 - (1 - level)/2) * se`, back-transformed (`exp()`) for ratio measures – the package's own display convention – so ratio-measure matrices land on the natural scale exactly as `subgroup_cox()`'s `exp(coef) / upper .95` do. Estimation failure, non-convergence, NA estimate/SE (e.g. a single-arm subgroup, where the treatment coefficient is aliased), or a non-finite result (e.g. a zero-residual-df fit) returns `c(NA, NA)`.

Usage

```
subgroup_glm(
  outcome.name = "y_sim",
  treat.name = "treat_sim",
  outcome_type = "continuous",
  effect_measure = NULL,
  adverse_outcome = TRUE,
  robust_se = TRUE,
```

```

offset.name = NULL,
adjust_covariates = NULL,
level = 0.95,
lean = TRUE
)

```

Arguments

outcome.name, treat.name	Column names of the outcome and the 0/1 treatment indicator; defaults match <code>simulate_from_glm_dgm()</code> 's <code>y_sim / treat_sim</code> .
outcome_type	"continuous", "binary", or "count" (all validated; count requires <code>offset.name</code>).
effect_measure	NULL resolves to the wrapper default for the outcome type: "MD" (continuous), "OR" (binary – the <code>generate_glm_dgm()</code> default and the calibration target; note that <code>make_effect_estimator()</code> 's own binary default is "RD"), or "IRR" (count – all layers agree). Binary also accepts "RD"; count also accepts "IRD"; see the supported-settings section for the gated measures.
adverse_outcome	Higher outcome = harm (TRUE, default). See the direction convention above.
robust_se, offset.name, adjust_covariates	Passed to <code>make_effect_estimator()</code> . <code>robust_se</code> is consulted only where the estimator layer uses sandwich-style SEs (the count IRR path, the RR modified-Poisson fallback, and other rate estimators); the unadjusted OR path, the RD tiers, the unadjusted IRD delta chain, and MD use model-based/analytic SEs, so the flag is a no-op for those measures (asserted in the phase gates). <code>offset.name</code> is REQUIRED for <code>outcome_type = "count"</code> : it names a strictly positive exposure / follow-up time column in the analysis data (the estimator adds an internal <code>.log_offset</code> column, clamping non-positive values at <code>.Machine\$double.eps</code>); for plain counts supply a unit column of 1s – $\log(1) = 0$ reproduces the no-offset Poisson model exactly. Supplying <code>adjust_covariates</code> disables the unadjusted fast path (matching <code>forestsearch()</code> 's routing).
level	Confidence level for the upper bound (default 0.95).
lean	Re-home the provenance formula's environment (default TRUE).

Value

A function of one argument (`data`) returning `c(estimate, ub)`; carries the provenance formula in `attr(, "formula")` and scale metadata in `attr(, "effect")` – a list with `measure`, `outcome_type`, `log_scale`, `null_value`, `adverse_outcome`, `est_label`, `ub_label`, `est_thresholds`, `ub_thresholds`, `level` – which `run_subgroup_sims()` stamps onto GLM results for scale-aware summaries and plots. A custom fitter can opt into the same behavior by attaching an "effect" attribute of this shape.

Direction convention

Estimates follow the estimator layer's harm-positive normalization: with `adverse_outcome = FALSE` (higher outcome = better) the outcome is negated internally so that `estimate > 0` consistently indicates treatment harm, aligning with the survival convention where $HR > 1$ = harm. Consequently

the null value is always 0 (identity measures) or 1 (ratio measures), whatever the raw outcome's direction.

Supported settings

Phase 4.5 validates `outcome_type = "continuous"` (MD), `outcome_type = "binary"` ("OR" – the binary default – and "RD"), and `outcome_type = "count"` ("IRR" – the count default – and "IRD"; `offset.name` is required, see below). Binary "RR" (estimator-supported; wrapper parity pending) still stops with an informative error; the estimator layer (`make_effect_estimator()`) already supports it and the gate lifts with its parity phase – no interface change.

Serialization (lean)

The returned closure is stripped of source references and given a minimal environment (the estimator closure and two scalars), and the estimator closure itself is rebuilt onto a fresh environment holding only its forced argument values (parented on the package namespace, which serializes by reference) with its own source references stripped. This severs shared `srcfile` environments (keep-source installs) and the constructor's frame chain from the serialized graph, so the fitter ships to parallel workers in the low single-digit KB regardless of session state. The provenance formula attached in `attr(, "formula")` (built from the column names, e.g. `y_sim ~ treat_sim`) is re-homed into an empty stats-parented environment when `lean = TRUE`, keeping the parallel-shipment guard in `run_subgroup_sims()` meaningful.

See Also

`subgroup_cox()`, `run_subgroup_sims()`, `make_effect_estimator()`, `generate_glm_dgm()`

Examples

```
## Not run:
fit <- subgroup_glm()                # y_sim ~ treat_sim, MD
fit(simulate_from_glm_dgm(dgm, n = 500, seed = 1))

# Binary (Phase 4.4): OR by default, RD available
fit_or <- subgroup_glm(outcome_type = "binary")
fit_rd <- subgroup_glm(outcome_type = "binary", effect_measure = "RD")

# Count (Phase 4.5): IRR by default, IRD available; offset required
fit_irr <- subgroup_glm(outcome_type = "count", offset.name = "t_exp")
fit_ird <- subgroup_glm(outcome_type = "count", offset.name = "t_exp",
                        effect_measure = "IRD")

## End(Not run)
```

suggest_thresholds	<i>Suggest Screening and Consistency Thresholds</i>
--------------------	---

Description

Recommends (c1, c2) threshold pairs for ForestSearch based on the corrected null approximation. Uses the per-subgroup detection probability and an L_{eff} multiplicity correction to find the smallest c1 (screening threshold) such that the corrected FPR is at most fpr_target.

Usage

```
suggest_thresholds(
  d_eff,
  N,
  fpr_target = 0.1,
  n_min = 60,
  L_eff_C = 1,
  L_eff_alpha = 0,
  c1_range = c(1.05, 2.5),
  c1_step = 0.05,
  c2_candidates = seq(0.9, 2, by = 0.05),
  effect_scale = c("ratio", "difference")
)
```

Arguments

d_eff	Numeric. Effective information in the expected harm subgroup. Use d_eff_survival , d_eff_binary , d_eff_count , or d_eff_continuous .
N	Integer. Total sample size.
fpr_target	Numeric. Maximum acceptable procedure-level FPR. Default: 0.10.
n_min	Integer. Minimum subgroup size. Default: 60.
L_eff_C	Numeric. Calibration constant for $L_{\text{eff}} = C(N/n_{\text{min}})^{\alpha}$. Default: 1.0 (conservative; see Details).
L_eff_alpha	Numeric. Calibration exponent. Default: 0.0 (no N-dependence; see Details).
c1_range	Numeric vector of length 2. Search range for c1. Default: c(1.05, 2.50).
c1_step	Numeric. Grid step for c1. Default: 0.05.
c2_candidates	Numeric vector. Candidate c2 values to evaluate for each c1. Default: seq(0.90, 2.0, by = 0.05).
effect_scale	Character. "ratio" (default) or "difference".

Details

The corrected FPR is:

$$\text{FPR}_{\text{corr}} = 1 - (1 - P_1)^{L_{\text{eff}}}$$

where $P_1 = \text{compute_detection_probability_glm}(1.0, d_{\text{eff}}, c1, c2)$ and $L_{\text{eff}} = C(N/n_{\text{min}})^\alpha$.

Default L_{eff} values ($C=1$, $\alpha=0$) assume $L_{\text{eff}} = 1$, which gives the per-subgroup approximation with no multiplicity correction. This is appropriate when:

- No calibration data is available
- The analysis has few confounders and small N

For more accurate suggestions, calibrate L_{eff} from an H0 simulation using [calibrate_L_eff](#) or [run_null_calibration](#).

Reference calibrations (from the forestsearch GLM extension analysis documents):

Binary, 4 confounders $C = 0.220$, $\alpha = 1.298$ (binary_threshold_calibration.qmd, 5000 sims)

GBSG survival, 7 confounders $C = 0.029$, $\alpha = 0.882$ (cox_vs_glm_approximation.qmd, 500 sims per N)

Value

A data.frame with columns:

c1 Screening threshold.

min_c2 Minimum consistency threshold achieving the target.

FPR_corr Corrected FPR at this (c1, min_c2).

P1 Per-subgroup detection probability.

L_eff Effective number of candidates at this N.

Sorted by c1 ascending. The first row is the smallest c1 that achieves the target FPR.

See Also

[compute_detection_probability_glm](#), [calibrate_L_eff](#), [predict_fpr_corrected](#)

Examples

```
# Survival: paper defaults should be recovered
d <- d_eff_survival(n_sg = 60, prop_cens = 0.45)
suggest_thresholds(d_eff = d, N = 700)

# Binary: needs higher thresholds
d <- d_eff_binary(n_sg = 150, p_event = 0.30)
suggest_thresholds(d_eff = d, N = 500,
  L_eff_C = 0.220, L_eff_alpha = 1.298)

# Use calibration object from run_null_calibration()
# cal <- run_null_calibration(...)
# suggest_thresholds(d_eff = d, N = 700,
```

```
#               L_eff_C = cal$C, L_eff_alpha = cal$alpha)
```

```
summarize_bootstrap_events
```

Summarize Bootstrap Event Counts

Description

Provides summary statistics for event counts across bootstrap iterations, helping assess the reliability of HR estimates when events are sparse.

Usage

```
summarize_bootstrap_events(boot_results, threshold = 5)
```

Arguments

boot_results List. Output from `forestsearch_bootstrap_dofuture()`
threshold Integer. Minimum event threshold for flagging low counts (default: 5)

Details

This function summarizes event counts in four scenarios:

1. ORIGINAL subgroup H evaluated on BOOTSTRAP samples
2. ORIGINAL subgroup Hc evaluated on BOOTSTRAP samples
3. NEW subgroup H* (found in bootstrap) evaluated on ORIGINAL data
4. NEW subgroup Hc* (found in bootstrap) evaluated on ORIGINAL data

Low event counts (below threshold) can lead to unstable HR estimates. This summary helps identify potential issues with sparse events.

Value

Invisibly returns a list with summary statistics:

threshold The event threshold used

nb_boots Total number of bootstrap iterations

n_successful Number of iterations that found a new subgroup

original_H List with low event counts for original H on bootstrap samples

original_Hc List with low event counts for original Hc on bootstrap samples

new_Hstar List with low event counts for new H* on original data

new_Hcstar List with low event counts for new Hc* on original data

Examples

```
## Not run:
# After running bootstrap analysis
summarize_bootstrap_events(fs_bc, threshold = 10)

## End(Not run)
```

summarize_bootstrap_results
<i>Enhanced Bootstrap Results Summary</i>

Description

Creates comprehensive output including formatted table with subgroup footnote, diagnostic plots, bootstrap quality metrics, and detailed timing analysis.

Usage

```
summarize_bootstrap_results(
  sgharm,
  boot_results,
  create_plots = FALSE,
  est.scale = "hr",
  digits = 2
)
```

Arguments

sgharm	The selected subgroup object from forestsearch results. Can be: • Character vector of factor definitions (e.g., c("{age>=50}", "{nodes>=3}")) • List with sgharm element containing factor definitions • List with sg.harm_label element (human-readable labels)
boot_results	List. Output from forestsearch_bootstrap_dofuture()
create_plots	Logical. Generate diagnostic plots (default: FALSE)
est.scale	Character. "hr" or "1/hr" for effect scale
digits	Integer or NULL. Decimal places for display formatting of numeric columns in the rendered table (per-arm summaries, Diff, effect estimate CI, bias-corrected CI). Default 2. When set, the formatted strings in boot_results\$FSsg_tab are parsed back to numeric and re-formatted at the requested precision; this works reliably as long as the construction-time precision (default 4 decimals; controlled by digits in forestsearch_bootstrap_dofuture) is at least as high as digits requested here. Set digits = NULL to display the construction-time strings unchanged. The n and n1 columns are not reformatted – their count-with-percent format is preserved as constructed.

Details

The table output includes a footnote displaying the identified subgroup definition, analogous to the `tab_estimates` table from [sg_tables](#). This is achieved by extracting the subgroup definition from `sg_harm` and passing it to [format_bootstrap_table](#).

Value

- List with components:
- table** `gt` table with treatment effects and subgroup footnote
- diagnostics** List of bootstrap quality metrics
- diagnostics_table_gt** `gt` table of diagnostics
- plots** List of `ggplot2` diagnostic plots (if `create_plots=TRUE`)
- timing** List of timing analysis (if timing data available)
- subgroup_summary** List from `summarize_bootstrap_subgroups()`

See Also

[format_bootstrap_table](#) for table creation [sg_tables](#) for analogous main analysis tables [summarize_bootstrap_subgroups](#) for subgroup stability analysis

Examples

```
## Not run:
library(survival)
df <- survival::gbsg
df$grade3 <- as.integer(df$grade == "3")
fs <- forestsearch(df,
  confounders.name = c("age", "meno", "size", "grade3", "nodes", "pgr", "er"),
  outcome.name = "rfstime", treat.name = "hormon", event.name = "status")
fs_bc <- forestsearch_bootstrap_dofuture(fs, nb_boots = 100)
summaries <- summarize_bootstrap_results(fs$sg_harm, fs_bc)
summaries$table

## End(Not run)
```

summarize_bootstrap_subgroups
<i>Summarize Bootstrap Subgroup Analysis Results</i>

Description

Comprehensive tabulation of bootstrap subgroup identification results including basic statistics, factor frequencies, consistency distributions, GRF-cut frequencies, and agreement with the original analysis subgroup. Returns `raw_data.table` tabulations suitable for custom post-processing, export, or plotting; for the full formatted pipeline (`gt` tables, diagnostics, plots) call [summarize_bootstrap_results](#) instead.

Usage

```
summarize_bootstrap_subgroups(results, nb_boots, original_sg = NULL, maxk = 2)
```

Arguments

results	Data.table or data.frame. Bootstrap results with subgroup characteristics including columns like Pcons, hr_sg, N_sg, K_sg, and M.1-M.k
nb_boots	Integer. Total number of bootstrap iterations
original_sg	Character vector. Original subgroup definition from main analysis (e.g., c("{age>=50}", "{nodes>=3}")) for a 2-factor subgroup)
maxk	Integer. Maximum number of factors allowed in subgroup definition

Value

List with summary components:

- basic_stats** Data.table of summary statistics
- consistency_dist** Data.table of Pcons distribution by bins
- size_dist** Data.table of subgroup size distribution
- factor_freq** Data.table of factor frequencies by position
- agreement** Data.table of subgroup definition agreement counts
- factor_presence** Data.table of base factor presence counts
- factor_presence_specific** Data.table of specific factor definitions
- grf_cut_freq** Data.table of GRF policy-tree cut frequencies across all bootstrap iterations (not just successful ones). Columns: Rank, grf_cut, N, Percent. NULL when results lacks the grf_cuts_b column (pre-Phase-C objects). Unlike subgroup-related tables, this tabulation is NOT restricted to iterations that identified a subgroup – it spans all bootstraps so that the analyst can distinguish "GRF produced no cut" from "GRF produced a cut that consistency rejected".
- original_agreement** Data.table comparing to original analysis subgroup
- n_found** Integer. Number of successful iterations
- pct_found** Numeric. Percentage of successful iterations

See Also

[summarize_bootstrap_results](#) for the full formatted summary pipeline (gt tables plus diagnostics and plots). [forestsearch_bootstrap_dofuture](#) to run the bootstrap analysis that produces the results object consumed here.

Examples

```
## Not run:
# Run bootstrap analysis
boot_out <- forestsearch_bootstrap_dofuture(fs.est, nb_boots = 500)

# Raw tabulations for custom post-processing
```

```

subg <- summarize_bootstrap_subgroups(
  results      = boot_out$results,
  nb_boots     = 500,
  original_sg  = fs.est$sg.harm,
  maxk        = 2L
)

# Inspect the GRF cut frequencies across all bootstraps
subg$grf_cut_freq

# Export factor frequencies to CSV for an external report
data.table::fwrite(subg$factor_freq, "factor_freq.csv")

## End(Not run)

```

summarize_cv_results *Summarise ForestSearch Cross-Validation Diagnostics*

Description

Post-hoc diagnostic summary of [forestsearch_tenfold](#) output, aggregating the sim x fold grid captured in `fold_summary` into publication-ready gt tables. Parallel to [summarize_bootstrap_results](#) but focused on cross-validation stability and GRF candidate-discovery behaviour.

Usage

```

summarize_cv_results(
  cv_output,
  original_sg = NULL,
  original_grf_cuts = NULL,
  create_plots = FALSE,
  top_n = 15L
)

```

Arguments

<code>cv_output</code>	List of class "fs_tenfold", as returned by forestsearch_tenfold . Must contain a non-empty <code>fold_summary</code> data frame.
<code>original_sg</code>	Character vector or NULL. Original full-data subgroup definition (e.g., <code>fs\$sg.harm</code>). When supplied, <code>original_agreement</code> reports exact and partial-factor match rates against this definition.
<code>original_grf_cuts</code>	Character vector or NULL. Original full-data <code>grf_cuts</code> (e.g., <code>fs\$grf_cuts</code>). When supplied alongside a <code>grf_cuts</code> column in <code>fold_summary</code> , <code>original_agreement</code> also reports GRF-cut match rate.
<code>create_plots</code>	Logical. If TRUE and ggplot2 is installed, produce bar-chart diagnostics (default: FALSE).

top_n Integer. Maximum number of detail rows to retain in frequency tables (default: 15L). If the true number of distinct values exceeds top_n, a single "(N others)" row aggregates the tail so that n_folds totals remain honest. Set to a large value (e.g., Inf) to disable truncation entirely. Plots use min(top_n, 10L).

Value

An object of class "fs_cv_summary", a list with components:

identification_summary, **grf_cut_summary**, **cut_vs_subgroup_xtab**, **no_subgroup_decomposition**, **pconsistency_gt** tables (or data.frame fallback if **gt** is unavailable). Some may be NULL depending on input columns and optional arguments.

plots NULL or a list of ggplot objects.

data List of raw data.frame tabulations underlying the tables (identification, grf_cuts, cut_vs_subgroup, no_subgroup, pconsistency, fold_numeric_summary, original_agreement).

n_sims, **n_folds**, **total_pairs** Grid dimensions.

has_grf_cuts, **has_pconsistency** Logical. Whether the corresponding fold_summary columns were available. The fold_numeric_summary slot adapts automatically to whichever numeric diagnostic columns are present and does not have a dedicated flag.

Diagnostic components

The returned object includes (as gt tables by default):

- **identification_summary**: top identified subgroups across all sim x fold pairs, with counts and percentages.
- **grf_cut_summary**: top raw grf_cuts strings across training sets – how often GRF returned each output (and how often it returned none). NULL if fold_summary\$grf_cuts is not available (e.g., objects from **forestsearch** prior to the GLM-extension branch).
- **cut_vs_subgroup_xtab**: cross-tabulation of unique grf_cuts strings against identified subgroups. Reveals whether GRF's discovery aligns with the pipeline's final selection.
- **no_subgroup_decomposition**: tabulation of folds that identified no subgroup, broken into "GRF returned no cut" vs "GRF returned a cut but consistency rejected it".
- **pconsistency_distribution**: binned distribution of the consistency probability (Pcons) achieved by the identified subgroup on each fold, restricted to folds that identified a subgroup; includes median and IQR. Reveals whether identifying folds are scraping pconsistency.threshold (e.g., Pcons just above 0.90) or are well above it. NULL if fold_summary\$pconsistency is not available. Rejected- candidate Pcons values are not surfaced – only the selected candidate's Pcons is captured in fold_summary.
- **fold_numeric_summary**: tidy median / IQR / range table with one row per numeric diagnostic column present in fold_summary (n_test, pconsistency, training_fs_hr, n_candidates_evaluated). Absent columns simply don't produce rows, so the table adapts to pre-feature-branch objects without error. Training-fold subgroup effects are on the effect measure's natural scale (HR for survival; OR, RR, IRR, RD, IRD, or MD for GLM, exponentiated at capture for ratio measures so the column is always on natural scale regardless of outcome type); they are in-sample estimates (optimistically biased) and are surfaced for diagnostic comparison across folds only.

- `original_agreement`: if `original_sg` and/or `original_grf_cuts` are supplied, fraction of folds matching the original full-data analysis (exact subgroup, partial shared factor, GRF-cut match, both).
- `metrics_table`: formatted version of `cv_output$sens_summary` and `cv_output$find_summary` (delegates to [cv_metrics_tables](#)).
- `plots`: if `create_plots = TRUE`, bar charts of top identified subgroups and top GRF cuts.

Raw data.frame versions of each tabulation are available at `$data$*` for custom post-processing.

Backward compatibility

The GRF-dependent slots (`grf_cut_summary`, `cut_vs_subgroup_xtab`, `no_subgroup_decomposition`) require a `grf_cuts` column in `fold_summary`; the `pconsistency_distribution` slot requires a `pconsistency` column. Both were added on the feature/glm-extension branch (`grf_cuts` first, `pconsistency` later). Older objects are detected and the affected slots return `NULL` without error. Metadata fields `has_grf_cuts` and `has_pconsistency` record which columns were present.

See Also

[forestsearch_tenfold](#) for running repeated K-fold CV; [cv_metrics_tables](#) for the underlying metrics formatter; [summarize_bootstrap_results](#) for the bootstrap analogue.

Examples

```
## Not run:
library(survival)
df <- survival::gbsg
df$grade3 <- as.integer(df$grade == "3")
df$time_months <- df$rfstime / 30.4375

fs <- forestsearch(
  df.analysis = df,
  confounders.name = c("age", "meno", "size", "grade3",
    "nodes", "pgr", "er"),
  outcome.name = "time_months",
  event.name = "status",
  treat.name = "hormon"
)

tf <- forestsearch_tenfold(fs.est = fs, sims = 100, Kfolds = 10)

# Basic diagnostic summary
cv_diag <- summarize_cv_results(tf)
cv_diag$identification_summary
cv_diag$grf_cut_summary
cv_diag$no_subgroup_decomposition

# With original-analysis comparison and plots
cv_diag_full <- summarize_cv_results(
  tf,
  original_sg = fs$sgharm,
```

```

    original_grf_cuts = fs$grf_cuts,
    create_plots      = TRUE
  )
  cv_diag_full$original_agreement
  cv_diag_full$plots$identification

## End(Not run)

```

```
summarize_simulation_results
```

Summarize Simulation Results

Description

Creates a summary table of operating characteristics across all simulations. Includes both HR and AHR metrics.

Usage

```

summarize_simulation_results(
  results,
  analyses = NULL,
  digits = 2,
  digits_hr = 3
)

```

Arguments

<code>results</code>	data.table with simulation results from <code>run_simulation_analysis</code>
<code>analyses</code>	Character vector. Analysis methods to include. Default: all
<code>digits</code>	Integer. Decimal places for proportions. Default: 2
<code>digits_hr</code>	Integer. Decimal places for hazard ratios. Default: 3

Value

Data frame with summary statistics

Examples

```

## Not run:
dgm <- create_gbsg_dgm()
sim_res <- run_simulation_analysis(dgm, nsim = 10)
summarize_simulation_results(sim_res)

## End(Not run)

```

summary.cox_ahr_cde	<i>Summary method for cox_ahr_cde objects</i>
---------------------	---

Description

Summary method for cox_ahr_cde objects

Usage

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

Arguments

object	A cox_ahr_cde object from cox_ahr_cde_analysis .
...	Additional arguments (not used).

Value

Invisibly returns the input object.

Examples

```
## Not run:  
library(survival)  
df <- survival::gbsg  
df$grade3 <- as.integer(df$grade == "3")  
res <- cox_ahr_cde_analysis(df,  
  outcome.name = "rfstime", event.name = "status", treat.name = "hormon",  
  confounders.name = c("age", "meno", "size", "grade3", "nodes", "pgr", "er"))  
summary(res)  
  
## End(Not run)
```

summary.dina	<i>Summary method for a DINA fit.</i>
--------------	---------------------------------------

Description

Builds a coefficient table with estimate, standard error, Wald z-statistic, and two-sided p-value for each coefficient.

Usage

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

Arguments

object	an object of class "dina".
...	unused.

Value

an object of class "summary.dina", printable via `print()`.

summary.forestsearch	<i>Summary Method for forestsearch Objects</i>
----------------------	--

Description

Provides a detailed summary of a ForestSearch analysis including input parameters, variable selection results, consistency evaluation, and the selected subgroup with key metrics.

Usage

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

Arguments

object	A forestsearch object returned by forestsearch .
...	Additional arguments (currently unused).

Value

Invisibly returns object.

Post-selection inference

Carries the same block as [print.forestsearch\(\)](#) (see there) and adds the log-scale standard errors (field, field-s with the naive complement SE beside it, and IJ two-term), the three largest re-selection frequencies with their labels, and $\hat{p}_{\text{hat_sum}}$ over the re-selection family. Absent MR results, the output is unchanged.

When the fit carries the membership-agreement diagnostics – that is, under `mr_inference_args = list(field_recovery = TRUE)` – one further line reports `sens_H`, the mean share of the identified subgroup that the field's re-selections retain, beside $\hat{p}$. The pair separates two very different low- $\hat{p}$ situations: $\hat{p}$ counts *exact* re-selections of $\hat{H}$, so a low $\hat{p}$ with a high `sens_H` says the draws re-selected near-twins of $\hat{H}$, while a low $\hat{p}$ with a low `sens_H` says they re-selected unrelated regions. Like $\hat{p}$ it is descriptive and no construction reads it, and it answers a narrower question than a bootstrap or cross-validation recovery rate: re-selection within the fixed candidate family under perturbation, not re-discovery from resampled data. The line is absent – and the rest of the output byte-identical – when the diagnostics were not requested. [print.forestsearch\(\)](#) does not report it.

Examples

```
## Not run:
fs <- forestsearch(df.analysis = mydata, ...)
summary(fs)

## End(Not run)
```

summary.fpr_calibration	Summary method for fpr_calibration objects
-------------------------	--

Description

Delegates to [print.fpr_calibration](#).

Usage

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

Arguments

object	An fpr_calibration object.
...	Passed to the print method.

Value

The input object, invisibly.

summary.subgroup_sims	Summarize an extreme-subgroups simulation study
-----------------------	---

Description

Computes, from the raw matrices of a [run_subgroup_sims\(\)](#) result (or a loaded RDS payload coerced with `class<-`), every statistic used by the vignettes' results table and forest panels: 1st/50th/99th percentile matrices for the estimate and its upper bound, convergence counts, mean subgroup N, tail probabilities, unconditional median display strings, the formatted `results_tbl`, category labels and colours, validity masks, the single/combination panel index vectors, and the high-risk panel (threshold filter, ITT anchor, ordering).

Usage

```
## S3 method for class 'subgroup_sims'
summary(
  object,
  hr_true = object$hr_true,
  probs = c(0.01, 0.5, 0.99),
  est_thresholds = NULL,
  ub_thresholds = NULL,
  est_label = NULL,
  ub_label = NULL,
  single_cats = c("ITT", "Clinical", "Continuous"),
  combo_cats = c("Interaction", "3-way", "Random"),
  pr_thresh_highrisk = 0.1,
  itt_name = "All Patients",
  cat_cols = c(ITT = "black", Clinical = "steelblue", Continuous = "darkgreen",
    Interaction = "purple4", `3-way` = "violetred3", Random = "darkorange"),
  ...
)
```

Arguments

<code>object</code>	A "subgroup_sims" object.
<code>hr_true</code>	True effect on the fitter's estimate scale, for reference lines (a hazard ratio for Cox fits, a mean difference for <code>subgroup_glm()</code> fits); defaults to the value stored on object.
<code>probs</code>	Quantile probabilities for the ECI matrices; the default <code>c(0.01, 0.50, 0.99)</code> matches the vignettes and downstream code indexes column 2 as the median.
<code>est_thresholds</code>	Length-2 numeric <code>c(low, high)</code> : the estimate tails are $\Pr(\text{est} < \text{low})$ and $\Pr(\text{est} > \text{high})$. <code>NULL</code> resolves via the effect metadata, then the HR legacy <code>c(0.5, 1.0)</code> . <code>NA</code> entries disable that tail (see above).
<code>ub_thresholds</code>	Length-2 numeric: the upper-bound tails are $\Pr(\text{UB} \geq \text{ub_thresholds}[1])$ and $\Pr(\text{UB} \geq \text{ub_thresholds}[2])$; the first also drives the high-risk panel. Resolution as for <code>est_thresholds</code> (HR legacy <code>c(2, 3)</code>).
<code>est_label, ub_label</code>	Display labels for the estimate and its upper bound in the table headers (legacy "HR" / "UB").
<code>single_cats, combo_cats</code>	Category labels routed to the single-variable and combination forest panels.
<code>pr_thresh_highrisk</code>	High-risk panel inclusion threshold on the first upper-bound tail probability, default 0.10.
<code>itt_name</code>	Name of the ITT row, always included in and anchored at the top of the high-risk panel.
<code>cat_cols</code>	Named colour map for categories.
<code>...</code>	Unused.

Details

Field names are structural and retained across outcome types (hr_q, pr_hr_lt050, pr_ub_ge2, ...), exactly as sim_hrs and hr.threshold serve generic duty elsewhere in the package: on a GLM result they hold the fitter’s estimate-scale statistics. Estimates follow the estimator layer’s harm-positive normalization, so the upper tail ($\Pr(\text{est} > \text{null})$, $\Pr(\text{UB} \geq t)$) always points toward harm whatever the raw outcome’s direction.

Threshold and label resolution: explicit arguments win; otherwise the result’s effect metadata supplies them (`subgroup_glm()` stamps scale-aware defaults from Phase 4.4 – ratio measures carry the legacy tails $c(0.5, 1) / c(2, 3)$, identity measures $c(\text{NA}, \text{null_value}) / c(\text{NA}, \text{NA})$ – with labels from the effect measure); otherwise the HR legacy values $c(0.5, 1.0) / c(2, 3)$ with the vignettes’ literal headers. An NA threshold degrades gracefully: its probability field is NA and its table column renders “-” under a placeholder header.

Structurally empty subgroups (all-NA columns) yield NA medians and NaN means/proportions, exactly as in the vignettes; the formatted table renders them as “-”.

Value

An object of class “subgroup_sims_summary”: a list with fields named as in the vignettes (hr_q, ub_q, n_valid, pct_valid, mean_n, pr_hr_lt050, pr_hr_gt1, pr_ub_ge2, pr_ub_ge3, mhr_uncond, mub_uncond, results_tbl, sg_names, cat, cat_cols, ok, ok_ub, idx_hr_single, idx_hr_combo, idx_ub_single, idx_ub_combo, n_single, n_combo, highrisk, n_sims, design, hr_true). Legacy calls return exactly this set; effect-aware calls append effect (the result’s metadata, possibly NULL when only explicit overrides were given), thresholds (`list(est =, ub =)` as resolved), and labels (`list(est =, ub =)`).

See Also

[run_subgroup_sims\(\)](#), [subgroup_glm\(\)](#)

summaryout_mrct	<i>Summary Tables for MRCT Simulation Results</i>
-----------------	---

Description

Creates summary tables from MRCT simulation results using the gt package. Summarizes hazard ratio estimates, subgroup identification rates, and classification of identified subgroups. Optionally displays two scenarios (e.g., alternative and null hypotheses) side by side.

Usage

```
summaryout_mrct(  
  pop_summary = NULL,  
  mrct_sims,  
  mrct_sims_null = NULL,  
  scenario_labels = c("Alternative", "Null"),  
  pop_summary_null = NULL,
```

```

sg_type = 1,
tab_caption = "Identified subgroups and estimation summaries",
digits = 3,
trim_threshold = 1000,
trim_fraction = 0.01,
table_width = 600,
font_size = 11,
showtable = TRUE
)

```

Arguments

pop_summary	List. Population summary from large sample approximation (optional). Default: NULL
mrct_sims	data.table. Simulation results from <code>mrct_region_sims</code> (first / primary scenario).
mrct_sims_null	data.table. Optional second set of simulation results (e.g., null hypothesis). When supplied, the table displays two value columns side by side. Default: NULL (single-scenario table).
scenario_labels	Character vector of length 2. Column headers for the two scenarios. Only used when <code>mrct_sims_null</code> is supplied. Default: <code>c("Alternative", "Null")</code> .
pop_summary_null	List. Population summary for the null scenario (optional). Default: NULL
sg_type	Integer. Type of subgroup summary: 1 = basic summary (found, biomarker, age); 2 = extended summary (all subgroup types). Default: 1
tab_caption	Character. Caption for the output table. Default: "Identified subgroups and estimation summaries"
digits	Integer. Number of decimal places for numeric summaries. Default: 3
trim_threshold	Numeric. When the raw mean of a metric exceeds this value in absolute terms, the summary switches to a symmetrically trimmed mean and SD (excluding the lower and upper <code>trim_fraction</code> of observations). Trimmed values are marked with * and a footnote is added to the table. Set to NULL to disable trimming entirely. Default: 1000.
trim_fraction	Numeric between 0 and 0.5. Fraction of observations to trim from each tail when trimming is triggered. Default: 0.01 (1 percent from each tail, i.e., the central 98 percent of values).
table_width	Numeric. Total table width in pixels. Column widths are allocated proportionally. Increase for HTML/wide displays (e.g., 750), decrease for beamer slides (e.g., 550). Default: 600.
font_size	Numeric. Base font size in pixels. Title is <code>font_size + 2</code> , subtitle matches base. Reduce for beamer (e.g., 9 or 10). Default: 11.
showtable	Logical. Print the table. Default: TRUE

Value

List with components:

res List of summary statistics from population. When dual-scenario, contains `res_alt` and `res_null`.

out_table Formatted gt table object, or data.frame if gt is unavailable.

data Processed `mrct_sims` data.table with derived variables. When dual-scenario, also contains `data_null`.

summary_df Data frame of computed summary statistics.

See Also

[mrct_region_sims](#) for generating simulation results

Examples

```
## Not run:
# Single scenario (backward-compatible)
summaryout_mrct(
  mrct_sims = results_alt,
  tab_caption = "H1: Heterogeneous treatment effect"
)

# Dual scenario: alternative vs null side-by-side
summaryout_mrct(
  mrct_sims = results_alt,
  mrct_sims_null = results_null,
  scenario_labels = c("Alternative (HTE)", "Null (uniform)"),
  tab_caption = "Operating characteristics: Alternative vs Null"
)

# Custom trimming: 2% from each tail when mean > 500
summaryout_mrct(
  mrct_sims = results_alt,
  mrct_sims_null = results_null,
  trim_threshold = 500,
  trim_fraction = 0.02
)

## End(Not run)
```

test_constant_cate

Test for Constant Conditional Average Treatment Effect

Description

H_0 : $\tau(x) = \tau$ for some τ and all x .

Usage

```
test_constant_cate(  
  Y,  
  W,  
  X,  
  poly_order = 1L,  
  covariate_select = c("all", "top_down", "bottom_up"),  
  t_threshold = 2,  
  regression = c("ols", "logistic", "poisson"),  
  offset = NULL  
)
```

Arguments

Y	Numeric outcome vector.
W	Treatment indicator (0/1).
X	Numeric covariate matrix (no factors; use as.numeric(factor) - 1 to convert).
poly_order	Integer. Default: 1.
covariate_select	"all", "top_down", or "bottom_up".
t_threshold	Numeric. Default: 2.0.
regression	"ols", "logistic", or "poisson".
offset	Numeric offset vector (Poisson only).

Value

A list with components test, chi_sq, df, p_value_chi, normal (normal-approximation statistic), p_value_normal, K (number of covariates), N0, N1, regression, covariates_selected, ate_diff (intercept-difference point estimate), and diff_slope, V_slope. Returns NULL if perm arm fits fail or fewer than two covariates are retained.

test_hte	<i>Run Both Crump et al. Tests</i>
----------	------------------------------------

Description

Run Both Crump et al. Tests

Usage

```
test_hte(  
  Y,  
  W,  
  X,  
  poly_order = 1L,
```

```

    covariate_select = c("all", "top_down", "bottom_up"),
    t_threshold = 2,
    regression = c("ols", "logistic", "poisson"),
    offset = NULL
  )

```

Arguments

Y	Numeric outcome vector.
W	Treatment indicator (0/1).
X	Numeric covariate matrix (no factors; use <code>as.numeric(factor) - 1</code> to convert).
poly_order	Integer. Default: 1.
covariate_select	"all", "top_down", or "bottom_up".
t_threshold	Numeric. Default: 2.0.
regression	"ols", "logistic", or "poisson".
offset	Numeric offset vector (Poisson only).

Value

Object of class `hte_test`.

test_hte_from_forestsearch

HTE Test Using ForestSearch-Selected Covariates

Description

HTE Test Using ForestSearch-Selected Covariates

Usage

```

test_hte_from_forestsearch(
  fs_result,
  df,
  outcome.name,
  treat.name,
  regression = c("ols", "logistic", "poisson"),
  offset = NULL,
  poly_order = 1L
)

```

Arguments

fs_result	A forestsearch result object.
df	Data frame used in the forestsearch() call.
outcome.name	Character. Outcome variable name.
treat.name	Character. Treatment variable name.
regression	Character: "ols", "logistic", or "poisson".
offset	Numeric offset vector (Poisson only).
poly_order	Integer. Default: 1.

Value

List with hte_test, covariates_used, source.

test_zero_cate	<i>Test for Zero Conditional Average Treatment Effect</i>
----------------	---

Description

H0: $\tau(x) = 0$ for all x .

Usage

```
test_zero_cate(  
  Y,  
  W,  
  X,  
  poly_order = 1L,  
  covariate_select = c("all", "top_down", "bottom_up"),  
  t_threshold = 2,  
  regression = c("ols", "logistic", "poisson"),  
  offset = NULL  
)
```

Arguments

Y	Numeric outcome vector.
W	Treatment indicator (0/1).
X	Numeric covariate matrix (no factors; use as.numeric(factor) - 1 to convert).
poly_order	Integer. Default: 1.
covariate_select	"all", "top_down", or "bottom_up".
t_threshold	Numeric. Default: 2.0.
regression	"ols", "logistic", or "poisson".
offset	Numeric offset vector (Poisson only).

Value

List with test results including covariates_selected.

validate_grf_data	<i>Validate input data for GRF analysis</i>
-------------------	---

Description

Checks that input data meets requirements for GRF analysis

Usage

```
validate_grf_data(W, D, n.min)
```

Arguments

W	Numeric vector. Treatment indicator
D	Numeric vector. Event indicator
n.min	Integer. Minimum subgroup size

Value

Logical. TRUE if data is valid, FALSE with warning otherwise

Examples

```
W <- rep(0:1, each = 50)
D <- rbinom(100, 1, 0.6)
validate_grf_data(W, D, n.min = 60)
```

validate_k_inter_effect	<i>Validate k_inter Effect on HR Heterogeneity</i>
-------------------------	--

Description

Test function to verify that k_inter properly modulates the difference between HR(H) and HR(Hc), and that AHR metrics align with Cox-based HRs.

Usage

```
validate_k_inter_effect(
  k_inter_values = c(-2, -1, 0, 1, 2, 3),
  verbose = TRUE,
  ...
)
```

Arguments

`k_inter_values` Numeric vector of `k_inter` values to test. Default: `c(-2, -1, 0, 1, 2, 3)`
`verbose` Logical. Print results. Default: `TRUE`
`...` Additional arguments passed to `create_gbsg_dgm`

Value

Data frame with `k_inter`, `hr_H`, `hr_Hc`, `AHR_H`, `AHR_Hc`, and `ratio` columns

Examples

```
## Not run:
# Test k_inter effect
results <- validate_k_inter_effect()

# k_inter = 0 should give hr_H approximately equals hr_Hc (ratio approximately 1)

## End(Not run)
```

<code>validate_mrct_data</code>	<i>Validate Dataset for MRCT Simulations</i>
---------------------------------	--

Description

Checks that a dataset contains all required variables for MRCT simulation functions and reports any issues. Required variables include outcome (`tte`, `event`), treatment (`treat`), continuous covariates (`age`, `bm`), and factor covariates (`male`, `histology`, `prior_treat`, `regA`).

Usage

```
validate_mrct_data(df.case, verbose = TRUE)
```

Arguments

`df.case` Data frame to validate
`verbose` Logical. Print detailed validation results. Default: `TRUE`

Details**Required Variables:**

The function checks for the following variables:

- **Outcome:** `tte` (time-to-event), `event` (0/1 indicator)
- **Treatment:** `treat` (0/1 indicator)
- **Continuous:** `age`, `bm` (biomarker)
- **Factor:** `male` (0/1), `histology`, `prior_treat` (0/1), `regA` (0/1)

The function also validates variable types and value ranges.

Value

Logical. TRUE if all requirements met, FALSE otherwise (invisibly)

See Also

[create_dgm_for_mrct](#) for creating DGM from validated data

Examples

```
## Not run:
# Check if dataset is ready for MRCT simulations
is_valid <- validate_mrct_data(df.case)

if (!is_valid) {
  stop("Please fix data issues before running simulations")
}

## End(Not run)
```

validate_subgroups	<i>Validate subgroup definitions against a data frame</i>
--------------------	---

Description

Evaluates every subgroup id expression against data (augmented with `flag_itt = 1L`, the scalar cutpoints as columns, and 0-valued benchmark membership columns) and fails with a named, collected error if any expression errors, is non-logical, or has the wrong length. Run automatically at the top of [run_subgroup_sims\(\)](#) so that a typo in a subgroup definition stops the study in milliseconds instead of surfacing as a silent all-NA column after the simulation loop.

Usage

```
validate_subgroups(
  subgroups,
  data,
  cutpoints = list(),
  benchmarks = benchmark_spec()
)
```

Arguments

subgroups	List of subgroup definitions; each element a list with character scalars <code>id</code> (an R expression over columns), <code>name</code> , and <code>grp</code> .
data	A data frame carrying the covariates the <code>id</code> expressions reference (e.g. <code>dgm\$df_source</code> or <code>dgm\$df_super</code>).
cutpoints	Named list of scalars exposed as columns (e.g. <code>list(age_med = 53)</code>), matching the runner's <code>cutpoints</code> argument.

benchmarks A `benchmark_spec()` (or NULL) whose membership columns are stubbed as 0L during validation.

Value

Invisibly TRUE; errors otherwise.

wilson_ci	<i>Wilson Score Confidence Interval</i>
-----------	---

Description

Computes Wilson score confidence interval for a proportion, which has better coverage properties than the normal approximation for small samples and proportions near 0 or 1.

Usage

```
wilson_ci(x, n, conf.level = 0.95)
```

Arguments

- x Integer. Number of successes.
- n Integer. Number of trials.
- conf.level Numeric. Confidence level (default 0.95).

Value

Named numeric vector with elements: estimate, lower, upper.

Examples

```
wilson_ci(90, 100)
wilson_ci(5, 20, conf.level = 0.90)
```

Index

- * **AFT**
 - find_k_inter_for_target_hr, 125
- * **bootstrap functions**
 - bootstrap_results, 10
- * **bootstrap**
 - generate_bootstrap_with_noise, 223
- * **calibration**
 - find_k_inter_for_target_hr, 125
- * **datagen**
 - generate_bootstrap_with_noise, 223
- * **interaction**
 - find_k_inter_for_target_hr, 125
- * **simulation**
 - find_k_inter_for_target_hr, 125
 - generate_bootstrap_with_noise, 223
- * **survival**
 - find_k_inter_for_target_hr, 125
- .consistency_glm_pieces(), 54
- acm.disjctif, 106
- analysis_covariates, 7
- analyze_subgroup, 8
- benchmark_spec, 9
- benchmark_spec(), 351, 352, 412
- bootstrap_results, 10, 16, 155, 157
- bootstrap_ystar, 14, 15, 155
- build_classification_table, 16, 21
- build_cox_formula, 11, 14, 18, 157
- build_estimation_table, 19, 48, 49, 170, 266, 267, 289
- calibrate_cens_adjust, 22, 29, 30
- calibrate_glm_interaction, 25, 229, 230
- calibrate_k_inter, 27, 29, 30, 70, 194
- calibrate_k_treat, 29
- calibrate_L_eff, 31, 318, 390
- cat, 34
- check_censoring_dgm, 24, 32, 370
- collapse_redundant_cuts, 139, 239
- collect_results, 33, 341
- compare_detection_curves, 36
- compare_fs_subgroups, 37
- compare_multiple_survreg, 38
- compare_selection_rules, 39, 122, 297
- compare_subgroup_sims, 43
- compute_detection_probability, 44, 46, 47
- compute_detection_probability_glm, 31, 46, 175, 390
- compute_dgm_cde, 48, 221, 359
- compute_frontier_cis, 40, 42, 49, 119, 120, 178, 280, 281, 297–299
- compute_node_metrics, 51
- compute_pareto_frontier, 281, 364
- compute_pareto_frontier(), 92
- confint.dina, 52
- consistency_resample, 52, 145, 380
- consistency_resample(), 55, 56
- consistency_resample_compare, 55
- consistency_resample_compare(), 54
- coord_cartesian, 289
- count_boot_id, 56
- cox_ahr_cde_analysis, 57, 319, 399
- cox_cs_fit, 58, 59, 244, 246, 247, 249
- cox_summary, 62
- create_dgm_for_mrct, 63, 276, 411
- create_forest_theme, 65, 315
- create_gbsg_dgm, 28, 68, 359, 371
- create_glm_row, 70, 253, 254
- create_grf_config, 71
- create_summary_table, 72
- cv_compare_results, 75
- cv_metrics_tables, 76, 79, 397
- CV_sgs, 77
- cv_summary_tables, 77, 78, 161
- cv_summary_text, 79
- d_eff_binary, 47, 106, 175, 389
- d_eff_continuous, 47, 107, 389

- d_eff_count, [47](#), [108](#), [389](#)
- d_eff_survival, [47](#), [109](#), [175](#), [389](#)
- dina, [80](#), [136](#)
- dina(), [83](#), [94](#), [97](#)
- dina_bagged, [82](#)
- dina_bagged(), [81](#), [94](#), [97](#)
- dina_fit, [84](#)
- dina_fit(), [80](#), [81](#), [89](#), [90](#), [94](#), [97](#)
- dina_fit_bagged, [87](#)
- dina_fit_bagged(), [82](#), [83](#), [94](#), [97](#)
- dina_frontier, [90](#), [136](#)
- dina_subgroup, [93](#), [103–105](#), [136](#), [137](#), [141](#)
- dina_subgroup(), [90](#), [92](#), [99](#), [100](#)
- dina_subgroup_bootstrap, [98](#), [105](#)
- dina_subgroup_bootstrap(), [94](#)
- dina_subgroup_refit, [100](#), [102](#), [103](#)
- dina_subgroup_refit(), [94](#)
- dummy_encode, [106](#)
- early_stop_decision, [109](#)
- estimate_propensity_scores, [110](#)
- evaluate_comparison, [112](#), [236](#)
- evaluate_consistency_twostage, [113](#), [116](#)
- evaluate_cuts_once, [115](#)
- evaluate_subgroup_consistency, [116](#)
- expand_covariates, [118](#)
- explain_pareto_selection, [41](#), [42](#), [119](#)
- extract_grf_args, [120](#)
- extract_selected_subgroups, [122](#)
- figure_note, [123](#)
- filter_by_lassokeep, [124](#)
- filter_call_args, [124](#)
- find_k_inter_for_target_hr, [125](#)
- find_leaf_split, [128](#)
- find_required_sample_size, [129](#)
- fit_causal_forest, [130](#)
- fit_cox_models, [14](#), [131](#), [157](#)
- fit_policy_trees, [132](#)
- foreach, [33](#)
- forest_height, [165](#)
- forest_height(), [286](#)
- forestsearch, [11](#), [39](#), [42](#), [114](#), [116](#), [117](#), [119](#),
[121](#), [132](#), [154](#), [156](#), [157](#), [159](#), [160](#),
[162](#), [163](#), [171](#), [173–175](#), [190](#), [191](#),
[194](#), [195](#), [200](#), [203](#), [205](#), [206](#), [208](#),
[236](#), [253](#), [254](#), [258](#), [268](#), [272](#),
[274–276](#), [280–282](#), [295](#), [296](#), [305](#),
[307](#), [313](#), [315](#), [322](#), [380–382](#), [384](#),
[385](#), [400](#)
- forestsearch(), [53](#), [55](#), [90–92](#), [94–97](#), [100](#),
[278](#), [279](#), [352](#), [386](#)
- forestsearch_bootstrap_dofuture, [11](#), [12](#),
[14](#), [51](#), [147](#), [152](#), [153](#), [154](#), [194](#), [303](#),
[313](#), [315](#), [324](#), [325](#), [394](#)
- forestsearch_bootstrap_dofuture(), [278](#),
[279](#)
- forestsearch_Kfold, [76](#), [153](#), [158](#), [163](#), [164](#),
[313](#), [315](#)
- forestsearch_KfoldOut, [78](#), [160](#), [161](#), [164](#)
- forestsearch_tenfold, [12](#), [13](#), [76](#), [160](#), [162](#),
[313](#), [395](#), [397](#)
- format_bootstrap_table, [166](#), [393](#)
- format_bootstrap_table(), [278](#), [279](#)
- format_CI, [155](#), [167](#)
- format_fs_details, [168](#)
- format_oc_results, [18](#), [21](#), [169](#), [267](#)
- fpr_approx, [171](#)
- fpr_calibration, [173](#), [177](#), [194](#)
- fpr_calibration_transfer, [176](#)
- frontier_member_flags, [51](#), [178](#), [299](#)
- fs_assert_membership, [179](#)
- fs_assert_membership(), [189](#), [215](#), [216](#)
- fs_declaration_calibration, [148](#), [180](#)
- fs_dgm_feasibility, [183](#)
- fs_dgm_scale, [186](#), [199](#), [201](#), [202](#), [212](#), [326](#)
- fs_export_maxeff_family, [188](#)
- fs_export_maxeff_family(), [179](#), [215](#), [216](#)
- fs_family_report, [190](#)
- fs_family_report(), [183](#)
- fs_fdr_report, [153](#), [157](#), [191](#)
- fs_fdr_report(), [183](#), [279](#)
- fs_focus_tag, [142](#), [194](#)
- fs_focus_tag(), [95](#), [100](#)
- fs_identification_structure(), [195](#)
- fs_identification_summary, [195](#)
- fs_identification_table, [197](#)
- FS_labels, [197](#)
- fs_mr_inference, [148](#), [150](#), [152](#), [153](#), [157](#),
[194](#)
- fs_mr_inference(), [183](#), [279](#)
- fs_mr_oc_summary, [198](#), [329](#)
- fs_oc_family_enumerate, [191](#), [200](#),
[203–209](#)
- fs_oc_grid, [184](#), [203](#), [206](#)
- fs_oc_invert, [204](#), [205](#)

- fs_oc_predict, [184](#), [185](#), [200](#), [202](#), [204](#), [207](#)
- fs_plot_bias_coverage, [210](#)
- fs_plot_bias_coverage(), [213](#)
- fs_rule_covariate_pairs, [211](#)
- fs_rule_covariate_pairs(), [292](#)
- fs_scale_se, [188](#), [212](#)
- fs_sim_bias_coverage, [212](#)
- fs_sim_bias_coverage(), [210](#)
- fs_sym_root, [208](#), [209](#), [214](#)
- fs_to_guohe, [215](#)
- fs_to_guohe(), [179](#), [188](#), [189](#)
- future::plan(), [351](#)
- generate_aft_dgm_flex, [22](#), [24](#), [29](#), [30](#), [32](#),
[33](#), [63](#), [64](#), [126](#), [127](#), [217](#), [228–230](#),
[274](#), [276](#), [344](#), [345](#), [348](#), [356](#), [368](#),
[370](#)
- generate_aft_dgm_flex(), [351](#)
- generate_bootstrap_synthetic, [222](#)
- generate_bootstrap_with_noise, [223](#)
- generate_detection_curve, [226](#), [287](#)
- generate_gbsg_bootstrap_general, [227](#)
- generate_glm_dgm, [26](#), [27](#), [186](#), [188](#), [200](#),
[228](#), [332](#), [345](#), [373](#)
- generate_glm_dgm(), [351](#), [352](#), [387](#), [388](#)
- get_bootstrap_exports, [14](#)
- get_combinations_info, [231](#)
- get_conf_force, [232](#)
- get_conf_force_jq, [232](#)
- get_covs_in, [233](#)
- get_Cox_sg, [14](#), [131](#), [234](#)
- get_dfpred, [235](#)
- get_dfpred(), [96](#), [97](#)
- get_dfRes, [14](#), [15](#), [236](#)
- get_dgm_hr, [20](#), [21](#), [49](#), [267](#)
- get_FSdata, [135](#), [136](#), [139](#), [238](#)
- get_leff, [240](#)
- get_sg_labels, [366](#)
- get_split_hr_fast, [241](#), [349](#)
- get_subgroup_membership, [242](#)
- gg_forest, [242](#)
- gg_forest(), [285](#), [286](#)
- glm_cs_fit, [244](#), [249](#)
- glm_effect_profile, [247](#), [247](#), [254](#)
- grf.subg.eval, [249](#)
- grf.subg.harm.glm, [71](#), [121](#), [137](#), [151](#), [249](#),
[250](#), [365](#), [381](#)
- grf.subg.harm.survival, [121](#), [128](#), [137](#),
[138](#), [252](#), [254](#), [255](#), [381](#)
- guohe_adaptive_r, [259](#)
- guohe_adaptive_r(), [216](#)
- guohe_algorithm3, [261](#)
- guohe_algorithm3(), [188](#), [215](#), [216](#), [259](#),
[260](#), [333](#)
- hrCI_format, [264](#)
- inject_mrct_structure, [264](#)
- inject_mrct_structure(), [8](#), [277](#), [300](#)
- interpret_estimation_table, [266](#)
- interpret_search_config, [267](#)
- lasso_selection, [269](#)
- latest_rds, [270](#)
- load_leff_calibration, [270](#)
- make_effect_estimator, [11](#), [114](#), [117](#), [271](#),
[349](#), [380](#), [384](#)
- make_effect_estimator(), [386–388](#)
- make_region_model, [273](#)
- message, [34](#)
- mr_estimates_table, [153](#), [157](#), [194](#), [278](#)
- mrct_region_sims, [64](#), [274](#), [301](#), [302](#), [361](#),
[362](#), [404](#), [405](#)
- mrct_truth_by_region, [277](#)
- n_pcnt, [279](#)
- pareto_frontier_table, [41](#), [42](#), [51](#), [119](#),
[120](#), [178](#), [279](#), [297](#), [299](#)
- plot.forestsearch, [282](#), [323](#)
- plot.fs_binary_plot
(plot_sg_glm_outcomes), [302](#)
- plot.fs_forestplot, [283](#)
- plot.fs_sg_plot, [284](#)
- plot.subgroup_sims_summary, [285](#)
- plot.subgroup_sims_summary(), [165](#)
- plot_detection_curve, [287](#)
- plot_effect_distribution, [288](#)
- plot_fs_identification_involvement,
[290](#)
- plot_fs_identification_structure, [291](#)
- plot_fs_rule_pairs, [292](#)
- plot_km_band_forestsearch, [292](#)
- plot_pareto_combined, [41](#), [42](#), [296](#)
- plot_pareto_frontier, [40](#), [42](#), [51](#), [120](#), [178](#),
[281](#), [297](#), [298](#)
- plot_region_modifier, [300](#)
- plot_sg_distribution, [300](#), [360](#)

- plot_sg_glm_outcomes, 302
- plot_sg_results, 282, 283, 295, 305
- plot_sg_weighted_km, 283, 295, 302, 307
- plot_spline_treatment_effect, 310
- plot_subgroup_effects, 310
- plot_subgroup_results_forestplot, 67, 283, 307, 312
- predict.dina, 317
- predict_fpr_corrected, 317, 390
- prepare_subgroup_data, 318
- print.cox_ahr_cde, 319
- print.dina_bagged, 320
- print.dina_subgroup, 320
- print.dina_subgroup_bootstrap, 321
- print.dina_subgroup_refit, 322
- print.forestsearch, 322
- print.forestsearch(), 400
- print.forestsearch_comparison, 323
- print.fpr_calibration, 324, 401
- print.fs_binary_plot
 - (plot_sg_glm_outcomes), 302
- print.fs_bootstrap, 156, 324
- print.fs_cv_summary, 325
- print.fs_declaration_calibration
 - (fs_declaration_calibration), 180
- print.fs_dgm_feasibility
 - (fs_dgm_feasibility), 183
- print.fs_dgm_scale, 326
- print.fs_family_report
 - (fs_family_report), 190
- print.fs_fdr_report, 326
- print.fs_forest_theme, 327
- print.fs_forestplot, 327
- print.fs_kfold, 328
- print.fs_mr_oc, 329
- print.fs_sg_plot, 329
- print.fs_tenfold, 330
- print.fs_weighted_km, 331
- print.gbsg_dgm, 331
- print.glm_cs_fit(glm_cs_fit), 244
- print.glm_dgm, 332
- print.glm_effect_profile, 332
- print.guohe_a3, 333
- print.guohe_ar, 333
- print.hte_test, 334
- print.leff_calibration, 334
- print.multi_survreg_comparison, 335
- print_grf_details, 335
- process_conf_force_expr, 336
- quick_km_band_plot, 337
- rbindlist, 33
- render_forestplot, 67, 315, 338
- render_reference_table, 339
- reset_workers, 35, 340
- rmst_calculation, 341
- rnorm, 225
- run_null_calibration, 342, 390
- run_simulation_analysis, 17, 20, 26, 34, 35, 169, 229, 230, 288, 289, 301, 302, 344, 373
- run_single_consistency_split, 349
- run_single_consistency_split(), 54–56
- run_subgroup_sims, 350
- run_subgroup_sims(), 10, 43, 44, 385, 387, 388, 401, 403, 411
- sample, 225
- save_forestplot, 353
- save_leff_calibration, 354
- select_best_subgroup, 355
- sensitivity_analysis_k_inter, 127, 356
- set.seed(), 277
- setup_gbsg_dgm, 194, 345, 348, 357
- setup_parallel_SGcons, 14, 359
- sg_consistency_out, 281, 307, 363
- SG_tab_estimates, 365, 366
- SG_tab_estimates_glm, 155, 161, 365
- sg_tables, 364, 393
- sgdist_fig_height, 302, 360
- SGplot_estimates, 289, 361
- sim_region_metrics, 374
- simulate_from_dgm, 22–24, 32, 33, 64, 194, 275, 276, 344, 348, 357, 359, 368
- simulate_from_dgm(), 350, 351, 374
- simulate_from_gbsg_dgm, 70, 371
- simulate_from_glm_dgm, 27, 185, 187, 230, 372
- simulate_from_glm_dgm(), 352, 387
- sort_subgroups, 142, 281, 364, 374, 378
- sort_subgroups(), 96, 97
- sort_subgroups_preview, 378
- stats::model.matrix(), 81
- stats::na.omit(), 81

subgroup.consistency, [114](#), [116](#), [117](#), [143](#),
[151](#), [153](#), [254](#), [258](#), [376](#)
subgroup.search, [383](#)
subgroup_cox, [385](#)
subgroup_cox(), [351](#), [352](#), [386](#), [388](#)
subgroup_glm, [386](#)
subgroup_glm(), [43](#), [44](#), [286](#), [351](#), [352](#), [402](#),
[403](#)
suggest_thresholds, [389](#)
summarize_bootstrap_events, [391](#)
summarize_bootstrap_results, [155](#), [392](#),
[393–395](#), [397](#)
summarize_bootstrap_results(), [278](#), [279](#)
summarize_bootstrap_subgroups, [324](#), [325](#),
[393](#), [393](#)
summarize_cv_results, [325](#), [395](#)
summarize_simulation_results, [18](#), [398](#)
summary.cox_ahr_cde, [399](#)
summary.dina, [399](#)
summary.forestsearch, [323](#), [400](#)
summary.fpr_calibration, [401](#)
summary.subgroup_sims, [401](#)
summary.subgroup_sims(), [44](#), [285](#), [286](#),
[352](#)
summaryout_mrct, [276](#), [362](#), [403](#)
survival::coxph(), [85](#), [86](#), [88](#), [385](#)
survival::strata(), [386](#)
survival::Surv(), [386](#)

tabulate, [16](#)
test_constant_cate, [405](#)
test_hte, [406](#)
test_hte_from_forestsearch, [407](#)
test_zero_cate, [408](#)

validate_grf_data, [409](#)
validate_k_inter_effect, [409](#)
validate_mrct_data, [410](#)
validate_subgroups, [411](#)
validate_subgroups(), [352](#)

warning, [34](#)
wilson_ci, [412](#)