Package 'ratingtables'

September 6, 2026

Title Table-Driven Insurance Rating

Version 0.2.2

Description Provides a lightweight, table-driven engine for executing insurance rating plans, including coverage-specific rating specifications, entity aggregation, and trace output for auditing. Given policy data, an ordered rating specification, and rating factor tables, it returns rated policies and, optionally, a step-by-step trace of the calculation.

License MIT + file LICENSE

URL https://github.com/gs-actuary/ratingtables

BugReports https://github.com/gs-actuary/ratingtables/issues

Encoding UTF-8

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Greg Sollenberger [aut, cre, cph]

Maintainer Greg Sollenberger <yagreg7@yahoo.com>

Repository CRAN

Date/Publication 2026-09-05 23:10:02 UTC

Contents

`aggregate_entity_values` *Aggregate rated entity values to parent records*

Description

Aggregate one or more numeric values from entity-level records to a parent or group level. This can be used, for example, to average driver factors or sum premiums for boats or scheduled items.

Usage

```
aggregate_entity_values(
  rated_entity_data,
  group_col,
  value_cols,
  aggregation = "mean",
  weight_col = NULL,
  output_names = NULL,
  output_prefix = NULL
)
```

Arguments

rated_entity_data	A data frame containing rated entity records.
group_col	A character string naming the column that identifies the parent or aggregation group.
value_cols	A character vector naming the numeric columns to aggregate.
aggregation	A character string specifying the aggregation method. Supported values are "sum", "mean", "min", "max", "count", and "weighted_mean".
weight_col	An optional character string naming the weight column. Required when aggregation = "weighted_mean".
output_names	An optional character vector giving the names of the aggregated output columns. It must have the same length as value_cols.
output_prefix	An optional character string prepended to generated output names when output_names is not supplied. By default, the aggregation name followed by an underscore is used.

Details

Common "sum", "mean", and "count" aggregations are performed in grouped vectorized passes rather than repeatedly subsetting the entity data once per parent group. Other supported aggregations reuse a single set of precomputed group indices.

Value

A data frame with one row per unique value of group_col and one aggregated column for each entry in value_cols.

Examples

```
rated_drivers <- data.frame(
  policy_id = c("P1", "P1", "P2"),
  indicated_BI = c(1.10, 0.90, 1.05)
)

aggregate_entity_values(
  rated_entity_data = rated_drivers,
  group_col = "policy_id",
  value_cols = "indicated_BI",
  aggregation = "mean",
  output_names = "average_BI"
)
```

append_rating_factors *Append wide rating-step values to rated data*

Description

Reshape normalized trace rows into wide applied-value columns and join them to the rated records.

Usage

```
append_rating_factors(rated_data, term_trace, by = "row_number")
```

Arguments

rated_data	A data frame containing rated policy or entity records.
term_trace	A data frame containing normalized rating trace rows.
by	A character vector naming the columns used both to reshape term_trace and to join the resulting wide data to rated_data. The default is "row_number".

Details

The function creates or replaces a row_number column in rated_data using its current row order.

Value

A data frame containing the rated records, a generated row_number column, and wide step_<number>_<term> columns containing applied step values.

Examples

```
ex <- example_rating_plan()

result <- rate_policies_with_trace(
  ex$policies,
  ex$plan
)

append_rating_factors(
  rated_data = result$rated_data,
  term_trace = result$term_trace,
  by = "row_number"
)
```

`apply_caps` *Apply rate-change caps to indicated premiums*

Description

Limit indicated premiums so that they do not increase or decrease by more than specified percentages relative to prior premiums.

Usage

```
apply_caps(
  rating_data,
  prior_data,
  by,
  coverages,
  max_increase = NULL,
  max_decrease = NULL
)
```

Arguments

`rating_data`	A data frame containing indicated premium columns named `indicated_<coverage>`.
`prior_data`	A data frame containing prior premium columns named `prior_<coverage>`.
`by`	A character vector naming the column or columns used to join `rating_data` and `prior_data`.
`coverages`	A character vector naming the coverages to cap.
`max_increase`	An optional nonnegative numeric value giving the maximum permitted proportional increase. For example, `0.10` permits a 10 percent increase.
`max_decrease`	An optional numeric value giving the maximum permitted proportional decrease. Its absolute value is used.

Value

A merged data frame with one `capped_<coverage>` column for each requested coverage.

Examples

```
indicated <- data.frame(
  policy_id = c("P1", "P2"),
  indicated_BI = c(125, 80)
)

prior <- data.frame(
  policy_id = c("P1", "P2"),
  prior_BI = c(100, 100)
)
```

```
apply_caps(
  rating_data = indicated,
  prior_data = prior,
  by = "policy_id",
  coverages = "BI",
  max_increase = 0.10,
  max_decrease = 0.15
)
```

`apply_rounding` *Apply a rounding rule*

Description

Apply a rounding rule

Usage

```
apply_rounding(x, rule = NA, digits = NA, increment = NA)
```

Arguments

`x`	A numeric vector.
`rule`	A character string naming the rounding rule. Supported values are `"none"`, `"round"`, `"floor"`, `"ceiling"`, `"nearest_dollar"`, `"nearest_cent"`, `"nearest_dime"`, and `"nearest_increment"`. A missing value also leaves `x` unchanged.
`digits`	The number of decimal places used when `rule = "round"`. A missing value defaults to zero.
`increment`	The numeric increment used when `rule = "nearest_increment"`.

Value

A numeric vector containing the rounded values. If `rule` is missing or `"none"`, `x` is returned unchanged.

Examples

```
premiums <- c(101.234, 105.678)

apply_rounding(
  premiums,
  rule = "nearest_cent"
)

apply_rounding(
  premiums,
  rule = "nearest_increment",
  increment = 5
)
```

`average_entity_factors` *Average entity rating factors*

Description

Average indicated coverage values across entity records belonging to the same parent record.

Usage

```
average_entity_factors(
  scored_entity_data,
  group_col,
  coverages,
  output_prefix = "avg_entity_factor_"
)
```

Arguments

`scored_entity_data`	A data frame containing scored entity records and columns named `indicated_<coverage>`.
`group_col`	A character string naming the column that identifies the parent or aggregation group.
`coverages`	A character vector of coverage names whose indicated values should be averaged.
`output_prefix`	A character string prepended to the generated output column names.

Value

A data frame with one row per parent group and one average entity factor column for each requested coverage.

Examples

```
rated_drivers <- data.frame(
  policy_id = c("P1", "P1", "P2"),
  indicated_BI = c(1.10, 0.90, 1.05)
)

average_entity_factors(
  scored_entity_data = rated_drivers,
  group_col = "policy_id",
  coverages = "BI"
)
```

`custom_rating_functions` *Custom rating functions*

Description

Custom rating functions allow a rating-plan step to use arbitrary R code when the calculation cannot conveniently be expressed as a standard factor lookup, interpolation, or input-value step.

Details

A custom function must accept the following arguments:

row A one-row data frame containing the record being rated.

coverage The coverage currently being rated.

current_premium The running premium immediately before this step.

plan The rating plan.

spec_row The rating-specification row for this step.

lookup A helper function for retrieving values from the plan's factor table.

The lookup helper has the form

`lookup(term_name, value_source = "factor_lookup", lookup_var = NULL, bounds = "error")`

and returns a lookup result whose numeric value is available as `$value`.

A custom function should normally return a single numeric value.

To use the function, set `value_source = "custom_function"` in the rating specification, supply the function name in `custom_function`, and register the function in the `custom_functions` argument to `new_rating_plan()`.

The `calculation_type` still controls how the returned value is applied. For example, `"multiplicative"` multiplies the running premium by the returned value, while `"replace"` replaces the running premium.

Examples

```
factor_table <- data.frame(
  coverage = c("BI", "BI"),
  term_name = c("base", "hidden_modifier"),
  term_value = c(100, 1.20)
)

rating_spec <- data.frame(
  step_number = 1:2,
  term_name = c("base", "custom_total"),
  value_source = c("factor_lookup", "custom_function"),
  calculation_type = c("multiplicative", "custom"),
  custom_function = c(NA, "add_fee_after_modifier")
```

```
)

add_fee_after_modifier <- function(
  row, coverage, current_premium, plan, spec_row, lookup
) {
  modifier <- lookup("hidden_modifier")$value
  current_premium * modifier + row$fee[[1]]
}

plan <- new_rating_plan(
  factor_table = factor_table,
  rating_spec = rating_spec,
  coverages = "BI",
  custom_functions = list(
    add_fee_after_modifier = add_fee_after_modifier
  )
)

policy <- data.frame(
  policy_id = "P1",
  fee = 5
)

rate_policies(policy, plan)
```

`ensure_slot_columns` *Ensure variable/level slot columns exist*

Description

Ensure variable/level slot columns exist

Usage

```
ensure_slot_columns(x, max_vars = 12)
```

Arguments

`x`	Data frame.
`max_vars`	Number of variable/level slots.

Value

A data frame containing the original columns in `x` plus any missing variable and level slot columns.

Examples

```
factors <- data.frame(
  term_name = "territory",
  term_value = 1.10
)

factors <- ensure_slot_columns(
  factors,
  max_vars = 2
)

factors
```

`example_rating_plan` *Build a small example rating plan*

Description

Build a small example rating plan

Usage

```
example_rating_plan()
```

Value

A list with two elements:

plan An example `rating_plan` object.

policies A data frame containing example policy records.

Examples

```
ex <- example_rating_plan()

ex$policies
ex$plan
```

`explain_rating` *Explain a rating calculation*

Description

Extract and format the rating trace for one source-data row and, optionally, one coverage.

Usage

```
explain_rating(rating_result, row_number = 1, coverage = NULL)
```

Arguments

`rating_result`	A `rating_result` object returned by `rate_policies_with_trace()`.
`row_number`	An integer identifying the source-data row to explain.
`coverage`	An optional character string identifying the coverage to include. If `NULL`, all coverages for the row are returned.

Value

An Excel-style data frame showing the selected rating steps in calculation order.

Examples

```
ex <- example_rating_plan()

result <- rate_policies_with_trace(
  ex$policies,
  ex$plan
)

explain_rating(
  rating_result = result,
  row_number = 1,
  coverage = "BI"
)
```

`find_duplicate_factors`
Find duplicate factor-table rows

Description

Identify factor-table rows that have duplicate lookup keys.

Usage

```
find_duplicate_factors(factor_table, max_vars = 12, ...)
```

Arguments

factor_table	A normalized long-form factor table.
max_vars	Maximum number of variable-level slot pairs to inspect.
...	Additional arguments accepted for backward compatibility.

Value

A data frame containing factor-table rows with duplicated lookup keys. An empty data frame is returned when no duplicates are found.

Examples

```
factor_table <- data.frame(
  state = c("IL", "IL", "IL"),
  coverage = c("BI", "BI", "BI"),
  term_name = c("territory", "territory", "territory"),
  term_value = c(1.10, 1.15, 0.95),
  variable1 = c("territory", "territory", "territory"),
  level1 = c("A", "A", "B"),
  stringsAsFactors = FALSE
)

duplicates <- find_duplicate_factors(
  factor_table,
  max_vars = 1
)

duplicates[
  ,
  c(
    "state",
    "coverage",
    "term_name",
    "term_value",
    "variable1",
    "level1"
  )
]
```

`join_entity_factors` *Join entity factors to rating data*

Description

Backward-compatible wrapper around `join_entity_values()` for joining aggregated entity factors to parent-level rating data.

Usage

```
join_entity_factors(rating_data, entity_factor_data, by)
```

Arguments

`rating_data`	A data frame containing the parent-level rating records.
`entity_factor_data`	A data frame containing aggregated entity factors.
`by`	A character vector naming the column or columns used to join the two data frames.

Value

A data frame containing all rows from `rating_data` with matching entity-factor columns appended.

Examples

```
policies <- data.frame(
  policy_id = c("P1", "P2"),
  base_premium = c(100, 120)
)

driver_factors <- data.frame(
  policy_id = c("P1", "P2"),
  average_driver_factor = c(1.05, 0.95)
)

join_entity_factors(
  rating_data = policies,
  entity_factor_data = driver_factors,
  by = "policy_id"
)
```

`join_entity_values` *Join aggregated entity values to parent records*

Description

Left-join aggregated entity-level values back to the parent-level rating data. When the entity-side join keys are unique and there are no overlapping non-key column names, a direct keyed match is used. More complicated joins fall back to base `merge()` to preserve general behavior.

Usage

```
join_entity_values(parent_data, entity_values, by)
```

Arguments

parent_data	A data frame containing parent-level records.
entity_values	A data frame containing aggregated entity values.
by	A character vector naming the column or columns used to join the two data frames.

Value

A data frame containing all rows from `parent_data` with matching columns from `entity_values`.

Examples

```
policies <- data.frame(
  policy_id = c("P1", "P2"),
  base_premium = c(100, 120)
)

driver_values <- data.frame(
  policy_id = c("P1", "P2"),
  average_driver_factor = c(1.05, 0.95)
)

join_entity_values(
  parent_data = policies,
  entity_values = driver_values,
  by = "policy_id"
)
```

---

lookup_exact_value *Look up an exact rating-table value*

Description

Select the single most specific factor-table row that matches a rating record, coverage, term, rate-set metadata, and variable-level conditions.

Usage

```
lookup_exact_value(row, coverage, plan, term_name)
```

Arguments

row	A one-row data frame containing the rating record.
coverage	A character string identifying the coverage being rated.
plan	A rating_plan object created by new_rating_plan().
term_name	A character string identifying the rating term to look up.

Value

A list containing the selected numeric value, the value source, the looked-up value, and the matching factor-row identifier.

Examples

```
ex <- example_rating_plan()
policy <- ex$policies[1, , drop = FALSE]

answer <- lookup_exact_value(
  row = policy,
  coverage = "BI",
  plan = ex$plan,
  term_name = "territory"
)

answer$value
answer$factor_row_id
```

---

lookup_factor_value *Look up a factor value by source*

Description

Dispatch a rating-table lookup to either exact matching or interpolated lookup according to value_source.

Usage

```
lookup_factor_value(
  row,
  coverage,
  plan,
  term_name,
  value_source = "factor_lookup",
  lookup_var = NULL,
  bounds = "error"
)
```

Arguments

row	A one-row data frame containing the rating record.
coverage	A character string identifying the coverage being rated.
plan	A rating_plan object created by new_rating_plan().
term_name	A character string identifying the rating term to look up.
value_source	A character string specifying the lookup method. Supported values are "factor_lookup" and "interpolated_lookup".
lookup_var	An optional character string naming the interpolation variable. Required for value_source = "interpolated_lookup".
bounds	A character string controlling out-of-range interpolation. Supported values are "error", "clamp", and "extrapolate".

Value

A list containing the selected or interpolated rating value and associated trace information.

Examples

```
ex <- example_rating_plan()
policy <- ex$policies[1, , drop = FALSE]

answer <- lookup_factor_value(
  row = policy,
  coverage = "BI",
  plan = ex$plan,
  term_name = "territory",
  value_source = "factor_lookup"
)

answer
```

---

lookup_interpolated_value

Look up an interpolated rating-table value

Description

Select the applicable interpolation curve for a rating record and calculate a linearly interpolated value from the surrounding table points.

Usage

```
lookup_interpolated_value(
  row,
  coverage,
  plan,
  term_name,
  lookup_var,
  bounds = "error"
)
```

Arguments

row	A one-row data frame containing the rating record.
coverage	A character string identifying the coverage being rated.
plan	A rating_plan object created by new_rating_plan().
term_name	A character string identifying the rating term to look up.
lookup_var	A character string naming the numeric input variable used as the interpolation axis.
bounds	A character string controlling values outside the available interpolation range. Supported values are "error", "clamp", and "extrapolate".

Value

A list containing the interpolated value and supporting trace information, including the lower and upper levels, values, interpolation weight, and factor-row identifiers.

Examples

```
factor_table <- data.frame(
  state = c("IL", "IL"),
  charter = c("STD", "STD"),
  book_segment = c("new", "new"),
  rate_eff_date = as.Date(c("2025-01-01", "2025-01-01")),
  rate_exp_date = as.Date(c("2025-12-31", "2025-12-31")),
  coverage = c("BI", "BI"),
  term_name = c("limit_factor", "limit_factor"),
  term_value = c(1.00, 1.20),
  variable1 = c("limit_value", "limit_value"),
  level1 = c("100", "200"),
  stringsAsFactors = FALSE
)

rating_spec <- data.frame(
  step_number = 1,
  term_name = "limit_factor",
  value_source = "interpolated_lookup",
  calculation_type = "multiplicative",
  lookup_var = "limit_value",
  stringsAsFactors = FALSE
```

```
)

plan <- new_rating_plan(
  factor_table = factor_table,
  rating_spec = rating_spec,
  coverages = "BI"
)

policy <- data.frame(
  policy_id = "P1",
  state = "IL",
  charter = "STD",
  book_segment = "new",
  rating_date = as.Date("2025-06-01"),
  limit_value = 150,
  stringsAsFactors = FALSE
)

answer <- lookup_interpolated_value(
  row = policy,
  coverage = "BI",
  plan = plan,
  term_name = "limit_factor",
  lookup_var = "limit_value"
)

answer$value
answer$interpolation_weight
```

`lookup_term_value` *Look up an exact rating term value*

Description

Compatibility wrapper around `lookup_exact_value()`. By default, it returns only the numeric factor value.

Usage

```
lookup_term_value(row, coverage, plan, term_name, return_match = FALSE, ...)
```

Arguments

`row`	A one-row data frame containing the rating record.
`coverage`	A character string identifying the coverage being rated.
`plan`	A `rating_plan` object created by `new_rating_plan()`.
`term_name`	A character string identifying the rating term to look up.
`return_match`	Logical. If `TRUE`, return the complete lookup result; otherwise return only its numeric value.

...	Additional arguments accepted for backward compatibility. They are currently ignored.

Value

If `return_match = FALSE`, a numeric rating value. If `return_match = TRUE`, a list containing the value and matching-row information.

Examples

```
ex <- example_rating_plan()
policy <- ex$policies[1, , drop = FALSE]

lookup_term_value(
  row = policy,
  coverage = "BI",
  plan = ex$plan,
  term_name = "territory"
)

lookup_term_value(
  row = policy,
  coverage = "BI",
  plan = ex$plan,
  term_name = "territory",
  return_match = TRUE
)
```

---

make_empty_slots *Add empty variable/level slot columns*

Description

Add empty variable/level slot columns

Usage

```
make_empty_slots(n = 1, max_vars = 12)
```

Arguments

n	Number of rows.
max_vars	Number of variable/level slots.

Value

A data frame with `n` rows and one `variable` and `level` column pair for each requested slot.

Examples

```
slots <- make_empty_slots(
  n = 2,
  max_vars = 3
)

slots
```

`new_rating_plan` *Create a rating plan*

Description

Create a rating plan

Usage

```
new_rating_plan(
  factor_table,
  rating_spec,
  coverages,
  use_rate_set_key = FALSE,
  max_vars = 12,
  policy_id_col = "policy_id",
  custom_functions = list(),
  validate = TRUE,
  metadata = list()
)
```

Arguments

`factor_table`	A normalized long-format data frame containing at least `term_name` and `term_value`, plus optional coverage, rate-set, and variable-level lookup columns.
`rating_spec`	A data frame containing at least `term_name` and `calculation_type`. Optional specification columns are normalized and supplied with defaults.
`coverages`	A character vector naming the coverages to rate.
`use_rate_set_key`	Logical. If `TRUE`, factor rows are selected using a `rate_set_key` supplied in the rating data rather than automatic state-, charter-, book-segment-, and date-based selection.
`max_vars`	A nonnegative integer giving the maximum number of variable-level slot pairs in the factor table.
`policy_id_col`	A single character string naming the policy or record identifier column. Its value is stored as `record_id` in trace output; if the column is absent, the source row number is used.

custom_functions
: A named list of custom rating functions referenced by rating-specification rows with value_source = "custom_function". Each function is called with row, coverage, current_premium, plan, spec_row, and lookup. See custom_rating_functions for the function contract and examples.

validate
: Logical. If TRUE, run validate_rating_plan() before returning the plan.

metadata
: An optional list stored unchanged in the rating-plan object.

Details

Exact lookup structure and standard rating-step metadata are precompiled when the plan is created. Value-only edits to plan$factor_table$term_value can reuse the compiled lookup structure. If lookup keys or the rating specification are changed, rate_policies() detects the structural change and rebuilds the internal cache before rating.

Value

A rating_plan object containing the normalized factor table, rating specification, coverages, configuration, custom functions, and metadata. The object also contains an internal precompiled cache used by the standard non-trace batch-rating engine. The cache is an implementation detail and should not be edited directly.

Examples

```
ex <- example_rating_plan()

plan <- new_rating_plan(
  factor_table = ex$plan$factor_table,
  rating_spec = ex$plan$rating_spec,
  coverages = "BI"
)

print(plan)
summary(plan)
```

---

populate_slots *Populate variable/level slots from named values*

Description

Populate variable/level slots from named values

Usage

```
populate_slots(x, ..., max_vars = 12)
```

Arguments

x	A data frame to which slot columns will be added or updated.
...	Named vectors of levels. Each argument name becomes the value of a variable<n> column, and its values populate the corresponding level<n> column. Each vector must have length one or nrow(x).
max_vars	A nonnegative integer giving the maximum number of variable-level slot pairs.

Value

A data frame containing x with variable and level slot columns populated from the named values supplied through ....

Examples

```
factors <- data.frame(
  term_name = c("territory_limit", "territory_limit"),
  term_value = c(1.10, 0.95)
)

factors <- populate_slots(
  factors,
  territory = c("A", "B"),
  limit = "100/300",
  max_vars = 2
)

factors
```

---

rate_entities *Rate child or entity records*

Description

Apply a rating plan to entity-level records such as drivers, vehicles, boats, or scheduled items, returning both rated values and trace output.

Usage

```
rate_entities(entity_data, plan, validate = TRUE)
```

Arguments

entity_data	A data frame containing one row per entity to be rated.
plan	A rating_plan object created by new_rating_plan().
validate	Logical. If TRUE, validate the entity data before rating.

Value

A `rating_result` object containing `rated_data`, `term_trace`, and the rating plan.

Examples

```
ex <- example_rating_plan()

drivers <- ex$policies
drivers$household_id <- "H1"
drivers$policy_id <- c("D1", "D2")

result <- rate_entities(
  entity_data = drivers,
  plan = ex$plan
)

result$rated_data
result$term_trace
```

`rate_one_row_one_coverage`

Rate one record for one coverage

Description

Execute the applicable rating specification one step at a time for a single record and coverage.

Usage

```
rate_one_row_one_coverage(row, coverage, plan, row_number = 1)
```

Arguments

`row`	A one-row data frame containing the rating record.
`coverage`	A character string identifying the coverage to rate.
`plan`	A `rating_plan` object created by `new_rating_plan()`.
`row_number`	An integer identifying the row within the source rating data. This value is included in the trace output.

Value

A list with two elements: `value`, containing the final indicated value, and `trace`, containing one trace row per rating-specification step.

Examples

```
ex <- example_rating_plan()

answer <- rate_one_row_one_coverage(
  row = ex$policies[1, , drop = FALSE],
  coverage = "BI",
  plan = ex$plan,
  row_number = 1
)

answer$value
answer$trace
```

`rate_policies` *Rate policy records*

Description

Apply a rating plan to every row and coverage in a policy-level data frame, returning only the resulting rated data. Plans using supported standard rating operations are evaluated with a vectorized batch engine. Plans using unsupported operations fall back to the general trace-capable engine.

Usage

```
rate_policies(rating_data, plan, validate = TRUE)
```

Arguments

`rating_data`	A data frame containing one row per policy or rating record.
`plan`	A `rating_plan` object created by `new_rating_plan()`.
`validate`	Logical. If `TRUE`, validate `rating_data` before rating.

Value

A data frame containing the original rating data plus one `indicated_<coverage>` column for each coverage in the rating plan.

Examples

```
ex <- example_rating_plan()

rated <- rate_policies(
  rating_data = ex$policies,
  plan = ex$plan
)

rated
```

`rate_policies_with_trace` *Rate policy records with trace output*

Description

Apply a rating plan to every row and coverage in a policy-level data frame and retain step-by-step trace information.

Usage

```
rate_policies_with_trace(rating_data, plan, validate = TRUE)
```

Arguments

`rating_data`	A data frame containing one row per policy or rating record.
`plan`	A `rating_plan` object created by `new_rating_plan()`.
`validate`	Logical. If `TRUE`, validate `rating_data` before rating.

Value

A `rating_result` object containing:

rated_data The original records with indicated coverage values.

term_trace Step-by-step rating trace rows.

plan The rating plan used for the calculation.

Examples

```
ex <- example_rating_plan()

result <- rate_policies_with_trace(
  rating_data = ex$policies,
  plan = ex$plan
)

result$rated_data
result$term_trace
```

rate_set_to_tables *Split a rate set into review-friendly tables*

Description

Separate a normalized long-form factor table into a collection of smaller tables suitable for human review.

Usage

```
rate_set_to_tables(factor_table)
```

Arguments

factor_table	A normalized long-form factor table.

Value

A named list of data frames containing review-friendly subsets of the supplied factor table.

Examples

```
ex <- example_rating_plan()

tables <- rate_set_to_tables(
  ex$plan$factor_table
)

names(tables)
tables$territory
```

required_policy_fields *Identify required rating-data fields*

Description

Collect fields referenced by factor-table lookup slots, specification input and lookup columns, and applicable rate-set metadata.

Usage

```
required_policy_fields(plan)
```

Arguments

plan	A rating_plan object created by new_rating_plan().

Value

A character vector containing the unique input-data column names required by the plan.

Examples

```
ex <- example_rating_plan()

required_policy_fields(ex$plan)
```

`score_entity_rows` *Score child or entity records*

Description

Score entity records without constructing trace output. This uses `rate_policies()` so supported standard plans can use the optimized vectorized batch-rating engine.

Usage

```
score_entity_rows(entity_data, plan, validate = TRUE)
```

Arguments

`entity_data`	A data frame containing one row per entity to be rated.
`plan`	A `rating_plan` object created by `new_rating_plan()`.
`validate`	Logical. If `TRUE`, validate the entity data before rating.

Value

A data frame containing the original entity data and calculated indicated values.

Examples

```
ex <- example_rating_plan()

drivers <- ex$policies
drivers$household_id <- "H1"
drivers$policy_id <- c("D1", "D2")

scored <- score_entity_rows(
  entity_data = drivers,
  plan = ex$plan
)

scored
```

trace_to_excel_style *Reshape rating trace to an Excel-style step table*

Description

Select and order the principal trace columns to produce a compact, human-readable view of the rating calculation.

Usage

```
trace_to_excel_style(term_trace)
```

Arguments

term_trace	A data frame containing normalized rating trace rows, typically from rate_policies_with_trace().

Value

A data frame ordered by row number, coverage, and step number. It contains the available columns among row_number, record_id, coverage, step_number, term_name, value_source, calculation_type, applied_value, value_before_step, and value_after_step.

Examples

```
ex <- example_rating_plan()

result <- rate_policies_with_trace(
  ex$policies,
  ex$plan
)

trace_to_excel_style(
  result$term_trace
)
```

trace_to_wide_factors *Reshape rating trace to wide step-value columns*

Description

Convert normalized trace rows into one row per unique combination of identifying columns, with a separate column for each rating step. Generated step columns contain the trace's applied_value.

Usage

```
trace_to_wide_factors(
  term_trace,
  id_cols = c("row_number", "record_id", "coverage")
)
```

Arguments

term_trace	A data frame containing normalized rating trace rows, typically from rate_policies_with_trace().
id_cols	A character vector naming columns in term_trace that define the output rows. Names not present in term_trace are ignored.

Value

If term_trace is nonempty, a data frame with one row per unique combination of the available id_cols and columns named step_<number>_<term> containing applied step values. If term_trace is empty, the empty input data frame is returned unchanged.

Examples

```
ex <- example_rating_plan()

result <- rate_policies_with_trace(
  ex$policies,
  ex$plan
)

trace_to_wide_factors(
  result$term_trace
)
```

---

validate_factor_table *Check minimum factor-table structure*

Description

Confirm that a factor table contains term_name and term_value and that every term_value is numeric or coercible to numeric. This function does not check for duplicate lookup keys; use find_duplicate_factors() for that.

Usage

```
validate_factor_table(factor_table, max_vars = 12)
```

Arguments

factor_table	A data frame containing rating factors.
max_vars	A nonnegative integer giving the number of variable-level slot pairs to add before validation.

Value

Invisibly returns TRUE if validation succeeds. Otherwise, the function stops with an error.

Examples

```
ex <- example_rating_plan()

isTRUE(
  validate_factor_table(
    ex$plan$factor_table,
    max_vars = ex$plan$max_vars
  )
)
```

---

validate_policy_data *Check required rating-data columns*

Description

Confirm that the input data contains every column returned by required_policy_fields(). This function checks column presence but does not validate individual values or column types.

Usage

```
validate_policy_data(rating_data, plan)
```

Arguments

rating_data	A data frame containing policy, risk, or entity records.
plan	A rating_plan object created by new_rating_plan().

Value

Invisibly returns TRUE if validation succeeds. Otherwise, the function stops with an error.

Examples

```
ex <- example_rating_plan()

isTRUE(
  validate_policy_data(
    rating_data = ex$policies,
    plan = ex$plan
  )
)
```

validate_rate_sets *Check rate-set identifiers and date ranges*

Description

Check that `rate_set_key` contains no missing or blank values when present, and that `rate_eff_date` is not later than `rate_exp_date` when both date columns are present.

Usage

```
validate_rate_sets(factor_table)
```

Arguments

`factor_table`	A data frame containing rating factors and optional rate-set fields.

Value

Invisibly returns `TRUE` if validation succeeds. Otherwise, the function stops with an error.

Examples

```
ex <- example_rating_plan()

isTRUE(
  validate_rate_sets(
    ex$plan$factor_table
  )
)
```

validate_rating_plan *Check a rating plan*

Description

Run factor-table, rating-specification, and rate-set checks and confirm that custom functions referenced by the specification are registered as functions in the plan.

Usage

```
validate_rating_plan(plan)
```

Arguments

`plan`	A `rating_plan` object created by `new_rating_plan()`.

Value

Invisibly returns TRUE if validation succeeds. Otherwise, the function stops with an error.

Examples

```
ex <- example_rating_plan()

isTRUE(
  validate_rating_plan(ex$plan)
)
```

`validate_rating_spec` *Check rating-specification fields*

Description

Normalize a rating specification and check its value sources, calculation types, and required lookup, input, or custom-function fields.

Usage

```
validate_rating_spec(rating_spec, ...)
```

Arguments

`rating_spec`	A data frame containing the rating specification. It must contain `term_name` and `calculation_type`; omitted optional columns are added during normalization.
`...`	Additional arguments accepted for backward compatibility and currently ignored.

Value

Invisibly returns TRUE if validation succeeds. Otherwise, the function stops with an error.

Examples

```
ex <- example_rating_plan()

isTRUE(
  validate_rating_spec(
    ex$plan$rating_spec
  )
)
```

Index