Package 'expstudy'

May 8, 2026

Title Tools for Actuarial Experience Studies

Version 2.0.0

Description Experiences studies are an integral component of the actuarial control cycle. Regardless of the decrement or policyholder behavior of interest, the analyses conducted is often the same. Ultimately, this package aims to reduce time spent writing the same code used for different experience studies, therefore increasing the time for to uncover new insights inherit within the relevant experience.

License GPL (>= 3)

URL https://github.com/cb12991/expstudy, https://cb12991.github.io/expstudy/

BugReports https://github.com/cb12991/expstudy/issues

Depends R (>= 3.5.0)

Imports cli, dplyr, lifecycle, rlang, stats, utils

Suggests covr, roxygen2, testthat

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.2.3

LazyData true

NeedsCompilation no

Author Cody Buehler [aut, cre]

Maintainer Cody Buehler <cb12991@me.com>

Repository CRAN

Date/Publication 2024-02-05 14:20:10 UTC

Contents

`compute_fct_adjs` *Calculate adjustment factors for an underlying assumption*

Description

There often are situations where an industry table is used for an assumed rate due to a company lacking sufficient credibility to write their own assumption. However, as experience becomes more available, a company would likely want to incorporate this experience into the industry assumption because it provides valuable insight into their own policyholders. A common industry approach is to apply "factor adjustments" developed using company experience to the industry assumption.

Usage

```
compute_fct_adjs(
  .data,
  expected_rate,
  measure_sets = guess_measure_sets(.data),
  amount_scalar = NULL,
  method = c("simultaneous", "sequential"),
  cred_wt_adjs = FALSE,
  balance_adjs = FALSE,
  na.rm = FALSE
)
```

Arguments

`.data`	A `base::data.frame()` that houses an experience study.
`expected_rate`	The underlying expected rate in the experience study for which factor adjustments are being generated for.
`measure_sets`	A (potentially named) list of measure sets. Only need to specify once if chaining multiple `expstudy` functions as the `measure_sets` will be passed as an attribute in results.
`amount_scalar`	A numeric vector to use when determining amount-weighted expecteds and variances. The function will determine whether or not the new expecteds/variances are amount-weighted if the corresponding actuals in the study have values greater than 1 (actuals that are not amount-weighted, i.e., counts, should only be 0 or 1).
`method`	String indicating the method of determining factor adjustments:

	* `simultaneous` will calculate factor adjustments for all combinations of group values in one iteration. * `sequential` will calculate factor adjustments for each grouping variable individually and applies that factor adjustment to the underlying expected rate before continuing with the next grouping variable's factor computation.
cred_wt_adjs	Logical indicating if factor adjustments should be credibility-weighted using partial credibility scores.
balance_adjs	Logical indicating if credibility-weighted adjustments should be scaled to produce a 100% A/E ratio in aggregate (has no effect if cred_wt_adjs = FALSE).
na.rm	logical. Should missing values (including NaN) be removed?

Details

This function piggy-backs off of measure_sets defined in other expstudy functions to quickly produce factor adjustments under a variety of methods. Providing a dplyr::grouped_df() will generate factors for each group according to the method specified. If two or more grouping variables are provided, an additional "composite" factor adjustment will also be generated which is the product of each individual adjustment.

Value

A list of data frames that house factor adjustments for each measure set provided in measure_sets.

Examples

```
mortexp |>
  dplyr::group_by(
    GENDER,
    SMOKING_STATUS
  ) |>
  compute_fct_adjs(
    EXPECTED_MORTALITY_RT,
    amount_scalar = FACE_AMOUNT
  )
```

---

guess_measure_sets *Guess a measure set using regexs*

Description

Attempt to guess the names of a **measure set** using regular expressions (or regexs).

Usage

```
guess_measure_sets(
  data,
  measure_regexs = getOption("expstudy.default_measure_regexs"),
  measure_set_prefixes = getOption("expstudy.default_measure_set_prefixes"),
  measure_set_suffixes = getOption("expstudy.default_measure_set_suffixes")
)
```

Arguments

data	A data.frame that houses an experience study.
measure_regexs	A named list of patterns to use as regexs when guessing columns in the study dataset to be used for one study measure in each measure set. There must be one column for each measure in a measure set (actuals, expecteds, exposures, and variances). Defaults to getOption('expstudy.default_measure_regexs').
measure_set_prefixes, measure_set_suffixes	Character vectors that will be use to differentiate the same measure in one measure set from another measure set. Using NULL indicates that the study measures do not differ by prefix/suffix and will error if more than one column is guessed using the measure regex for a single measure. Defaults to measures sets not differing by prefix (measure_set_prefixes = NULL) but do differ by count and amount suffixes (measure_set_prefixes = c('_CNT', '_AMT')). If the experience study has columns that follow a consistent naming structure, this function can seamlessly provide other expstudy functions information on the study measures to use for various calculations.

Value

A named list of measure sets that identify common variables used for expstudy analyses.

Examples

```
guess_measure_sets(mortexp)
```

---

metrics *Experience study metrics*

Description

A collection of common metrics used in an actuarial environment are provided. Two versions of each metric functions have been developed: one where it takes a measure set for an experience study as its primary argument, and one where vectors can be provided instead.

Usage

```
avg_observed(measure_set, ...)

avg_observed_vec(actuals, exposures, ...)

avg_expected(measure_set, ...)

avg_expected_vec(expecteds, exposures, ...)

ci_fctr(measure_set, se_conf = 0.95, two_tailed = TRUE, ...)

ci_fctr_vec(exposures, variances, se_conf = 0.95, two_tailed = TRUE, ...)

ae_ratio(measure_set, ...)

ae_ratio_vec(actuals, expecteds, ...)

credibility(measure_set, distance_from_mean = 0.05, cred_conf = 0.95, ...)

credibility_vec(
  expecteds,
  variances,
  distance_from_mean = 0.05,
  cred_conf = 0.95,
  ...
)
```

Arguments

`measure_set`	A named character vector or list with each element mapping a column in the experience study to one of the following measures: `actuals`, `expecteds`, `exposures`, or `variances`.
`...`	Not used directly and be left blank.
`actuals`, `expecteds`, `exposures`, `variances`	Columns in experience study that correspond to individual measures for vector versions of metric functions.
`se_conf`	A number between 0 and 1 corresponding to the confidence level surrounding the standard error calculation.
`two_tailed`	A boolean indicating whether or not a two-tailed hypothesis test should be utilized.
`distance_from_mean`	A number between 0 and 1 representing the precision of the credibility estimate.
`cred_conf`	A number between 0 and 1 corresponding to the confidence level surrounding the credibility calculation.

Details

Metric functions that use a measure set as its primary argument are intended to be used with `mutate_metrics()` and return a (quosure)`rlang::quo()`. Use the vector versions (those ending in _vec) if instead a numeric vector result is desired.

Value

Measure set versions return a (quosure)`rlang::quo()` to be evaluated in `mutate_metrics()`. Vector versions numeric vector of the same length of measures used in the calculation per group (if grouping applied).

Functions

- `avg_observed()`: Calculates the average actual decrements observed per unit of exposure.
- `avg_expected()`: Calculates the average expected decrements per unit of exposure.
- `ci_fctr()`: Calculates the additive factor which constructs a confidence interval around the expected decrement rate for a given level of confidence.
- `ae_ratio()`: Calculates the ratio of actual decrements to expected decrements, also referred to as the AE ratio.
- `credibility()`: Calculates the credibility score according to limited fluctuation credibility theory.

`mortexp` *Mortality experience study*

Description

A dataset containing an example of a mortality experience study for 1000 fictional whole life insurance policyholders.

Usage

```
mortexp
```

Format

A `data.frame()` with over 175,000 rows and 24 columns:

`AS_OF_DATE` This indicates which point in time a record encompasses.

`POLICY_HOLDER` An index used to distinguish policyholders. In this example the policyholder is also the (only) insured.

`GENDER`, `SMOKING_STATUS`, `UNDERWRITING_CLASS`, `INSURED_DOB`, `ISSUE_DATE`, `ISSUE_AGE` Various characteristics of an insured at time of issue.

`FACE_AMOUNT` Face amount of insurance for a corresponding policy.

`TERMINATION_DATE` If terminated, the effective date of termination. An `NA` value will be listed for policies that are still in-force.

`ATTAINED_AGE` The age of the insured at the record's `AS_OF_DATE`

`EXPECTED_MORTALITY_RT` An expected mortality rate for an insured. The rate is calculated according to De Moivre's Law (also known as uniform distribution of deaths, or UDD) with $\omega = 120$.

`POLICY_DURATION_MNTH`, `POLICY_DURATION_YR` Temporal indices describing how long a policy has been in-force at the `AS_OF_DATE`. For example, when a policy is first issued (i.e., $t = 0$), it is in policy duration year one and policy duration month one.

`POLICY_STATUS` The current status of the policy, either in-force, surrendered, or death. The value will be listed for each policy record even though a decrement only occurs at the end of the policy's duration (for policies which are no longer in-force).

`MORT_EXPOSURE_CNT`,`MORT_EXPOSURE_AMT` Measures how many policyholders or how much face amount of insurance is exposed to the risk of decrement for an associated observations.

`MORT_ACTUAL_CNT`,`MORT_ACTUAL_AMT` Measures the decrement occurrence on a policy count or face amount of insurance basis.

`MORT_EXPECTED_CNT`,`MORT_EXPECTED_AMT` Measures the expected decrement value for an associated observation on a policy count or face amount of insurance basis.

`MORT_VARIANCE_CNT`,`MORT_VARIANCE_AMT` Measures the variance of the decrement expectation, also on a policy count or face amount of insurance basis. Used to calculate credibility scores and confidence intervals.

Source

All policy record detail is randomly generated. See the Society of Actuaries' publication on experience study calculations for additional information regarding experience study calculations.

`mutate_expecvar` *Add new expecteds and variances to an experience study*

Description

`mutate_expecvar()` uses a new expected rate for a decrement of interest and adds a corresponding expected decrements column and corresponding variance of expected decrements column. If there are already expecteds and variances measures within the study dataset, either new, prefixed columns will be added or the current expecteds and variances can be overwritten.

Usage

```
mutate_expecvar(
  .data,
  new_expected_rates,
  new_expecvar_prefix = "auto",
  measure_sets = guess_measure_sets(.data),
  amount_scalar = NULL,
```

```
  .by = NULL,
  .keep = c("all", "used", "unused", "none"),
  .before = NULL,
  .after = NULL
)
```

Arguments

`.data`	A `base::data.frame()` that houses an experience study.
`new_expected_rates`	A numeric vector to use as the expected probability for the study's event of interest (i.e., policy lapse or insured death). This can be a column in the dataset or a new numeric vector of length 1 or `nrow(.data)`.
`new_expecvar_prefix`	A string to distinguish the new expecteds and variances columns in the dataset. To overwrite existing expecteds and variances columns, use an argument value of `NULL`, `character()`, or `''`. The default `'auto'` will add a numeric prefix based on the previous names of expecteds/variances so that names will remain unique.
`measure_sets`	A (potentially named) list of measure sets. Only need to specify once if chaining multiple `expstudy` functions as the `measure_sets` will be passed as an attribute in results.
`amount_scalar`	A numeric vector to use when determining amount-weighted expecteds and variances. The function will determine whether or not the new expecteds/variances are amount-weighted if the corresponding actuals in the study have values greater than 1 (actuals that are not amount-weighted, i.e., counts, should only be 0 or 1).
`.by`	**[Experimental]** <tidy-select> Optionally, a selection of columns to group by for just this operation, functioning as an alternative to `group_by()`. For details and examples, see ?dplyr_by.
`.keep`	Control which columns from `.data` are retained in the output. Grouping columns and columns created by `...` are always kept. • `"all"` retains all columns from `.data`. This is the default. • `"used"` retains only the columns used in `...` to create new columns. This is useful for checking your work, as it displays inputs and outputs side-by-side. • `"unused"` retains only the columns *not* used in `...` to create new columns. This is useful if you generate new columns, but no longer need the columns used to generate them. • `"none"` doesn't retain any extra columns from `.data`. Only the grouping variables and columns created by `...` are kept.
`.before, .after`	<tidy-select> Optionally, control where new columns should appear (the default is to add to the right hand side). See `relocate()` for more details.

Value

An object of the same type as `.data`. The output has the following properties:

- Columns from `.data` will be preserved according to the `.keep` argument.
- Existing columns that are modified by `...` will always be returned in their original location.
- New columns created through `...` will be placed according to the `.before` and `.after` arguments.
- The number of rows is not affected.
- Columns given the value `NULL` will be removed.
- Groups will be recomputed if a grouping variable is mutated.
- Data frame attributes are preserved.

Underlying Assumptions

This function was developed according to current industry practice relating to experience study calculations. Some of the assumptions incorporated are briefly outlined below.

1. The experience study data is at a seriatim level where repeated observations of multiple units can exist. For example, the study data can contain experience for multiple policies over multiple calendar or policy years.
2. Each decrement event can be described as a Bernoulli random variable with expected rate of decrement equal to p. Furthermore, combining multiple observation units with equal rates of decrement p can be considered a Binomial random variable with n equal to the number of observation units.
3. Decrements are considered to be uniform between observations.

With these assumptions, new expecteds that are not amount-weighted are calculated as the product of exposures and the expected decrement rate, new variances are calculated as the product of the previously calculated new expecteds and 1 minus the previously calculated new expecteds. Amount-weighted expecteds and variances follow the prior calculations and additionally multiply by the amount scalar and amount scalar squared, respectively.

For a more detailed explanation of these methods used, please refer to the Society of Actuary's publication over experience study calculations.

Naming convention

`expstudy` uses a naming convention where some functions are prefixed by the underling `dplyr` verb. The purpose of this is to associate the resulting structure of the `expstudy` function with a very similar output as what the `dplyr` function would produce. Note that the intention here is not replace all `dplyr` use cases but instead add specific functionality to streamline routine experience study analyses.

Examples

```
mortexp |>
  dplyr::mutate(
    NEW_EXPECTED_MORT_RT = runif(n = nrow(mortexp))
```

```
) |>
mutate_expecvar(
  new_expected_rates = NEW_EXPECTED_MORT_RT,
  new_expecvar_prefix = 'ADJ_',
  amount_scalar = FACE_AMOUNT
)
```

`mutate_metrics` *Add common metrics to an experience study*

Description

`mutate_metrics()` calculates metrics for an experience study using common measures associated with the data. These measures are identified via the `measure_sets` argument which can be provided directly or be guessed using regular expressions (`regexs`). See `guess_measure_sets()` for additional detail on how this guessing is implemented.

Usage

```
mutate_metrics(
  .data,
  measure_sets = guess_measure_sets(.data),
 metrics = list(AVG_OBSRV = avg_observed, AVG_EXPEC = avg_expected, CI_FCTR = ci_fctr,
    AE_RATIO = ae_ratio, CREDIBILITY = credibility),
  ...,
  .by = NULL,
  .keep = c("all", "used", "unused", "none"),
  .before = NULL,
  .after = NULL
)
```

Arguments

`.data`	A `base::data.frame()` that houses an experience study.
`measure_sets`	A (potentially named) list of measure sets. Only need to specify once if chaining multiple `expstudy` functions as the `measure_sets` will be passed as an attribute in results.
`metrics`	A named list of functions to calculate metrics. Each function will be applied to each set identified in `measure_sets`.
`...`	Additional (optional) arguments passed along to each (metric function)metrics.
`.by`	**[Experimental]** <tidy-select> Optionally, a selection of columns to group by for just this operation, functioning as an alternative to `group_by()`. For details and examples, see ?dplyr_by.

`.keep` Control which columns from `.data` are retained in the output. Grouping columns and columns created by `...` are always kept.

- `"all"` retains all columns from `.data`. This is the default.
- `"used"` retains only the columns used in `...` to create new columns. This is useful for checking your work, as it displays inputs and outputs side-by-side.
- `"unused"` retains only the columns *not* used in `...` to create new columns. This is useful if you generate new columns, but no longer need the columns used to generate them.
- `"none"` doesn't retain any extra columns from `.data`. Only the grouping variables and columns created by `...` are kept.

`.before, .after` <`tidy-select`> Optionally, control where new columns should appear (the default is to add to the right hand side). See `relocate()` for more details.

Details

This function is structured in a way that uses sets of measures within the study as the first function argument of each metric function. The default argument uses a set of metric functions, provided by `expstudy`, which are commonly requested metrics used in actuarial analyses. For convenience, a vectorized version of these default metric functions have also been provided; see metrics for more information.

Value

An object of the same type as `.data`. The output has the following properties:

- Columns from `.data` will be preserved according to the `.keep` argument.
- Existing columns that are modified by `...` will always be returned in their original location.
- New columns created through `...` will be placed according to the `.before` and `.after` arguments.
- The number of rows is not affected.
- Columns given the value `NULL` will be removed.
- Groups will be recomputed if a grouping variable is mutated.
- Data frame attributes are preserved.

Naming convention

`expstudy` uses a naming convention where some functions are prefixed by the underling `dplyr` verb. The purpose of this is to associate the resulting structure of the `expstudy` function with a very similar output as what the `dplyr` function would produce. Note that the intention here is not replace all `dplyr` use cases but instead add specific functionality to streamline routine experience study analyses.

Examples

```
# Metrics can be added at a seriatim level, but often are
# calculated after some aggregation is applied to a cohort:
mortexp |>
  dplyr::group_by(
    GENDER
  ) |>
  summarise_measures() |>
  mutate_metrics()
```

`summarise_measures` *Aggregate an experience study*

Description

`summarise_measures()` functions the same as `dplyr::summarise()` and returns a new data frame per combination of grouping variable. However, this function is is streamlined to return the sum of an experience study's measures instead of any arbitrary summary function. These measures are identified via the `measure_sets` argument which can be provided directly or be guessed using regular expressions (`regexs`). See `guess_measure_sets()` for additional detail on how this guessing is implemented.

Usage

```
summarise_measures(
  .data,
  measure_sets = guess_measure_sets(.data),
  na.rm = TRUE,
  .groups = "drop",
  .by = NULL
)
```

Arguments

`.data`	A `base::data.frame()` that houses an experience study.
`measure_sets`	A (potentially named) list of measure sets. Only need to specify once if chaining multiple `expstudy` functions as the `measure_sets` will be passed as an attribute in results.
`na.rm`	logical. Should missing values (including `NaN`) be removed?
`.groups`	**[Experimental]** Grouping structure of the result. • "drop_last": dropping the last level of grouping. This was the only supported option before version 1.0.0. • "drop": All levels of grouping are dropped. • "keep": Same grouping structure as `.data`. • "rowwise": Each row is its own group.

When .groups is not specified, it is chosen based on the number of rows of the results:

- If all the results have 1 row, you get "drop_last".
- If the number of rows varies, you get "keep" (note that returning a variable number of rows was deprecated in favor of reframe(), which also unconditionally drops all levels of grouping).

In addition, a message informs you of that choice, unless the result is ungrouped, the option "dplyr.summarise.inform" is set to FALSE, or when summarise() is called from a function in a package.

.by **[Experimental]**

<tidy-select> Optionally, a selection of columns to group by for just this operation, functioning as an alternative to group_by(). For details and examples, see ?dplyr_by.

Value

An object *usually* of the same type as .data.

- The rows come from the underlying group_keys().
- The columns are a combination of the grouping keys and the summary expressions that you provide.
- The grouping structure is controlled by the .groups= argument, the output may be another grouped_df, a tibble or a rowwise data frame.
- Data frame attributes are **not** preserved, because summarise() fundamentally creates a new data frame.

Naming convention

expstudy uses a naming convention where some functions are prefixed by the underling dplyr verb. The purpose of this is to associate the resulting structure of the expstudy function with a very similar output as what the dplyr function would produce. Note that the intention here is not replace all dplyr use cases but instead add specific functionality to streamline routine experience study analyses.

Examples

```
mortexp |>
  dplyr::group_by(
    UNDERWRITING_CLASS
  ) |>
  summarise_measures()
```

Index