Package: lossrx (via r-universe)

September 2, 2024

Title Actuarial Loss Development and Reserving with R

Version 0.0.4

Description Actuarial Loss Development and Reserving Helper Functions and ShinyApp.

License MIT + file LICENSE

URL https://github.com/jimbrig/lossrx, https://docs.jimbrig.com/lossrx

BugReports https://github.com/jimbrig/lossrx/issues

Depends R (>= 2.10)

Imports cli, config, connections, crayon, dbx, dplyr, flipTime (>= 2.9.4), formattable, fs, glue, here, lubridate, magrittr, pool, purrr, randomNames, readr, rlang (>= 0.4.11), RPostgres, shiny, stats, stringr, tibble, timevis, usethis, utils

Suggests actuar, attachment, covr, dbplyr, desc, devtools, fplot, janitor, kableExtra, knitr, pkgdown, rmarkdown, roxygen2, spelling, summarytools, testthat (>= 3.0.0), tidyr

VignetteBuilder knitr

Remotes Displayr/flipTime, rstudio/connections

Config/testthat/edition 3

Encoding UTF-8

Language en-US

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Repository https://jimbrig.r-universe.dev

RemoteUrl https://github.com/jimbrig/lossrx

RemoteRef HEAD

RemoteSha ae12502cff623998fc82a947d09716ea5e4ed84d

Contents

`claims_transactional` *claims_transactional*

Description

Transactional claims dataset. Convert to a static `lossrun` using the `loss_run()` function.

Usage

```
claims_transactional
```

Format

A `data.frame` with 80278 rows and 12 variables:

`claim_num` integer. DESCRIPTION.

`claim_id` character. DESCRIPTION.

`accident_date` double. DESCRIPTION.

`state` character. DESCRIPTION.

`claimant` character. DESCRIPTION.

`report_date` double. DESCRIPTION.

`status` character. DESCRIPTION.

payment double. DESCRIPTION.

case double. DESCRIPTION.

transaction_date double. DESCRIPTION.

trans_num integer. DESCRIPTION.

paid double. DESCRIPTION.

---

coalesce_join *Coalesce Join*

Description

Coalesce Join

Usage

```
coalesce_join(
  x,
  y,
  by = NULL,
  suffix = c(".x", ".y"),
  join = dplyr::full_join,
  ...
)
```

Arguments

x	x
y	y
by	by
suffix	suffix
join	join type
...	passed to dplyr join function

Value

a tibble

connect_db *Connect DB*

Description

Connect to the Actuarial Database Instance.

Usage

```
connect_db(pool = TRUE)
```

Arguments

pool	Logical - should the connection be pooled?

Value

a database connection

create_tbl *Create Table*

Description

Creates a database table given a connection, table name, and:

- SQL file specifying the table's schema
- CSV file to seed the table's values

Usage

```
create_tbl(
  conn,
  tbl_name,
  csv_path = "data-raw/database/CSV",
  sql_path = "data-raw/database/SQL",
  drop_if_exists = TRUE
)
```

Arguments

conn	database connection
tbl_name	character string representing table name
csv_path	base path (excluding file) to the CSV file
sql_path	base path (excluding file) to the SQL file
drop_if_exists	Should the table be dropped (with CASCADE) if it already exists?

Value

the created database table returned as an R `data.frame()`.

Note

It is assumed that the `tbl_name` mirrors both the basename of the CSV file and the SQL file (excluding extensions).

---

data-utils *Data Manipulation Utilities*

Description

Data Manipulation Utilities

Pull Unique Values from a dataframe

Usage

```
pull_unique(df, var)
```

Arguments

`df`	a provided `data.frame()`
`var`	character/numeric - quoted named of a variable from `df` or its numerical index.

Value

- `pull_unique` returns a character vector of unique, sorted values from specified column

Examples

```
df <- data.frame(let = rep(letters, 2), num = rep(c(1:26), 2))
pull_unique(df, 1)
pull_unique(df, "num")
```

dates *Date Utility Functions*

Description

A set of helper functions for dealing with dates in a typical actuarial analysis context.

Derive the number of months elapsed between two dates.

Usage

```
end_of_month(date)

start_of_month(date)

extract_date(string)

elapsed_months(end_date, start_date)
```

Arguments

date	character or date representation of a date.
string	string to extract a date from
end_date	end date
start_date	start date

Value

- `end_of_month` returns last day of the month given.
- `start_of_month` returns the first day of the month given.
- `extract_date` returns a date object extracted from the provided string.

numeric

See Also

`as.Date()`, `lubridate::ceiling_date()`, `lubridate::floor_date()`, `flipTime::AsDate()`

Examples

```
# character input
start_of_month("2020-08-13")
end_of_month("2017-10-20")

# can handle human-readable dates also
start_of_month("July 7, 1999")
end_of_month("February 5, 2019")
```

```
# date input
start_of_month(as.Date("2020-08-13"))
end_of_month(as.Date("2020-10-20"))
```

`doc_data` *Document Datasets*

Description

Creates skeleton to document datasets via `roxygen2`.

Usage

```
doc_data(
  obj,
  title = deparse(substitute(obj)),
  description = "DATASET_DESCRIPTION",
  write_to_file = TRUE,
  ...
)
```

Arguments

`obj`	object to document
`title`	Title
`description`	Description
`write_to_file`	Logical
`...`	N/A

Value

silently returns the doc_string

Examples

```
library(lossrx)
data(losses)
string <- doc_data(losses, "Loss Data", "Claims Data", FALSE)
cat(string)
```

exposures *exposures*

Description

Exposure data.

Usage

```
exposures
```

Format

A `data.frame` with 855 rows and 5 variables:

`member` character. DESCRIPTION.

`program_year` double. DESCRIPTION.

`department` character. DESCRIPTION.

`payroll` double. DESCRIPTION.

`miles` double. DESCRIPTION.

extract_num *Extract numbers from a string*

Description

Extract numbers from a string

Usage

```
extract_num(string)
```

Arguments

`string`	String to pull numbers from

Value

String of numbers

feedback *Feedback Message Utility Functions*

Description

A set of helper functions for providing verbose feedback to the developer using this packages functions.

Usage

```
msg_field(x)

msg_value(x)

msg_done(x)

msg_bullet(x, bullet = cli::symbol$bullet)

msg_err(x)

msg_path(x)

msg_info(x)

msg_code(x)

msg_feedback(x)
```

Arguments

x	The string passed to various msg_ functions.
bullet	What to use for the message's bullet. Defaults to cli::symbol$bullet

See Also

- usethis::ui-questions()
- cli::list_symbols()

Other Feedback Utilities: indent(), inform()

indent *Indent*

Description

Indentation around various `msg_` feedback functions.

Usage

```
indent(x, first = "  ", indent = first)
```

Arguments

`x`	The string passed to various `msg_` functions.
`first`	what to indent with - defaults to `" "`.
`indent`	indentation of next line - defaults to `first`

Value

string

See Also

Other Feedback Utilities: `feedback`, `inform()`

inform *Inform*

Description

A wrapper around `rlang::inform()` for providing feedback to developers using this packages functions.

Usage

```
inform(...)
```

Arguments

`...`	Passed to `rlang::inform()`

Value

feedback in console

See Also

`rlang::inform()`

Other Feedback Utilities: `feedback`, `indent()`

`interp` `interp` - *Actuarial Interpolation*

Description

Interpolate Cumulative Loss Development Factors (CDFs).

Usage

```
interp(new_age, cdf_array, age_array, cutoff = 450, method = 3)

interp.dblexp(new_age, age_high, age_low, cdf_high, cdf_low)

interp.exp(new_age, age_high, age_low, cdf_high, cdf_low)

interp.linear(new_age, age_high, age_low, cdf_high, cdf_low)
```

Arguments

`new_age`	integer - value of the new age whose CDF is to be interpolated
`cdf_array`	numeric vector of CDFs (usually representative of the selected factors)
`age_array`	numeric vector of ages corresponding to the supplied `cdf_array`
`cutoff`	the largest possible age, after which, no interpolation is performed
`method`	integer - must be 1, 2, or 3 where 1 represents linear, 2 represents exponential, and 3 represents double exponential. Defaults to 3, but falls back onto 1 if necessary.
`age_low`, `age_high`	Low and High ages
`cdf_low`, `cdf_high`	Low and High CDFs

Details

This generic function comes with three possible `methods`:

1. Linear Interpolation
2. Exponential Interpolation
3. Double Exponential Interpolation

Actuaries often have to *interpolate* values in-between the selected Loss Development Factors (LDFs) / Cumulative Loss Development Factors (CDFs) in order to derive development factors at a variety of possible ages of maturity, outside the scope of the selected factors by the actuary.

For example, an actuary will select factors by maturity or development age in months using actuarial triangles. Due to the fact the actuarial selections are limited to the maturities present in the triangle (i.e. ages 12, 24, etc.), the factors for ages before, after, and in-between the selection ages must be *interpolated*.

A comprehensive approach to deriving the interpolated values would follow a pattern similar to the following:

- For ages of maturity `<= First Selected Age of Maturity` (i.e. `<= 12`), factors are derived using *persistencies*. A persistency is simply a percentage value representing the percent paid/reported at a given age compared to the age's ceiling and floor. For example, a persistency as of age 3 would represent the percent paid/reported at 3 months of development out of the total percent paid/reported at 12 months of development. The persistency as of age 15 would represent the percent paid/reported at age 15 compared to the total percent paid/reported between ages 12 and 24.
- For ages of maturity `Selected Age of Maturity Floor <= x <= Selected Age of Maturity Ceiling`, i.e. *in-between* ages, the factors are derived using *double-exponential* interpolation using the selections at the floor and ceiling ages.
- For ages of maturity `>= Last Selected Age of Maturity` (i.e. `>= 106`), a *decay factor approach* is used to *decay* the final selected factor across the ages beyond that final age.

Value

derived numeric value for the supplied `new_age`'s CDF

Functions

- `interp.dblexp()`: Double Exponential Interpolation
- `interp.exp()`: Exponential Interpolation
- `interp.linear()`: Linear Interpolation

Examples

```
cdfs <- c(3.579, 2.866, 2.489, 2.121, 1.876, 1.543, 1.222, 1.150, 1.109, 1.005, 1.0025)
ages <- seq(from = 12, to = (length(cdfs) * 12), by = 12)

interp(14, cdfs, ages)
interp(12, cdfs, ages) == cdfs[[1]]
interp(27, cdfs, ages, method = 2)
```

losses *losses*

Description

losses

Usage

```
losses
```

Format

A data.frame with 79748 rows and 30 variables:

eval_date double. DESCRIPTION.

devt_age double. DESCRIPTION.

occurrence_number character. DESCRIPTION.

coverage character. DESCRIPTION.

member character. DESCRIPTION.

program_year character. DESCRIPTION.

loss_date double. DESCRIPTION.

rept_date double. DESCRIPTION.

hire_date double. DESCRIPTION.

report_lag double. DESCRIPTION.

report_lag_group integer. DESCRIPTION.

day_of_week character. DESCRIPTION.

claim_type character. DESCRIPTION.

claimant_state character. DESCRIPTION.

loss_state character. DESCRIPTION.

cause character. DESCRIPTION.

department character. DESCRIPTION.

tenure double. DESCRIPTION.

tenure_group integer. DESCRIPTION.

claimant_age double. DESCRIPTION.

claimant_age_group integer. DESCRIPTION.

driver_age double. DESCRIPTION.

driver_age_group integer. DESCRIPTION.

status character. DESCRIPTION.

total_paid double. DESCRIPTION.

total_incurred double. DESCRIPTION.

count double. DESCRIPTION.

open_count double. DESCRIPTION.

close_count double. DESCRIPTION.

incurred_group integer. DESCRIPTION.

---

loss_run *loss_run*

Description

view losses as of a specific date

Usage

```
loss_run(val_date, trans_dat)
```

Arguments

val_date	date the valuation date of the loss run. Claim values from trans will be values as of the val_date
trans_dat	data frame of claims transactions

Value

data frame of claims (1 claim per row) valued as of the val_date

---

open_pkgdown *Open pkgdown site of the package*

Description

Open pkgdown site of the package

Usage

```
open_pkgdown()
```

Examples

```
# open_pkgdown()
```

simulate_claims *simulate_claims*

Description

A function to simulate *transactional* actuarial claims/loss data for Property Casualty Insurance.

Usage

```
simulate_claims(
  n_claims = 1000,
  start_date = "2015-01-01",
  end_date = Sys.Date(),
  seed = 12345,
  loss_distribution = "lnorm",
  params = list(mean_log = 7.5, sd_log = 1.5),
  status_prob_open = 0.96,
  cache = FALSE,
  ...
)
```

Arguments

`n_claims`	Numeric - Number of claims to be simulated.
`start_date, end_date`	Character/Date - Start and End dates for simulation to create claims within (experience_period).
`seed`	Numeric - the seed is used to isolate randomness during statistical simulations.
`loss_distribution`	Character - must be one of the distributions mentioned in the details below. Defaults to lognormal.
`params`	Parameters associated with the specified `loss_distribution` in a list (i.e. `list(mean_log = 7.5, sd_log = 1.5)` for lognormal distribution).
`status_prob_open`	Numeric - must be within `0 < x < 1` and represents probability a claim is open when running binomial simulations for claims' status.
`cache`	Boolean/Logical - enable caching?
`...`	If needed

Details

Severity/Loss Distributions:

- Normal: `norm`
- Lognormal: `lnorm`

- Gamma: `gamma`
- LogGamma: `lgamma`
- Pareto: `pareto`
- Weibull: `weibull`
- Generalized Beta: `genbeta`

Value

The return value, if any, from executing the function.

`toproper` *To Proper*

Description

To Proper

Usage

```
toproper(
  string,
  replace_underscores = TRUE,
  underscore_replacement = " ",
  return_as = c("titlecase", "uppercase", "lowercase", "asis"),
  uppers = c("Tpa")
)
```

Arguments

`string`	string to manipulate on
`replace_underscores`	Logical: if `TRUE` replaces all underscores with specified `underscore_replacement` argument's value.
`underscore_replacement`	Character: if argument `replace_underscores` equals `TRUE`, will replace all "_"'s with specified string.
`return_as`	How should the string be returned? Options are: • "titlecase": Applies `stringr::str_to_title`. • "uppercase": Applies `toupper`. • "lowercase": Applied `tolower`. • "asis": No manipulation. Returns as is.
`uppers`	Abbreviations to keep upper-case.

Value

"Proper" string

See Also

str_replace.

Examples

```
s <- "variable_a is awesome"
toproper(s)
```

`view_claim_history` *View an Individual Claim's History*

Description

This is a comprehensive function that allows the user to gain valuable insights about an individual claim's historical transactions.

Usage

```
view_claim_history(claim_id, claims_data = NULL)
```

Arguments

`claim_id`	Claim ID
`claims_data`	Dataset

Value

a list containing a) claim details b) transactional history c) interactive timeline

Index