Module 2 – Earned Premium

Actuarial Data Manipulation with R – CAS Spring Meeting 2024

Denis Dreano

2024-05-05

Content

Goal: Calculate the earned premium displayed in the rate indication, from the policy data.

Content:

- Loading and exploring tabular data (data frames)
- Data manipulations with `dplyr`:
  - `filter()`
  - `group_by()`
- Manipulating dates with `lubridate`
- The pipe operator `|>`
- Iterating calculation with `purrr`
  - `map()`
  - `map_dbl()`
  - `map_dfr()`

Loading Policy Data

```
library(readr)
policy_data <- read_csv("../Data/policy_table.csv")
policy_data
```

```
# A tibble: 109,987 x 6
   policy_id      inception_date expiration_date n_expo  year premium
   <chr>          <date>         <date>           <dbl> <dbl>   <dbl>
 1 policy_2004_1  2004-01-01     2004-12-31           1  2004   1796.
 2 policy_2004_2  2004-01-01     2004-12-31           1  2004   1796.
 3 policy_2004_3  2004-01-01     2004-12-31           1  2004   1796.
 4 policy_2004_4  2004-01-01     2004-12-31           1  2004   1796.
 5 policy_2004_5  2004-01-01     2004-12-31           1  2004   1796.
 6 policy_2004_6  2004-01-01     2004-12-31           1  2004   1796.
 7 policy_2004_7  2004-01-01     2004-12-31           1  2004   1796.
 8 policy_2004_8  2004-01-01     2004-12-31           1  2004   1796.
 9 policy_2004_9  2004-01-01     2004-12-31           1  2004   1796.
10 policy_2004_10 2004-01-01     2004-12-31           1  2004   1796.
# i 109,977 more rows
```

Exploring Data

Other commands to have a quick look at the data:

```
str(policy_data)

summary(policy_data)

library(dplyr)
glimpse(policy_data)
```

Manipulating Data with `dplyr`: `filter()`

Use the `filter()` function from the `dplyr` package to filter rows based on a condition.

```
library(dplyr); library(lubridate)

# Filter the data frame for a specific policy type
policy_data_2010 <- filter(policy_data, year(inception_date) == 2010)

# Filter the data frame for policies with a premium greater than 1000
policy_data_high_premium <- filter(policy_data, premium > 1500)

# Combine multiple conditions with logical operators
policy_data_combined <- filter(
  policy_data, year(inception_date) & premium > 1000
)
```

Manipulating Data with dplyr: group_by()

Use the group_by() function from the dplyr package to group rows based on a variable.

```
# Group the data frame by policy type
policy_data_grouped <- group_by(policy_data, year(inception_date))

# Summarize the data by group
policy_data_summary <- summarize(policy_data_grouped, avg_premium = mean(premium))

head(policy_data_summary)
```

```
# A tibble: 6 x 2
  `year(inception_date)` avg_premium
                   <dbl>       <dbl>
1                   2004       1796.
2                   2005       1796.
3                   2006       1792.
4                   2007       1917.
5                   2008       1729.
6                   2009       1393.
```

One can also group and mutate, or group and filter.

Manipulating Dates with `lubridate`

Create a date object:

```
library(lubridate)
inception_date <- ymd("2022-01-01")
inception_date <- mdy("12/31/2022")
```

Extract components of a date

```
year(inception_date)
```

```
[1] 2022
```

```
month(inception_date)
```

```
[1] 12
```

```
day(inception_date)
```

```
[1] 31
```

Add or subtract time intervals

```
inception_date + years(1)
```

```
[1] "2023-12-31"
```

```
inception_date - days(7)
```

```
[1] "2022-12-24"
```

Calculate the difference between two dates:

```
diff_days <- (today() %--% inception_date) / days(1)
```

The Pipe Operator

- The pipe operator |> allows you to chain multiple operations together in a readable way.

```
# without pipe
policy_data_grouped <- group_by(policy_data, year(inception_date))
policy_data_summary <- summarize(policy_data_grouped, avg_premium = mean(premium))

# with pipe
policy_data_summary <- policy_data |>
  group_by(year(inception_date)) |>
  summarize(avg_premium = mean(premium))
```

Iterating with purrr: map(), map_dbl(), map_dfr()

The purrr package provides functions for iterating over data structures.

```
library(purrr)
map(c(1, 2, 3), function(x) x * 2) |> str()
```

```
List of 3
 $ : num 2
 $ : num 4
 $ : num 6
```

```
map_dbl(c(1, 2, 3), function(x) x * 2)
```

```
[1] 2 4 6
```

```
map_dfr(c(1, 2, 3), function(x) tibble(x = x, y = x * 2))
```

```
# A tibble: 3 x 2
      x     y
  <dbl> <dbl>
1     1     2
2     2     4
3     3     6
```

References

- Cheat sheets in `References/`
  - `data-import.pdf`
  - `data-transformation.pdf`
  - `lubridate.pdf`
  - `purrr.pdf`
- Wickham, H. et al. *R for Data Science* (r4ds.hadley.nz)
  - Chap. 17 Dates and Times
  - Chap. 3 Data Transformation
  - Chap. 4 Workflow Style
- Wickham, H. *Advanced R* (adv-r.hadley.nz)
  - Chap. 9 Functionals