Article from

The Modeling Platform

August 2020

Tidy Data Formats, Part 2

Applications of Tidy Mortality Tables

By Matthew Caseres

In this article we continue our work on tidy mortality tables by creating a format that can join mortality rates to data sets in a single line of code. For purposes of this demonstration, I have combined these mortality rates with lapse rates and demographics from a 2014 RGA study in order to create a semirealistic data set. This data set is then used for a Monte Carlo simulation to generate a distribution of claim amounts. I then discuss the creation of open-source implementations of regulations as well as trade-offs between the freedom to use actuarial judgment and consistency of results across companies.

R code snippets are included for the discussion of simulating claim amounts. Any programming language can be used for the tasks described here, but a language meant for data work, like R or Python, will provide constructs that make tasks easier.

TIDY COMBINED SELECT AND ULTIMATE MORTALITY RATE TABLES

In Part 1 of this series, published in the April 2020 issue of *The Modeling Platform*, I wrote a script that allowed me to reformat hundreds of select mortality tables into a tidy format. I now create a representation of a mortality basis that can be attached to a data set in a single line of code. See that Figure 1 is a tidy mortality table that contains select and ultimate mortality rates for each combination of issue age and duration up to the end of the mortality table.

Figure 1
Tidy Mortality Table With Select and Ultimate Mortality Rates

issue_age	duration	attained_age	q_sel	q_ult	q
60	60	119	*NA*	0.50000	0.50000
60	61	120	*NA*	0.50000	0.50000
61	1	61	0.00152	0.00772	0.00152
61	2	62	0.00310	0.00860	0.00310
61	3	63	0.00525	0.00957	0.00525

Use in Experience Studies

Once this table has been created, I am able to easily join mortality rates using a multi-key join, as demonstrated in Figure 2.

Storage

We currently use mortality tables from a web page maintained by the Society of Actuaries (SOA). However, I believe storing common tables as CSV files in cloud file storage would be easier to

Figure 2
Joining the Tidy Mortality Table to Experience Data

```
exposures %>% inner_join(mortality, by = c('issue_age', 'duration', 'gender'))
```

exposures

issue_age	duration	gender	exposures	deaths
35	4	Male	1	0
35	5	Male	1	0
35	6	Male	1	1

mortality

q	issue_age	duration	gender
0.00021	35	1	Male
0.00029	35	2	Male
0.00043	35	3	Male
0.00050	35	4	Male
0.00058	35	5	Male
0.00066	35	6	Male
0.00075	35	7	Male

after join

issue_age	duration	gender	exposures	deaths	q
35	4	Male	1	0	0.00050
35	5	Male	1	0	0.00058
35	6	Male	1	1	0.00066

maintain and use than the existing solution. I have created a OneDrive directory as an example but do not plan on actively maintaining the reformatted tables. I hope that others working in this area can extend this work into something that is well maintained.

I investigated Google BigQuery (a serverless analytics data warehouse) as a storage option but found that the platform is meant to store large data sets for analytics, rather than collections of tables. Tables are not discoverable in BigQuery, since there is no graphical representation of a directory of files.

Simulating Data

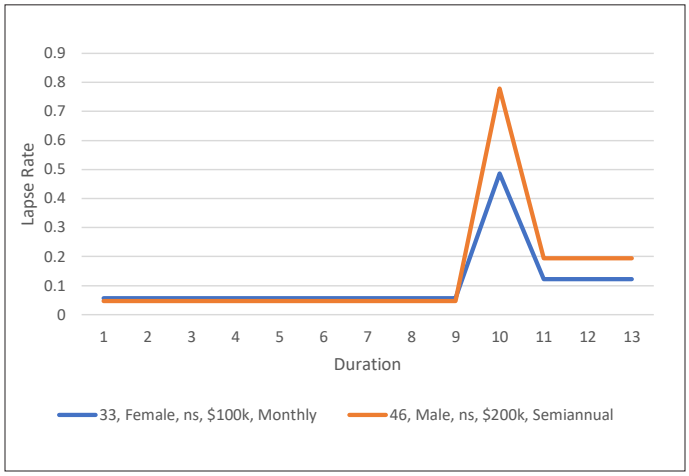
As I am unaware of any data sets that can be used to demonstrate methods related to lapses or premium patterns, I created a simulated data set that contains decrements from both lapse and mortality to aid in the demonstration of methods.

Methods

The data created is for a 10-year term life product with decrements due to lapse and death. The approach is to simulate a random number indicating a death when the simulated number is less than the mortality rate. I simulated another random number to determine if the policy has lapsed within a given year. It is possible that a policy would both lapse and die within a given year, in which case I used a tie-breaker random number that assigns the policy as lapsing or dying, each with a probability of 0.5. If the policy neither lapsed nor died, I moved to the next time step and ran the simulation again with rates for the new attained age. The simulation returns the year and cause of decrement.

My previous work on tidy mortality rates allowed for extraction of mortality rates up to the end of the mortality table for a given issue age, gender and underwriting classification. For lapse rates in duration 10, I used the general linear models factors from page 108 of the 2014 Report on the Lapse and Mortality Experience of Post-Level Premium Period Term Plans by RGA. Durations past the 10th duration were assigned one-fourth the duration 10 rate, and durations before the 10th were determined using a set of factors that I thought would give the data a reasonable shape. Figure 3 shows the lapse rates for two simulated policies.

Figure 3
Lapse Rate by Issue Age, Gender, Underwriting, Face Amount, Premium Mode



The gender, underwriting, face amount and payment mode were simulated using the data proportions given in the previously referenced RGA study. Issue ages were randomly assigned to ages from 30 to 55.

Figure 4 shows an example of what each simulated record looks like at this point.

To make this a VM-51 format, I randomly selected an issue date between Jan. 1, 1990, and Jan. 1, 2020, and randomly selected a

Figure 4
Example Simulated Records

Issue Age	Gender	Underwriting	Face Amount	Premium Mode	Decrement Year	Decrement Cause
35	Male	Nonsmoker	$100,000	Monthly	9	Lapse

termination date from the year of decrement. When the simulated termination date was past the assumed current date of Jan. 1, 2020, I changed the decrement cause to "Truncated" to indicate that at the date of analysis I did not know the cause or time of decrement. A date of Jan 1., 2020, was assigned to the termination date field for truncated data. Now our data has an issue date and termination date as required.

You can access the 100,000 simulated VM-51 format policy records in BigQuery:

*SELECT * FROM `soa-mortality-demo.tidy_mortality.VM_51`*

MONTE CARLO MORTALITY DISTRIBUTION IMPLEMENTATION

I believe that the best way to communicate methods is to produce open-source code. Communicating methods in natural language can lead to differences in results between companies.

Method for Simulating Deaths

I used the simulated data and the expstudies R package to create a data set with rows for each policy year. Taking the 100,000 simulated VM-51 records from our previous discussion, I generated 752,990 intervals representing policy years for each policyholder.

```
expstudies::addExposures(records)
```

Some manipulations are performed to put the data in the mortality study format described in the SOA publication on Experience Study Calculations. In the SOA publication, exposures depend on whether the year is a leap year or not. This has the consequence of inflating mortality in leap years by a factor of 366/365 compared to non-leap years. In my calculation I considered 365.25 days to be an exposure. I do not expect choices on this matter to have a material impact.

Because I had split the data into policy years, I could join the corresponding mortality rate for the record's smoker status, gender, attained age and duration from the tidy VBT2015. Again, we see the utility of having mortality tables in a tidy format. A column of random numbers is generated and used to simulate deaths.

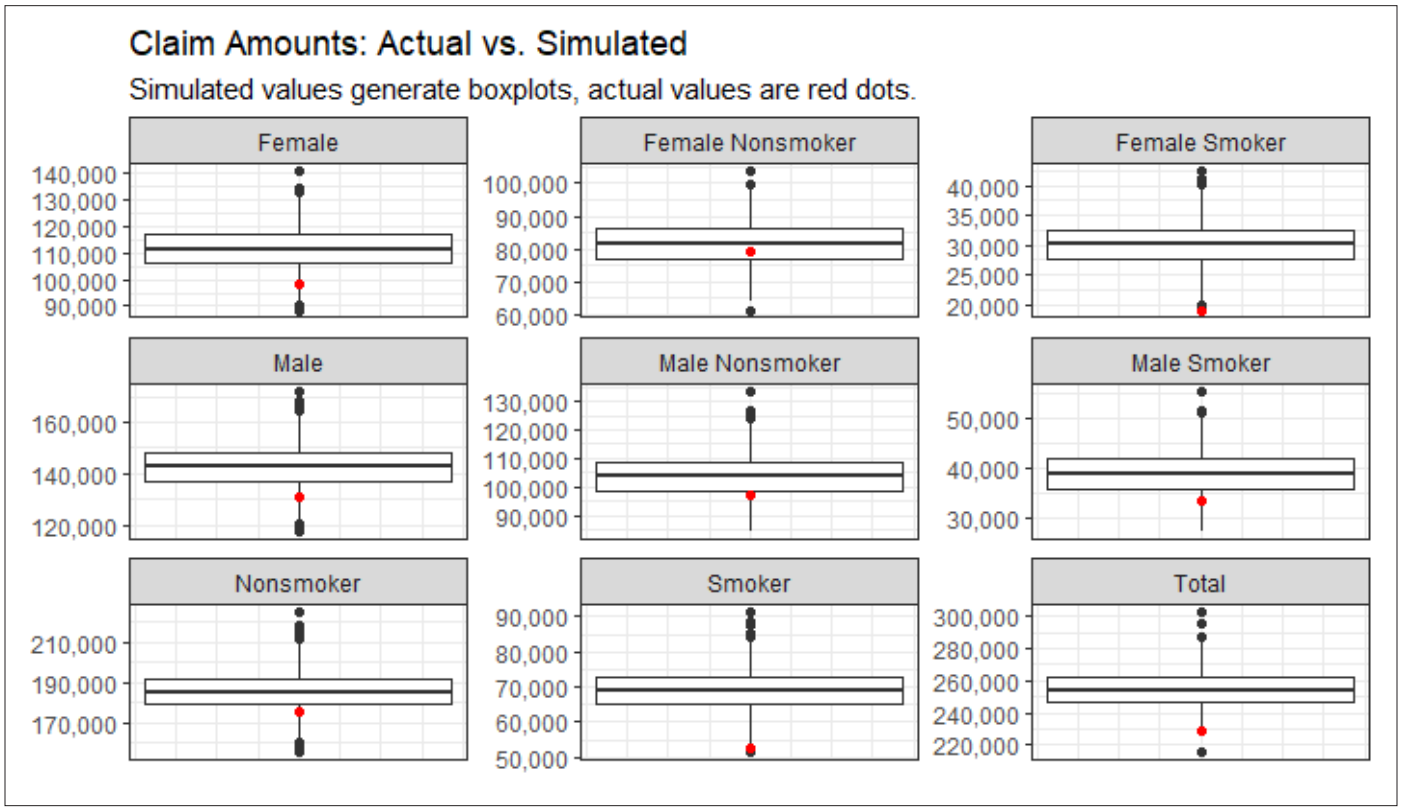
```
#mortality_data is modified as described in Experience Study Calculations SOA publication.

mortality_with_table <- mortality_data %>%
  inner_join(VBT2015),
             by = c('tobacco', 'gender', 'attained_age', 'duration'))

mortality_with_table$rand = runif(nrow(mortality_with_table))

mortality_with_table <- mortality_with_table %>%
      mutate(sim_death = if_else(rand <= q, 1, 0))
```

I performed this simulation 1,000 times and generated a distribution of results. It may be desirable to aggregate the results at varying levels of granularity. I would recommend aggregating the data at the most granular level and summing these results to produce the less granular results. This avoids performing multiple aggregations on the data set. Since I used random numbers in the simulation, I set a seed to ensure that others can run the code and get the same results.

```
set.seed(1)

female_ns_col <- rep(0, 1000)
female_s_col <- rep(0, 1000)
male_ns_col <- rep(0, 1000)
male_s_col <- rep(0, 1000)
female_col <- rep(0, 1000)
male_col <- rep(0, 1000)
ns_col <- rep(0, 1000)
s_col <- rep(0, 1000)
```

```
total_col <- rep(0, 1000)

for(i in 1:10000) {

  mortality_with_table$rand = runif(nrow(mortality_with_table))
  mortality_with_table <- mortality_with_table %>% mutate(sim_death = if_else(rand <= q*-
exposure, 1, 0))

  #aggregate at most granular basis of analysis
  single_simulation <- mortality_with_table %>%
    group_by(gender, tobacco_mapped) %>%
    summarise(sim_death_amt = sum(sim_death*face))

  female_ns <- single_simulation[1,3] %>% unlist()
  female_s <- single_simulation[2,3] %>% unlist()
  male_ns <- single_simulation[3,3] %>% unlist()
  male_s <- single_simulation[4,3] %>% unlist()

  female_ns_col[i] <- female_ns
  female_s_col[i] <- female_s
  male_ns_col[i] <- male_ns
  male_s_col[i] <- male_s
  female_col[i] <- female_ns + female_s
  male_col[i] <- male_ns + male_s
  ns_col[i] <- female_ns + male_ns
  s_col[i] <- female_s + male_s
  total_col[i] <- female_ns + male_ns + female_s + male_s
}

results <- tibble(female_ns = female_ns_col,
       female_s = female_s_col,
       male_ns = male_ns_col,
       male_s = male_s_col,
       female = female_col,
       male = male_col,
       ns = ns_col,
       s= s_col,
       total = total_col)
```

Figure 5
Boxplots Showing Actual vs. Simulated Claim Amounts



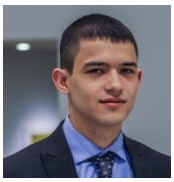
Displaying Results

In Figure 5 I have created boxplots from the distribution of simulated claim amounts and show a red dot that represents the actual outcome of the simulated data. This is like the Monte Carlo testing done in the National Association of Insurance Commissioners Model Regulation XXX and I used it to see if the simulated data has a mortality rate consistent with the mortality rate used to generate the data.

CONCLUSION AND THOUGHTS

I have shown how to simulate data and use it to communicate actuarial methods. There are many natural language documents that ultimately serve as specifications for software. I often find myself confused about how exactly I should implement something, which is why I advocate for implementations when giving technical specifications.

Open-source implementations can serve as guidelines for what to do but would not seek to replace the law. I can imagine the specific details of a company would require the freedom to exercise judgment. Freedom to exercise judgment comes at a price, though. The more freedom granted in structuring assumptions, the more potential variance there is in results due to differences in methods. ■



Matthew Caseres, ASA, is a master's student in computer science at Georgia Institute of Technology. He can be reached at *matthewcaseres@outlook.com* and on GitHub at *github.com/ActuarialAnalyst.*