Expanding Exposures

Philip Adams

2024-04-24

This document will show how to expand a list of policies that includes a termination date into a listing of associated exposures, broken by calendar period and policy period. For our purposes, we set both calendar and policy periods to be 12 months.

I only build an exposure basis for mortality. The "option" to exercise a claim can occur at any time, so exposure is uniformly spread out.

For persistency for premiums payable on a due date, the option to stop payment is usually only on a premium due date. Thus, the exposure is technically concentrated on that day. This is handled separately from surrenders, which can occur at any time.

Setup

R

```r
library(data.table)
library(arrow)
library(tidyverse)
library(parallel)
library(doParallel)
library(reticulate)

nPolicyCensusSize <- 2000000
arrIssueYearRange <- 2041:2054

study_start <- as.Date("2041-01-01")
study_end <- as.Date("2053-12-31")

pol_period_granularity <- 12 # months per policy period
```

```r
cal_period_granularity <- 12 # months per calendar period

cal_yr_breaks <- (study_start %m+% days(-1)) %m+%
  months(
    cal_period_granularity*(1:(interval(study_start %m+% days(-1),study_end) %/%
                                 months(cal_period_granularity)))
  )

arrow::read_parquet(file="policy_pop.parquet") %>%
  data.table() ->
  policy_pop

policy_pop[,Prem_Mode_Months:=sapply(prem_mode,
                                     FUN=\(x) switch(x,A=12,Q=3,M=1),
                                     USE.NAMES=F)]

source('LoadAssumptions.R')
```

Python

```python
nPolicyCensusSize = 2000000
arrIssueYearRange = range(2041,2055)

import pandas as pd
import pyarrow as pa
import pyarrow.dataset as ds
import pyarrow.parquet as pq
import numpy as np

from dateutil.relativedelta import *
from dateutil.rrule import *
from dateutil.parser import *
from datetime import *

import session_info

rng_seed = 0xBEEF
rng = np.random.default_rng(rng_seed)

studyStartDate = datetime(2041,1,1)
```

```
studyEndDate = datetime(2053,12,31)

pol_period_granularity = 12 # months per policy period
cal_period_granularity = 12 # months per calendar period

def months_between(date1, date2, exact=False):
  months_bn = (date2.year - date1.year)*12 + date2.month - date1.month

  if exact:
    months_bn += date2.day/((date2 + relativedelta(day=31)).day) - \
      date1.day/((date1 + relativedelta(day=31)).day)

  return months_bn

studyStartDateMinus1 = studyStartDate + relativedelta(days=-1)
studyMonths = np.floor(months_between(studyStartDate, studyEndDate) / 12)

cal_yr_breaks = ([studyStartDate + relativedelta(days=-1,months=i)
      for i in np.arange(start=1,stop=studyMonths + 1,step=1)*cal_period_granularity])

policy_pop = pq.read_table('policy_pop.parquet').to_pandas()

policy_pop['Prem_Mode_Months'] = policy_pop['prem_mode'].map({
  'M' : 1,
  'Q' : 3,
  'A' : 12
})

policy_pop['Issue_Date'] = pd.to_datetime(policy_pop['Issue_Date'])
policy_pop['Term_Date'] = pd.to_datetime(policy_pop['Term_Date'])

policy_pop_sample = policy_pop.head(1000).copy()
```

The Exposure Building Function

First, we need to determine the upper and lower bounds of the study periods.

- Set the *first date* to be the later of the issue date and study start date.
- Set the *last date* to be the earlier of the termination date, if valid, and study end date.
- Calculate the *full policy periods* as the exact months between the issue date and last date.

- Calculate the *pre-study periods* as the exact months between the issue date and the *first date*.

R

```
.data %>%
  mutate(
    first_date=as.Date(
      pmax({{.issue_date}},.exp_period_start)
    ),
    last_date=as.Date(
      pmin(replace_na({{.term_date}},.exp_period_end),.exp_period_end)
    ),
    full_pol_periods = interval({{.issue_date}},last_date) /
      months(.pol_period_granularity),
    prestudy_pol_periods = interval({{.issue_date}},first_date) /
      months(.pol_period_granularity)
  ) ->
  .data
```

Python

```
policy_pop_sample['first_date'] = ([max([d,studyStartDate]) for d
  in policy_pop_sample['Issue_Date']])
policy_pop_sample['last_date'] = ([min([d,studyEndDate]) for d in
  policy_pop_sample['Term_Date'].fillna(studyEndDate)])

policy_pop_sample['full_pol_periods'] = policy_pop_sample.apply(lambda x: \
  months_between(x['Issue_Date'],x['last_date'],True) / \
  pol_period_granularity,axis=1)
policy_pop_sample['prestudy_pol_periods'] = policy_pop_sample.apply(lambda x: \
  months_between(x['Issue_Date'],x['first_date'],True) / \
  pol_period_granularity,axis=1)

policy_pop_sample
```

```
  Sex  Issue_Age  ...  full_pol_periods  prestudy_pol_periods
0   F         37  ...         -0.658602                   0.0
1   M         34  ...          4.768817                   0.0
```

```
2     F          38  ...          1.870690                        0.0
3     M          34  ...          1.000000                        0.0
4     M          47  ...         10.201613                        0.0
..   ..         ...  ...               ...                        ...
995   M          32  ...          9.416667                        0.0
996   M          52  ...          9.225806                        0.0
997   M          46  ...          1.082258                        0.0
998   M          32  ...          1.165143                        0.0
999   F          41  ...          6.500000                        0.0

[1000 rows x 30 columns]
```

Next, create the modal sequence for a policy. This is just a sequence of numbers which will be cross-joined into the policy census to build out the mode-aversaries.

R

```r
modal_sequence <- pol_period_granularity*as.numeric(
  seq(to=.data[,ceiling(max(full_pol_periods))])
)
modal_sequence
```

Python

```python
modal_sequence = pol_period_granularity*np.arange(1, \
  np.ceil(policy_pop_sample['full_pol_periods'].max())+1, \
    dtype=np.int32)
modal_sequence
```

```
array([ 12,  24,  36,  48,  60,  72,  84,  96, 108, 120, 132, 144, 156],
      dtype=int32)
```

We have reached the part where we develop the frame of exposures. The expansion is carried out in three parts:

1. Create the sequence of policy mode-aversaries and save the day before.
2. Add the termination dates.
3. Create the sequence of calendar period “anniversaries”.

4. Bind them all together. The foregoing dates become the exposure ending period (*exp_period_end*).
5. Sort by policy ID and exposure ending period.
6. Set the field for exposure period start, *exp_period_start*, to the day after the exposure period end of the prior record. If there is no prior record, set it to the *first date*.
7. Filter as needed at any appropriate point to ensure that the exposure periods are within the pre-defined study start and end dates.

Policy Mode-aversaries

Steps for this stage:

1. Select only the needed columns.
2. Cross-join the modal sequence.
3. Filter out modes which aren't in the study.
4. Align only to premium due date modes.
5. Set the ending experience period to be the day before the mode-aversary.
6. Select the needed columns, and store.

R

```r
.data %>%
  select({{.ID}},
         prestudy_pol_periods,
         full_pol_periods,
         {{.prem_mode_months}},
         {{.issue_date}},
         first_date) %>%
  cross_join(y=data.table(monthaversary=modal_sequence)) %>% # Modal cross-join
  filter(monthaversary >= ceiling(prestudy_pol_periods*.pol_period_granularity) &
           monthaversary <= floor(full_pol_periods*.pol_period_granularity)) %>%
  filter(monthaversary %% {{.prem_mode_months}} == 0) %>%
  mutate(exp_period_end = {{.issue_date}} %m+% months(monthaversary) %m+%
           days(-1)) %>%
  select({{.ID}},first_date,monthaversary,exp_period_end,{{.issue_date}}) ->
  stage1a
```

Python

```python
stage1a = (
  policy_pop_sample[
    ['PolID','prestudy_pol_periods','full_pol_periods','Prem_Mode_Months', \
      'Issue_Date','first_date']
    ].merge(
      pd.DataFrame(data={'monthaversary' : modal_sequence}),
      how='cross'
      ).query(
        'monthaversary >= ceil(prestudy_pol_periods*@pol_period_granularity)'
        ).query('monthaversary <= floor(full_pol_periods*@pol_period_granularity)')
          .query('monthaversary % Prem_Mode_Months == 0')
)

stage1a['exp_period_end'] = stage1a.apply(lambda x: x['Issue_Date'] + \
  relativedelta(days=-1, months=x['monthaversary']),axis=1)

stage1a = stage1a[['PolID','first_date','monthaversary','exp_period_end','Issue_Date']]

stage1a
```

```
       PolID first_date  monthaversary exp_period_end Issue_Date
13         2 2049-03-24             12     2050-03-23 2049-03-24
14         2 2049-03-24             24     2051-03-23 2049-03-24
15         2 2049-03-24             36     2052-03-23 2049-03-24
16         2 2049-03-24             48     2053-03-23 2049-03-24
26         3 2052-02-16             12     2053-02-15 2052-02-16
...      ...        ...            ...            ...        ...
12988   1000 2042-07-23             24     2044-07-22 2042-07-23
12989   1000 2042-07-23             36     2045-07-22 2042-07-23
12990   1000 2042-07-23             48     2046-07-22 2042-07-23
12991   1000 2042-07-23             60     2047-07-22 2042-07-23
12992   1000 2042-07-23             72     2048-07-22 2042-07-23

[3485 rows x 5 columns]
```

Terminating Records

Steps for this stage:

1. Filter only for actual terminations.
2. Restrict terminations only to those occurring in the study period.
3. Set the experience period end date to be the termination date.
4. Set the mode-aversary corresponding to this date.
5. Select only the needed columns, and store.

R

```
.data %>%
  filter(!is.na({{.term_date}})) %>%
  filter({{.term_date}} >= study_start & {{.term_date}} <= study_end) %>%
  select({{.ID}},{{.issue_date}},first_date,{{.term_date}}) %>%
  rename(exp_period_end={{.term_date}} ) %>%
  mutate(monthaversary=interval({{.issue_date}},exp_period_end) / months(1)) %>%
  select({{.ID}},first_date,monthaversary,exp_period_end,{{.issue_date}}) ->
  stage1b
```

Python

```
stage1b = (
  policy_pop_sample.query('Term_Date == Term_Date')
    .query('Term_Date >= @studyStartDate and Term_Date <= @studyEndDate')
)

stage1b = stage1b[['PolID','Issue_Date','first_date','Term_Date']]
stage1b = stage1b.rename(columns={'Term_Date':'exp_period_end'})
stage1b['monthaversary'] = [months_between(x,y,exact=True) for x,y in \
  zip(stage1b['Issue_Date'],stage1b['exp_period_end'])]

stage1b
```

```
     PolID Issue_Date first_date exp_period_end  monthaversary
3        4 2052-09-22 2052-09-22     2053-09-22      12.000000
7        8 2043-11-29 2043-11-29     2048-10-29      58.968817
9       10 2044-09-22 2044-09-22     2045-08-22      10.976344
11      12 2052-11-25 2052-11-25     2053-10-25      10.973118
19      20 2047-06-29 2047-06-29     2048-06-29      12.000000
..     ...        ...        ...            ...            ...
986    987 2047-12-25 2047-12-25     2048-03-25       3.000000
```

```
990     991 2050-10-08 2050-10-08     2050-11-08       1.008602
997     998 2049-09-12 2049-09-12     2050-10-12      12.987097
998     999 2047-06-17 2047-06-17     2048-08-17      13.981720
999    1000 2042-07-23 2042-07-23     2049-01-23      78.000000

[320 rows x 5 columns]
```

Calendar-Year Breaks

Steps for this stage:

1. Cross-join the calendar year breaks.
2. Filter only those occurring in the policy's exposure window.
3. Set the corresponding *modeaversary*.
4. Select only the needed columns, and store.

R

```r
.data %>%
  cross_join(data.table(exp_period_end=.cal_yr_breaks)) %>%
  filter(exp_period_end >= first_date &
           exp_period_end <= last_date) %>%
  mutate(monthaversary=interval({{.issue_date}},exp_period_end) / months(1)) %>%
  select({{.ID}},first_date,monthaversary,exp_period_end,{{.issue_date}}) ->
  stage1c
```

Python

```python
stage1c = (
  policy_pop_sample.merge(
      pd.DataFrame(data={'exp_period_end' : cal_yr_breaks}),
      how='cross'
      ).query(
        'exp_period_end >= first_date and exp_period_end <= last_date'
        )
)

stage1c['monthaversary'] = [months_between(x,y,exact=True) for x,y in \
  zip(stage1c['Issue_Date'],stage1c['exp_period_end'])]
```

```
stage1c = stage1c[['PolID','first_date','monthaversary','exp_period_end', \
  'Issue_Date']]

stage1c
```

```
       PolID first_date  monthaversary exp_period_end Issue_Date
20         2 2049-03-24       9.225806     2049-12-31 2049-03-24
21         2 2049-03-24      21.225806     2050-12-31 2049-03-24
22         2 2049-03-24      33.225806     2051-12-31 2049-03-24
23         2 2049-03-24      45.225806     2052-12-31 2049-03-24
35         3 2052-02-16      10.448276     2052-12-31 2052-02-16
...      ...        ...            ...            ...        ...
11991   1000 2042-07-23      29.258065     2044-12-31 2042-07-23
11992   1000 2042-07-23      41.258065     2045-12-31 2042-07-23
11993   1000 2042-07-23      53.258065     2046-12-31 2042-07-23
11994   1000 2042-07-23      65.258065     2047-12-31 2042-07-23
11995   1000 2042-07-23      77.258065     2048-12-31 2042-07-23

[3618 rows x 5 columns]
```

Putting It All Together and Final Touches

Steps for this stage:

1. Concatenate the three pieces.
2. Ensure we have distinct records.
3. Order by policy ID and exposure period end.
4. Group by Policy ID.
5. Within policy ID, compute the exposure period start as the day after the prior record.
6. Within Policy ID, set null exposure starts to be the first date.
7. Compute policy duration and exact exposure.
8. Select the needed columns, and store.

R

```
rbind(
  stage1a,
  stage1b,
  stage1c
) %>%
  distinct() %>%
  arrange({{.ID}}, exp_period_end) %>%
  group_by({{.ID}}) %>%
  mutate(exp_period_start=lag(exp_period_end) %m+% days(1),
         .before=exp_period_end) %>%
  mutate(exp_period_start=as.Date(ifelse(is.na(exp_period_start),
                                         first_date,
                                         exp_period_start))) %>%
  mutate(pol_duration = pmax(1,ceiling(interval({{.issue_date}},exp_period_end) /
                                         years(1))),
         exposure = (interval(exp_period_start,exp_period_end) / days(1) + 1) /
           ifelse(year(exp_period_end) %% 4 == 0, 366, 365)
  ) %>%
  select({{.ID}},monthaversary,exp_period_start,exp_period_end,pol_duration,exposure) %>%
  data.table() ->
  .data
```

Python

```
exposures = (
  pd.concat(
  [stage1a,stage1b,stage1c]
).drop_duplicates()
 .sort_values(['PolID','exp_period_end'])
 ).reset_index(drop=True)



exposures['exp_period_start'] = [d + relativedelta(days=1) \
  for d in exposures['exp_period_end']]
exposures['exp_period_start'] = exposures.groupby('PolID')['exp_period_start'].shift(1)
exposures['exp_period_start'] = exposures['exp_period_start'].fillna(exposures['first_date

exposures['pol_duration'] = [np.ceil(months_between(x,y,exact=True)/12) for x,y in \
  zip(exposures['Issue_Date'],exposures['exp_period_end'])]
exposures['pol_duration'] = exposures['pol_duration'].astype('int32')

exposures['exposure'] = \
  exposures.apply(
    lambda x: ((x['exp_period_end'] - x['exp_period_start']).days + 1)/ \
        (366 if (x['exp_period_end'].year % 4 == 0) else 365),
      axis=1)

exposures = exposures[['PolID','monthaversary','exp_period_start', \
  'exp_period_end','pol_duration','exposure']]

exposures
```

```
      PolID  monthaversary  ... pol_duration  exposure
0         2       9.225806  ...            1  0.775342
1         2      12.000000  ...            1  0.224658
2         2      21.225806  ...            2  0.775342
3         2      24.000000  ...            2  0.224658
4         2      33.225806  ...            3  0.775342
...     ...            ...  ...          ...       ...
7417   1000      60.000000  ...            5  0.556164
7418   1000      65.258065  ...            6  0.443836
7419   1000      72.000000  ...            6  0.557377
7420   1000      77.258065  ...            7  0.442623
7421   1000      78.000000  ...            7  0.063014

[7422 rows x 6 columns]
```

Final Build With R Function

The R function that I built can be run in parallel, and I have opted to run by issue year for this example. In production, it might be better to run it by experience year.

Without parallelism, it will take several hours to develop the entire exposure set. Using 13 cores as I do here for each issue year, it takes about 12 minutes. This is why I opted not to demonstrate this in Python in this document. Doing this in Python's parallelism will work as well (i.e., create a function and manually spawn processes, and so on).

```
source('expand.exposures.R')

cl <- makeCluster(13)
registerDoParallel(cl=cl)

policy_exposures <- foreach(i=2041:2053,.packages=c("data.table","tidyverse")) %dopar% {
          policy_pop[Issue_Year==i] %>%
```

```r
              filter(Issue_Date <= study_end) %>%
              select(PolID,Issue_Date,Term_Date,Prem_Mode_Months) %>%
              expand_exposures(
                .exp_period_start = study_start,
                .exp_period_end = study_end,
                .cal_yr_breaks = cal_yr_breaks,
                .pol_period_granularity = pol_period_granularity,
                .cal_period_granularity = cal_period_granularity,
                .issue_date = Issue_Date,
                .term_date = Term_Date,
                .ID=PolID,
                .prem_mode_months = Prem_Mode_Months
              )
          }

stopCluster(cl)

policy_exposures <- rbindlist(policy_exposures)

policy_exposures[,ID:=1:nrow(.SD)]

arrow::write_parquet(x=policy_exposures,
                     sink="policy_exposures.parquet")

policy_exposures
```

```
            PolID monthaversary exp_period_start exp_period_end pol_duration
            <int>         <num>           <Date>         <Date>        <num>
       1:      30      4.354839       2041-08-20     2041-12-31            1
       2:      30     12.000000       2042-01-01     2042-08-19            1
       3:      30     16.354839       2042-08-20     2042-12-31            2
       4:      30     24.000000       2043-01-01     2043-08-19            2
       5:      30     28.354839       2043-08-20     2043-12-31            3
     ---
15990849: 1999958     10.225806       2053-02-24     2053-12-31            1
15990850: 1999977     10.161290       2053-02-26     2053-12-31            1
15990851: 1999984      6.967742       2053-06-01     2053-12-31            1
15990852: 1999993      8.870968       2053-04-04     2053-12-31            1
15990853: 2000000      0.000000       2053-03-08     2053-03-08            1
             exposure       ID
                <num>    <int>
       1: 0.367123288        1
       2: 0.632876712        2
       3: 0.367123288        3
       4: 0.632876712        4
       5: 0.367123288        5
      ---
15990849: 0.852054795 15990849
15990850: 0.846575342 15990850
15990851: 0.586301370 15990851
15990852: 0.745205479 15990852
15990853: 0.002739726 15990853
```

Attaching Expected Claims

Attaching the expecting claims is once again a basic database operation.

1. Join the policy census with the exposures.
2. Join the expected basis for the event of interest, if the basis is in table form.
3. Compute derived quantities, and store.

After that, the information can be fed into reporting and analytics pipelines as needed.

```
policy_pop %>%
  filter(UW_Decision != "DEC") %>%
  inner_join(y=policy_exposures,
             by="PolID") %>%
  inner_join(
    y=vbt2015,
    by=join_by(
      Sex==Sex,
      Issue_Age==Issue_Age,
      pol_duration==Duration
    )
  ) %>%
  mutate(ExpectedClaims_2015VBT_Count=exposure*-log(1-qx_su),
         ExpectedClaims_2015VBT_Amount=ExpectedClaims_2015VBT_Count*Face_Amount) %>%
  select(ID,ExpectedClaims_2015VBT_Count,ExpectedClaims_2015VBT_Amount)->
  expected_claims

expected_claims
```

```
                ID ExpectedClaims_2015VBT_Count ExpectedClaims_2015VBT_Amount
```

```
            <int>                        <num>                         <num>
       1: 12646805                1.163101e-04                    11.6310093
       2: 12646806                3.370116e-05                     3.3701158
       3: 12646807                1.318194e-04                    13.1819424
       4: 12646808                3.819503e-05                     3.8195027
       5: 12646809                2.016153e-04                    20.1615252
      ---
15918397: 13740151                2.074082e-04                   165.9265296
15918398: 13740152                1.111676e-04                    88.9340419
15918399: 13740153                2.485927e-04                   198.8741557
15918400: 13740154                1.300547e-04                   104.0437685
15918401: 15990853                4.109897e-07                     0.2054949
```

Session Information

R

```
sessionInfo()
```

```
R version 4.3.3 (2024-02-29)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 22.04.4 LTS

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.20.so;  LAPACK version 

locale:
[1] C

time zone: Etc/UTC
tzcode source: system (glibc)

attached base packages:
[1] parallel  stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] openxlsx_4.2.5.2  reticulate_1.35.0 doParallel_1.0.17 iterators_1.0.14
 [5] foreach_1.5.2     lubridate_1.9.3   forcats_1.0.0     stringr_1.5.1
 [9] dplyr_1.1.4       purrr_1.0.2       readr_2.1.5       tidyr_1.3.1
[13] tibble_3.2.1      ggplot2_3.5.0     tidyverse_2.0.0   arrow_15.0.1
[17] data.table_1.15.2

loaded via a namespace (and not attached):
 [1] utf8_1.2.4        generics_0.1.3    lattice_0.22-5    stringi_1.8.3
 [5] hms_1.1.3         digest_0.6.35     magrittr_2.0.3    evaluate_0.23
 [9] grid_4.3.3        timechange_0.3.0  fastmap_1.1.1     rprojroot_2.0.4
[13] Matrix_1.6-3      jsonlite_1.8.8    zip_2.3.1         fansi_1.0.6
[17] scales_1.3.0      codetools_0.2-19  cli_3.6.2         rlang_1.1.3
[21] bit64_4.0.5       munsell_0.5.0     withr_3.0.0       yaml_2.3.8
[25] tools_4.3.3       tzdb_0.4.0        colorspace_2.1-0  here_1.0.1
[29] assertthat_0.2.1  png_0.1-8         vctrs_0.6.5       R6_2.5.1
[33] lifecycle_1.0.4   bit_4.0.5         pkgconfig_2.0.3   pillar_1.9.0
[37] gtable_0.3.4      Rcpp_1.0.12       glue_1.7.0        xfun_0.42
[41] tidyselect_1.2.1  rstudioapi_0.15.0 knitr_1.45        htmltools_0.5.7
[45] rmarkdown_2.26    compiler_4.3.3
```

Python

```
session_info.show()
```

```
-----
dateutil            2.8.2
numpy               1.23.5
pandas              2.0.3
pyarrow             14.0.1.dev0+gba5374836.d20240125
session_info        1.0.0
-----
Python 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0]
Linux-5.15.146.1-microsoft-standard-WSL2-x86_64-with-glibc2.35
-----
Session information updated at 2024-04-25 13:52
```