Life insurance companies collect significant amounts of data for the purposes of pricing its products. With the size of insurance datasets growing at a rapid rate, organisations are always searching for efficiency in managing and analysing the available data. One way organisations can access efficiency is through the use of R, which is a powerful programming language that can assist in actuarial statistics. This document explores the advantages of using R in life insurance analytics. More specifically, this document discusses how R can be used to perform complex data manipulations; visualize data in a meaningful way; and match member data from the claims file with the premiums file. By examining possible examples of R being used in life insurance, this document aims to demonstrate the value and benefits of incorporating R into life insurance analytics.

1 Data Manipulation

Data manipulation incorporates the processes of cleaning, transforming and restructuring raw data such that further analysis becomes simpler to perform. Often the most time-consuming process in an analytics exercise, data manipulation can include multiple stages, such as

- Cleaning and filtering data to remove errors and adjust for missing values.
- Reformatting data (e.g. converting date of birth from 'Text' type to 'Date' type).
- Linking multiple data sources with different information to create a single dataset.
- Creating new variables and fields (e.g. calculating 'Age Next Birthday' from Date of Birth).

1.1 Advantages

There are multiple benefits of life insurance organisations, using R to assist in this exercise. These include, but are not limited to:

- Scalability: Data cleaning can be scaled to handle larger and more complex datasets.
- Flexibility: Custom functions can be created to address the different challenges.
- Reproducibility/Consistency: Once the processes are set in place, future analysis can be easily completed as another user can perform the same steps to arrive at similar results.

1.2 Applications in Life Insurance Pricing

In a pricing exercise, data is received in the form of membership and claims extracts, oftentimes arriving in a variety of different formats (.xlsx, .csv, SQL DB etc.). Instead of relying on multiple software to access the different file types, R provides us with useful functions that can read and import these files into the R console.

- From the readxl package, *read_excel,* can be used to import .xlsx and .xls files. Thise function also allow the user to specify *sheet* and *range* which can be useful when dealing with complex spreadsheets. *read_csv* provides similar functionality but for .csv files.

```
## Code Demo 1
#The code below uses the readxl package to import the .xlsx files as a data.table object
membership_data <- read_excel("R_sample_data.xlsx", sheet = "Membership Extract") %>% as.data.table()
```

  - From the RODBC package, *sqlFetch* can be used to import in datasets from a SQL server. The *odbcConnect* function is used to connect with the server and is provided as an input to *sqlFetch*.

- Changing Column Headers and Datatypes:

```
## Code Demo 2
#The code below uses the janitor package to adjust the column headings to a more simple to use format. E.g. 'Member ID' to 'member_id'
```

```
3. membership_data <- clean_names(membership_data)
```

Sometimes, the provided data may be provided in an incorrect format and changes are required to be able to use these fields (for example, date_of_birth incorrectly being labelled a character instead of Date).

```
1. ## Code Demo 3
2. #Manual changes to datatypes =>
3. str(membership_data) #To examine the existing datatypes, note date fields imported as POSIXct
4. #Use data.table package (':=' function) to alter datatype
5. membership_data[, date_of_birth := as.Date(date_of_birth)]
```

The following code demonstrates an example of creating additional fields, for example, calculating the age next birthday from the date of birth.

```
1. ## Code Demo 4
2. #The code below combines the data.table and lubridate packages to calculate a new field using existing fields.
3. membership_data[, extract_date := as.Date("2023-01-01")]
4. membership_data[, age_next_brithday := floor(lubridate::interval(date_of_birth, extract_date)/years(1) + 1)]
```

When data is received from the incumbent insurer, the format of the data may need to be changed to match with company standards. R provides simple methods to consolidate the different data entries to conform with what is required. These methods are also useful when multiple extracts of different formats are provided and consolidation is required.

```
1. ## Code Demo 5
2. #Mapping to match company standards =>
3. #unique function used to see unique values in the data field
4. unique(membership_data[, occupation_code])
5. membership_data[, occupation_code := dplyr::case_when(
6.   occupation_code == "Professional" ~ "P",
7.   occupation_code == "Standard" ~ "S",
8.   occupation_code == "White Collar" ~ "WC"
9. )] #data.table ':=' function and dplyr package used to provide correct mapping
```

Missing data is a common issue that is seen across many different pricing exercises, frequently causing headaches for actuaries. R provides useful methods to deal with this problem. In the provided example the missing values are replaced with the mode of each of field, however, R can provide more complex and accurate methods to estimate the missing values (for example, linear imputation using the simputation package and median imputation using the naniar package)

```
1. ## Code Demo 6
2. #Dealing with Missing Values =>
3. is.na(membership_data) %>% colSums() #Here we can see which columns have NA values in them
4. prop.table(table(membership_data$occupation_code))
5. # Can see majority fall into 'S' - treat the missing as 'S'
6. membership_data[is.na(occupation_code), occupation_code := "S"]
```

As such, R provides insurers with many different packages and functions that can directly assist in the data manipulation of complex datasets. Furthermore, these can help life insurers improve in the scalability, flexibility and reproducibility of their data analysis processes.

2 Visualisation

Business Decision Makers often prefer graphical charts to understand key trends and insights from data analysis. Insightful visualisations can be used to explore patterns and trends that would otherwise go unnoticed, allowing management to make the right decisions.

2.1 Advantages

Generally, actuaries rely on Microsoft Excel to create charts and communicate key trends and insights arising from any analysis. However, there are multiple benefits of using R to create charts. These include:

- Flexibility: R provides more flexibility in creating a wide variety of different chart types. Users can also customise most of the elements to the desired output (e.g. it is more simple to adjust the colour palette simply by referencing a pre-defined colour vector, rather than manual changes).
- Scalability: When using R, larger and more complex datasets are more easily handled as the memory requirements are much less when compared to traditional software. Thus, producing visualisations is quicker and more efficient.
- Time Savings: R can create and save charts, such that restructuring and reformatting is not required when new data is added/received. Also, underlying data does not have to be restructured to produce different chart types (proportions vs static values).
- Interactivity: R can produce interactive charts that can further enhance the understanding of complex data by providing additional layers of information.

2.2 Applications in Life Insurance Pricing

Visualisation of Membership Demographics – Can easily view number/premiums from different membership groups. Comparisons can be made to the claims side to see if any unwanted cross-subsidies exist. R provides simple methods to manipulate the original dataset such that visualisations of different segments of the membership become clearer.

- One of the most useful data visualisation packages in R is the ggplot2 package. Ggplot2 allows users to create a variety of charts and plot types, by using a range of built-in functions and tools that provide a more programmatic approach to visualising data. Ggplot2 is designed to work iteratively, meaning individual elements are added onto each other to lead to the final output.
    - Initially, a ggplot object must be created using the *ggplot()* function. In here, the data that will be used in the plot can be specified.

```
## Code Demo 7
membership_state <- ggplot(data = membership_data[, .(number_of_members = .N), by = .(state)]
                           [, prop := number_of_members/sum(number_of_members)])
```

    - Next, a geometric object (the visual element used to represent the data) must be added onto the ggplot object. This is done using the *geom_ *function (* can be replaced by multiple different chart types, in this example it is *geom_col).* In this step the aesthetics (such as the x and y axis, colours, sizes etc.) must also be specified as well.

```
## Code Demo 8
membership_state <- membership_state + geom_col(aes(x = factor(state, levels = order), y = prop), fill =
azuria_cols[1])
```

    - Lastly, additional layers can be added to the chart to change its visual appearance.

```
## Code Demo 9
membership_state <- membership_state + theme_bw() +
```

```
    scale_y_continuous(labels = scales::percent, breaks = seq(0,1,0.1)) +
    labs(title = "Membership Exposure by State", y = "Proportion of membership", x = "State")
```

Claims Analysis

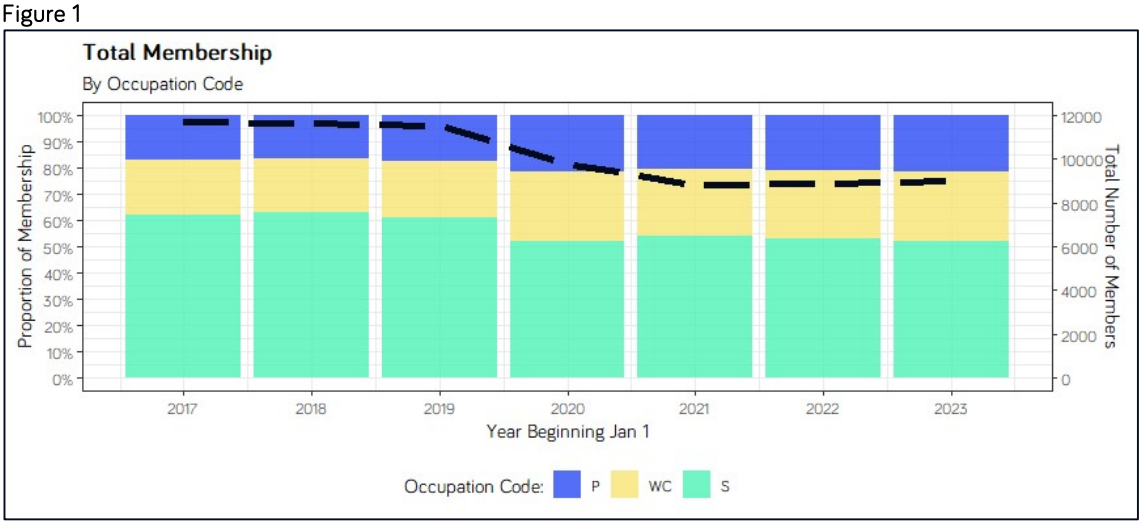
- Data visualisation can be used to analyse claims data and identify patterns in claims behaviour. This can include visualisations of claims frequency, severity, and duration, as well as visualisations of the types of claims causes that are most commonly made by policyholders. These charts can be constructed using a similar approach as above.

Analysing Exposure Changes Over Time

- Following a similar approach to that described in Code Demo's 7-9, we can plot our example membership exposure over time, split by occupation categories. From the Code Demo below, we can see that ggplot2 is able to handle the additional complexity with just a few additional lines of code (Figure 1).

```
## Code Demo 10
occ_code_exposure <- ggplot() +
  geom_bar(data = exposure_data_occ, aes(x = extract, y = members, fill = variable), position = "fill", stat = "identity")
  +
  scale_fill_manual(values = azuria_cols[3:1]) +
  geom_line(data = exposure_data, aes(x = extract, y = total/12000), lwd = 2, col = azuria_cols[4], alpha = 0.7,
  linetype = 5) +
  scale_y_continuous("Proportion of Membership",labels = scales::percent, breaks = seq(0,1,0.1),
                     sec.axis = sec_axis(~ . * 12000, breaks = seq(0,12000,1000), name = "Total Number of Members")) +
  scale_x_continuous("Year Beginning Jan 1", breaks = seq(2017,2023,1))+
  theme_bw() +
  theme(legend.position = "bottom", text = element_text(family = "Bariol Regular"), plot.title = element_text(face =
  "bold")) +
  labs(title = "Total Membership", subtitle = "By Occupation Code", fill = "Occupation Code")
```

Figure 1



The ability of R to handle large data sets, using insightful visualisation packages, make it a valuable tool for life insurers looking to extract insights and improve decision-making. R's versatility enables life insurers to create custom visualisations that can enhance their capabilities and improve their time savings and scalability.

3 Claims and Membership Matching

When performing a pricing exercise of a scheme, claims data is usually matched to the premiums data for multiple reasons including:

- Confirming that the claims information (e.g., sum insured, benefit period) is the same as what is in the premiums file. This ensures that information is consistent across claims and premiums for the purposes of experience investigations.
- With the introduction of PYS and PMIF in 2019/2020, ensuring that members without insurance cover are excluded from the analysis.

3.1 Advantages

- Time Savings: R provides the ability to create functions and algorithms that can be used across multiple different standardised datasets, producing a time saving as new/custom processes do not have to be created for each exercise. R can perform the matching exercise faster than traditional Microsoft Excel methods as there are fewer memory requirements.
- Automation: Using set processes in R can minimise errors that arise from custom user defined functions. Once the process has been built, and testing has been completed, users can be ensured that the correct results are outputted each time.
- Scalability: Similar to the previous sections, using R allows for the same matching processes to be applied to standardised datasets with varying sizes and complexity, thus reducing workload associated with each pricing exercise.

3.2 Applications in Life Insurance Pricing

After performing the necessary steps described in the Data Manipulation Section to standardise the membership and claims datasets, we can join the 2 datasets together. Here, the *left_join* function is used to ensure all elements from the claims data is included, even if there are no matching records.

```
## Code Demo 11
#Some important fields missing in the claims data, have to join with the membership
matched_claims_data<-left_join(x = claims_data, y = membership_data, by = "member_id", all.x = T)
```

Once the two datasets are joined, checks can be built to ensure that specific conditions are met.

```
## Code Demo 12
#The below code are examples of specific checks that can be made
# Check to see whether info is consistent across the different datasets.
matched_claims_data[gender.x != gender.y]
#Check to see if the claim would be active after PYS/PMIF policy changes
#This information is only available after the matching was performed
matched_claims_data[account_balance_indicator == "<6000" | age_next_brithday < 25]
matched_claims_data[contribution_date_indicator == "> 16 months"]
```

As such, claims and membership matching using offers significant benefits to life insurers in term of time savings and scalability, when compared to traditional methods.

Kundan Pooni
E: kundan@azuria.partners
M: +61 405 138 407