CS 567

COMPUTATIONAL STATISTICS

PROJECT 1

Life Insurance using R

Authors
Chao HUANG LIN
41716747
Hermann YEPDJIO
140917845
Lubna ALZAMIL
4144208

Instructor:
Dr.Donald DAVENDRA

January 28, 2019

Contents

List of Figures

1 Introduction

This our first project. We've been asked to implement the formulas used by life insurance companies using R programming language. Implementing the formulas into the computer will make it easier and more efficient to calculate and visualize several types of products offered by the insurance company.

2 Methodology

These steps have been followed to reach the goal of this project:

1. A mortality table is downloaded and saved it as compare.csv.
2. We started to code the necessary formulas in Life-table.R
3. After that, we called the formulas file into our Project1.R to calculate The Net Single Premium for one individual and for 10000 individuals with random ages and benefits.
4. Then, Probability of Death is Calculated.
5. All the output data is stored in BusinessData.csv.
6. Finally, data in BusinessData.csv is visulized using ggplot library.

3 The Code

3.1 Life-table.R

As mentioned earlier, We coded the necessary formulas in Life-table.R.

```
1 startTime <- Sys.time()
2 inputs <- read.table("life_table_inputs.txt", header = TRUE, sep = "\t", dec = ".", stringsAsFactors=FALSE) #read the inputs values from the life_table_inputs.txt file
3 inputMortalityFile <- (inputs[inputs$label == "inputMortalityFile", c("value")])
4 print(inputMortalityFile)
5 data <- read.csv(inputMortalityFile, header = TRUE, stringsAsFactors=FALSE) #avoid convert string to factor
6 dataTotalRow <- nrow(data)
7 beginRowData = 1
```

Line 1 stores the start time when the code is started to execute. At the end of code execution, the total time will be printed. Line 2 read the inputs values from the life-table-inputs.txt file and store them in *inputs*. Line 3 gets the value of inputMortalityFile row from *inputs* data frame which is compare.csv and stores it in *inputMortalityFile*.Line 4 prints the value of *inputMortalityFile*. Line 5 reads the csv data of inputMortalityFile and store them in *data*. stringsAsFactors=FALSE means that do not covert the string data to factors. Line 6 counts the number of rows in *data*.

```
8  ages = as.numeric(data[beginRowData:dataTotalRow, 1])
9  mortality = as.numeric(data[beginRowData:dataTotalRow, 2])
10
11 lifeTable <- data.frame(ages,mortality)
12
13 lifeTableTotalRow = nrow(lifeTable)
14 lifeTable$t <- lifeTable$ages + 1
15 lifeTable$q <- lifeTable$mortality
16 lifeTable$p <- 1 - lifeTable$q
```

Line 8 starts to loop inside the first column (ages column) of*data* data frame from 1 until the total rows of *data* then convert the ages into numeric values and store them in *ages*. Line 9 does the same with the second column ,which contains the mortality, storing them as numeric values in *mortality*. Line 11 creates a data frame called *lifeTable* which contains for now two columns *ages* and *mortality*. Line 13 counts the number of rows in *lifeTable* and store them in *lifeTableTotalRow*. Line 14 adds new column to the data frame *lifeTable* the column is called t and contains the values ages+1. Line 15 adds new column to the data frame *lifeTable* the column is called q and contains the same data as *mortality* column. Line 16 as well adds new column to the data frame *lifeTable* the column is called p and contains the values of 1-q.

```
17 #calculate v^t
18 interest <- as.numeric(inputs[inputs$label == "interest", c("value")])
19 lifeTable$vt <- 1/(1+interest)^lifeTable$t
20
21 #calculate t_p_x | x=0
22 lifeTable$t_p_x0[1] <- lifeTable$p[1]
23 for (i in 2:lifeTableTotalRow) {
24   lifeTable$t_p_x0[i] <- lifeTable$t_p_x0[i-1] * lifeTable$p[i]
25 }
```

Lines 18 gets the value of interest from *inputs* and stores it in *interest*. Line 19 add new column to the data frame *lifeTable* the column is called v^t which contains the value of $v^t = (1+i)^-t$. Lines 21-25 add new column to the data frame *lifeTable* the column is called ${}_tp_{x0}$ that contains the value of
${}_tp_x = lx_n/lx_{n-1}$=$p_x{}^*{}_{t-1}p_{x=0}$.

```
26 #calculate t_E_x | x=0
27 lifeTable$t_E_x0 <- lifeTable$vt * lifeTable$t_p_x0
28 
29 #calculate a_x
30 lifeTable$a_x[1] <- 1 + sum(lifeTable$t_E_x0)
31 for (i in 2:lifeTableTotalRow) {
32   lifeTable$a_x[i] <- 1 +
       sum(lifeTable$t_E_x0[i:lifeTableTotalRow])/lifeTable$t_E_x0[i-1]
33 }
34 
35 #calculate A_x
36 d = interest/(1 + interest)
37 lifeTable$A_x <- 1 - d * lifeTable$a_x
```

Line 27 adds new column to the data frame *lifeTable* the column is called ${}_tE_{x0}$ and contains the value ${}_tE_x = v^t * {}_tp_{x0}$. Lines 29-33 add new column to the data frame *lifeTable* the column is called ax and its values are $ax = 1 + \Sigma_t E_x$.The first value in ax column is= $1 + \Sigma_t E_{x0}$. Lines 35-37 add new column to the data frame *lifeTable* the column is called A_x and contains the value of $A_x = 1 - d * ax$.

3.2 Projec1.R

In this file, we are calling the formulas located in life-table.R using the source function.

```
1 setwd(C://Desktop//Statis//GroupProject1//3R)
2 inputsProject1 <- read.delim("project1_inputs.txt", header = TRUE, sep =
    "\t", dec = ".", stringsAsFactors=FALSE) #read the inputs values from
    the project1_inputs.txt file
3 print (inputsProject1)
4 source("life_table.R")
5 #plot
6 #install.packages("ggplot2")
7 library(ggplot2)
8 library(reshape)
9 x11()
10 myGraph <- ggplot(lifeTable, aes(ages, A_x))
11 myGraph <- myGraph + geom_point()
12 print(myGraph)
```

Line 1 sets the working directory to the path of where the project and needed files are stored . Line 2 reads the inputs values fromthe project1-inputs.txt file and stores them in *inputsProject1*. Line 3 prints the values in *inputsProject1*. Line 4 calls Life-table.R file that contains the necessary formulas. Line 6 must be written in console to install the library need to plot graphs then after that you need to call the libraries you want to use in your project. Here we are calling ggplot2 and reshape libries in lines7-8. Reshape library is used to cast a data frame into the reshaped form. The library ggplot2 is used for creating plots. Line 9 opens a new graph window before creating a new graph to avoid overwriting of previous plotted graphs. X11() function is used in Unix platform, windows() is used in Windows platform and quartz() is used in Mac platform. Line 10 myGraph contains plots the *ages* and A_x cloumns from *lifeTable* data frame. Line 11 geom point function is used to create scatter points displaying the relationship between *ages* and A_x. Line 12 shows the plot. Figure 1 shows the result of Line 12.

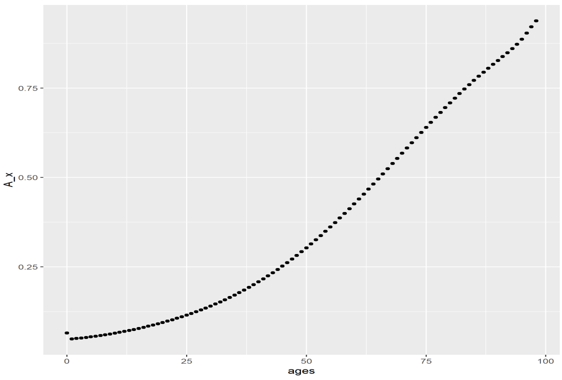

Figure 1: The Relationship between A_x and Ages

```
13 inputAges = as.numeric(inputsProject1[inputsProject1$label ==
       "inputAges", c("value")])
14 inputBenefit = as.numeric(inputsProject1[inputsProject1$label ==
       "inputBenefit", c("value")])
15 nsp <- lifeTable[lifeTable$ages == inputAges, c("A_x")]*inputBenefit
16 print(paste("input Ages: ", inputAges))
17 print(paste("Whole Life Net Single Premium: ", nsp))
```

In lines 13-17 we are passing the same values found in (Life Insurance Pricing Drew Heber.xlsx) file given by our instructor. We did that to make sure that our formulas are running correctly. We stored these inputs in *inputsProject1* and we are calling them , convert them to numeric and store them in *inputAges* and *inputBenefit*. Lines 12-13 are taken from the given file. Line 14 extracts the corresponding A_x of the *inputAges* which is 12 and multiples it by the *inputBenefit* which is 1000 storing the result in *nsp*. The formula used is $NetSinglePremium = Benefit * A_x$.
Line 16 prints the age. Line 17 prints the result of Net Single Premium.

```r
18 #calculate probability of (x) lives t years t_p_x where x = inputAges
19 previousAgeP <- lifeTable[ages == (inputAges - 1), c("t_p_x0")]
20 if (inputAges > 0) {
21   pLives <- lifeTable[ages >= inputAges, c("t_p_x0")]/previousAgeP
22 } else {
23   pLives <- lifeTable[ages >= inputAges, c("t_p_x0")]
24 }
25
26 pLivesAges <- lifeTable[ages >= inputAges, c("ages")]
27 pCurveX <- data.frame(pLivesAges, pLives)
```

Line 19 finds the probablity ${}_tp_{x0}$ of *inputAges - 1* and stores the result in *previousAgeP*. Lines 20-24 checks whether the *inputAges* is greater than zero or not. If it is, it will get all the probabilities ${}_tp_{x0}$ of all ages and divide by *previousAgeP* and store the result in *pLives*. If the *inputAges* is not greater than zero, it will get all the probabilities ${}_tp_{x0}$ of all ages and store them in *pLives*. Line 26 stores all the ages that are greater than or equal to *inputAges* in *pLivesAges*. Line 27 creates a data frame called *pCurveX* that has two columns *pLivesAges* and *pLives*.

```r
28 #calculate probability t_1_q_x that x survives t years and dies within 1 year
29 pCurveXRow <- nrow(pCurveX)
30
31 #probability of dead based on input age
32 for (i in 1:(pCurveXRow-1)) {
33   pCurveX$t_1_q_x[i] <- pCurveX$pLives[i] - pCurveX$pLives[i+1] #i = u in the pdf 1 = t in pdf
34 }
35 pCurveX$t_1_q_x[pCurveXRow] <- pCurveX$pLives[pCurveXRow] # last line
```

Now, we will calculate ${}_{t1}q_x$ which is the probability that x survives t years and dies within 1 year. The general formula used to calculate this probability is ${}_{u|t}q_x = {}_u p_x - {}_{u+t}p_x$. In Line 29 *pCurveXRow* stores the number of rows in *pCurveX* data frame. Line 32 iterates from 1 until *pCurveXRow*-1. Line 33 adds new column to the data frame *pCurveX* the column is called ${}_{t1}q_x$ and contains the values of the formula ${}_{u|t}q_x = {}_u p_x - {}_{u+t}p_x$. Finally, Line 35 stores the ${}_{u|t}q_x$ of the last row in *pCurveX* data frame.

```r
36 #plot lives probability based on input age
37 myGraph <- ggplot(pCurveX, aes(pLivesAges, pLives))
38 myGraph + geom_point()
39
40 x11()
41 #plot t_1_q_x based on input age
42 myGraph <- ggplot(pCurveX, aes(pLivesAges, t_1_q_x))
43 myGraph <- myGraph + geom_point() + labs(title = "probability of (x) survive t years and die next year")
```

```
44 print(myGraph)
```

Lines 37-38 plot *pLivesAges* and *pLives* of *pCurveX* data frame and will store the plot in *myGraph*. Line 40 x11() function opens a new graph window before creating a new graph to avoid overwriting of previous plotted graphs as mentioned earlier. Line 42 plots *pLivesAges* and ${}_{t1}q_x$ *pCurveX* data frame and will store the plot in *myGraph*. Line 43 sets the title of graph window to "probability of (x) survive t years and die next year". Line 44 displays the graph on the screen.

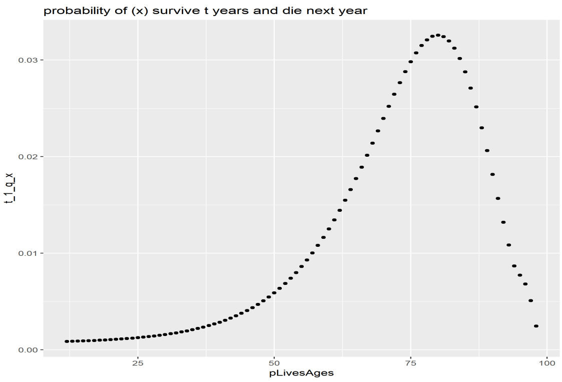

Figure 2: probability of (X) survive t years and die next year

```
45 #now creating a block of 10000 people who are in different ages and want
     different benefits
46 BussinessBlock <- function(rAge,rBenefit) {
47   netSinglePremium <- lifeTable[lifeTable$ages == rAge, c("A_x")]*rBenefit
48   return(netSinglePremium)
49 }
50 
51 bAge <- 0
52 bBen <- 0
53 bNps <- 0
54 bFAge <- 0
55 maxAges <- max(lifeTable$ages)
56 lifeTableAges <- lifeTable$ages[ages < maxAges]# avoid picking max ages
57 inputNumberClients <- as.numeric(inputsProject1[inputsProject1$label ==
     "inputNumberClients", c("value")])
```

Line 46 creates a function named ***BussinessBlock*** that receives two parameters -age and benefit- and calculates and returns the Net Single Premium according to the formula $NetSinglePremium = Benefit * A_x$. Lines 51-54 creates new variables and set them to zeros for later use. Line 55 gets the maximum age from *lifeTable* data frame and stores in *maxAges*. Line 56 stores all the ages that are less than *maxAges* in *lifeTableAges*. Line 57 gets the population value which is 10000 from *inputsProject1* and stores it in *inputNumberClients*.

```
58 print("---------Calculating Lifetimes-----------")
59 print(paste("Number of clients: ", inputNumberClients))
60 for (i in 1:inputNumberClients){ # 10,000 (whole life) incurances with
     Net Single Premium
61   randomAge <- sample(lifeTableAges, 1)
62   randomBenefit <- sample.int(9000, 1, replace=TRUE) + 1000 # picking one
       randonm integer from range $1000-$1000000 benefit
63   bAge[i] <- randomAge # concatenate
64   bBen[i] <- randomBenefit # concatenate
65   bNps[i] <- BussinessBlock(randomAge,randomBenefit) # calling the
       function to calculate the net single premium then concatenate
```

Lines 58-59 are print statements to notify the user of calculating life times and number of clients. Line 60 will loop through the *inputNumberClients*. Line 61 will pick up one random age from *lifeTableAges* and stores it in *randomAge*. Line 62 will pick one random benefit in range \$1000-\$1000000 and stores it in *randomBenefit*. Line 63 concatenates the picked *randomAge* to bAge. Line 64 concatenates the picked *randomBenefit* to bBen. Line 65 passes the*randomAge* and *randomBenefit* to ***BussinessBlock*** function and concatnates the result to bNps.

```
66   #calculate probability of (x) lives t years t_p_x where x = inputAges
67   previousAgeP <- lifeTable[ages == (randomAge - 1), c("t_p_x0")]
68   if (randomAge > 0) {
69     pLives <- lifeTable[ages >= randomAge, c("t_p_x0")]/previousAgeP
70   } else {
71     pLives <- lifeTable[ages >= randomAge, c("t_p_x0")]
72   }
73   pLivesAges <- lifeTable[ages >= randomAge, c("ages")]
74   pCurveX <- data.frame(pLivesAges, pLives)
75     #calculate probability t_1_q_x that x survives t years and dies
         within 1 year
76   pCurveXRow <- nrow(pCurveX)
77
78   #probability of dead based on input age
79   for (j in 1:(pCurveXRow-1)) {
80     pCurveX$t_1_q_x[j] <- pCurveX$pLives[j] - pCurveX$pLives[j+1] #i = u
         in the pdf 1 = t in pdf
81   }
82   pCurveX$t_1_q_x[pCurveXRow] <- pCurveX$pLives[pCurveXRow] # last line
83   #generate random dies based on input age
84   randomFinalAge <- sample(pCurveX$pLivesAges, 1, replace = T,
       pCurveX$t_1_q_x)
85   bFAge[i] <- randomFinalAge
```

Line 67 finds the probablity ${}_tp_{x0}$ of *randomAge - 1* and stores the result in *previousAgeP*. Lines 68-72 checks whether the *randomAge* is greater than zero or not. If it is, it will get all the probabilities ${}_tp_{x0}$ of all ages and divide by *previousAgeP* and

store the result in *pLives*. If the *randomAge* is not greater than zero, it will get all the probabilities $_tp_{x0}$ of all ages and store them in *pLives*. Line 73 stores all the ages that are greater than or equal to *randomAge* in *pLivesAges*. Line 74 creates a data frame called *pCurveX* that has two columns *pLivesAges* and *pLives*. Now, we will calculate $_{t1}q_x$ which is the probability that x survives t years and dies within 1 year. The general formula used to calculate this probability is
$_{u|t}q_x =_u p_x$ - $_{u+t}p_x$. In Line 76 *pCurveXRow* stores the number of rows in *pCurveX* data frame. Line 79 iterates from 1 until *pCurveXRow*-1. Line 80 adds new column to the data frame *pCurveX* the column is called $_{t1}q_x$ and contains the values of the formula $_{u|t}q_x =_u p_x$ - $_{u+t}p_x$. Finally, Line 82 stores the $_{u|t}q_x$ of the last row in *pCurveX* data frame. Line 84

```
86 bDataFrame <- data.frame(Age = bAge, Benefit = bBen, NetSinglePremium = bNps, Die = bFAge) #Creating the final dataframe
87 bDataFrame$SurviveYears <- bDataFrame$Die - bDataFrame$Age
88  # creating the CSV file, each time we run the code new file will be created and the older file will be replaced
89 write.table(bDataFrame, file="BusinessData.csv", row.names=F, col.names=T, append=F, sep=",")
90 paymentTable <- bDataFrame[c("SurviveYears", "Benefit")]
91 paymentTable <- aggregate(. ~SurviveYears, data=paymentTable, sum, na.rm = TRUE)
```

Line 86 creates a data frame named *bDataFrame* that contains Age, Benefit, NetSinglePremium and Die columns. Line 87 adds ner cloumn to *bDataFrame* named *SurviveYears* that contains the difference between Die and Age column. Line 89 creates the CSV file, each time we run the code new file will be created and the older file will be replaced. Lines 90-91 creates a data frame named *paymentTable* that contains SurviveYears and Benefit columns from *bDataFrame* data frame.

```
92 investmentInterest <- as.numeric(inputsProject1[inputsProject1$label == "investmentInterest", c("value")])
93 year <- 0
94 money <- sum(bDataFrame$NetSinglePremium)
95 earnInterest <- money*investmentInterest
96 paymentRec <- 0
97 #find payment of the year
98 payment <- paymentTable[paymentTable$SurviveYears == 0, c("Benefit")]
99 if(length(payment) == 0){ #sometime there is no payment for specific year, so convert numeric(0) to 0
100   paymentRec <- 0
101 }else{
102   paymentRec <- payment
103 }
104 nYears <- max(paymentTable$SurviveYears)
```

Now, we are gonna calculate and see whether the insurance company makes money or loses money with interest rate of 5.00%. Line 92 gets the interest value which is 0.05

from *inputsProject1* and stores it in *investmentInterest*. Line 93 sets the year to zero which indicates the current year. Line 94 calculates the sum of NetSinglePremium column in *bDataFrame* data frame and stores the result in *money*. Lines 98-103 calculates the payment of benefits in current year. Line 98 stores the benefits paid in the current year in *payment*. Lines 99-103 checks if the length of *payment* is equal to zero or not. If it is , let *paymentRec* be zore. Else, copy the data of *payment* to *paymentRec*. Line 104 gets the maximum SurviveYears and stores it in *nYears*.

```
105 for (i in 1:nYears) {
106   year[i+1] <- i
107   money[i+1] <- money[i] + earnInterest[i] - paymentRec[i]
108   if(money[i+1] > 0){ #only calculate earnInterest when money > 0
109     earnInterest[i+1] <- money[i+1]*investmentInterest
110   }else {
111     earnInterest[i+1] <- 0
112   }
113   #find payment of next year
114   payment <- paymentTable[paymentTable$SurviveYears == i, c("Benefit")]
115   if(length(payment) == 0){ #sometime there is no payment for specific
        year, so convert numeric(0) to 0
116     paymentRec[i+1] <- 0
117   }else{
118     paymentRec[i+1] <- payment
119   }
120 }
121 
122 profitTable <- data.frame(year, money, earnInterest, paymentRec)
```

Line 105 loops from 1 to *nYears*. Line 106 sets the next year to i. Line 107 calculates the money that will be earn next year. The money of the next year = money of the last year + interests earned last year - benefit payment record of last year. Lines 108-112 checks whether the money of the next year is greater than zero or not. If it is, Then interests that will be earned next year = money of the next year * investment Interest. If it is not grater than zero, then there is no earnings for the next year. Line 114 gets the benefit that should be paid for i survived years. Lines 115-120 checks if the payment is equal to zero or not. If it is, the payment record for the next year will be zero and that meas nobody died this year. Otherwise, the payment record for the next year will be equal to *payment* that contains the benefit. Finally, Line 122 creates a data frame called *profitTable* that has four columns year, money, earnInterest and paymentRec. As mentioned earlier, x11() function opens a new graph window before creating a new graph to avoid overwriting of previous plotted graphs. The following lines 123-151 plots the Age, Benefit, NetSinglePremium, Die and SurviveYears columns from *bDataFrame*. Each of these column is plotted on its own window. All these plots are plotted as histograms. All the images will be saved in images directory which should be created on the same working directory. Each code will have its graph under it.

```
123 x11()
124 myGraph <- ggplot(bDataFrame, aes(Age))
125 myGraph <- myGraph + geom_histogram() + labs(title = "Random Ages
      Histogram")
126 print(myGraph)
127 ggsave(filename = "images/Random Ages Histogram.png", plot = myGraph)
```

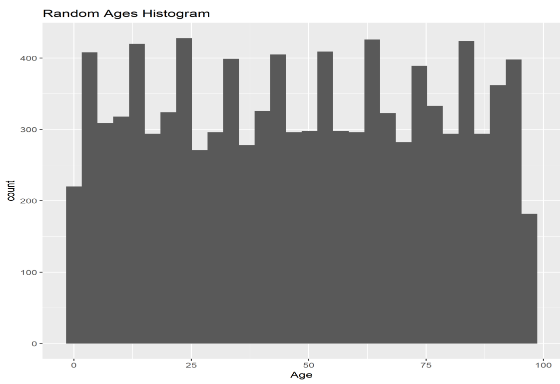

Figure 3: Random Ages Histogram

```
128 x11()
129 myGraph <- ggplot(bDataFrame, aes(Benefit))
130 myGraph <- myGraph + geom_histogram() + labs(title = "Random Benefit
      Histogram")
131 print(myGraph)
132 ggsave(filename = "images/Random Benefit Histogram.png", plot = myGraph)
```

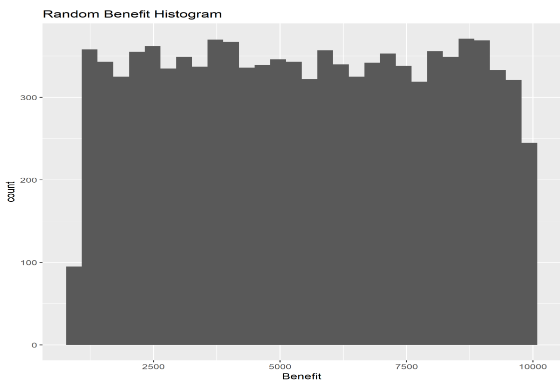

Figure 4: Random Benefit Histogram

```
x11()
myGraph <- ggplot(bDataFrame, aes(NetSinglePremium))
myGraph <- myGraph + geom_histogram() + labs(title = "Random Net Single
    Premium Histogram")
print(myGraph)
ggsave(filename = "images/Random Net Single Premium Histogram.png", plot
    = myGraph)
```

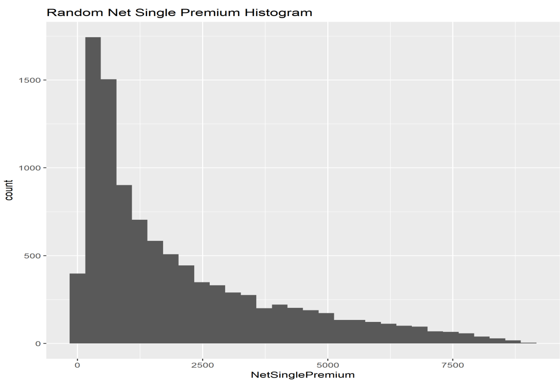

Figure 5: Random Net Single Premium Histogram

```
x11()
myGraph <- ggplot(bDataFrame, aes(Die))
myGraph <- myGraph + geom_histogram() + labs(title = "Random Dead Ages
    Histogram")
print(myGraph)
ggsave(filename = "images/Random Dead Ages Histogram.png", plot = myGraph)
```

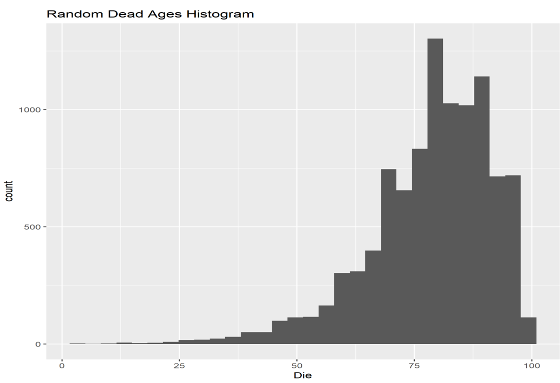

Figure 6: Random Dead Ages Histogram

```
143 x11()
144 myGraph <- ggplot(bDataFrame, aes(SurviveYears))
145 myGraph <- myGraph + geom_histogram() + labs(title = "Random Survive
        Years Histogram")
146 print(myGraph)
147 ggsave(filename = "images/Random Survive Years Histogram.png", plot =
        myGraph)
```

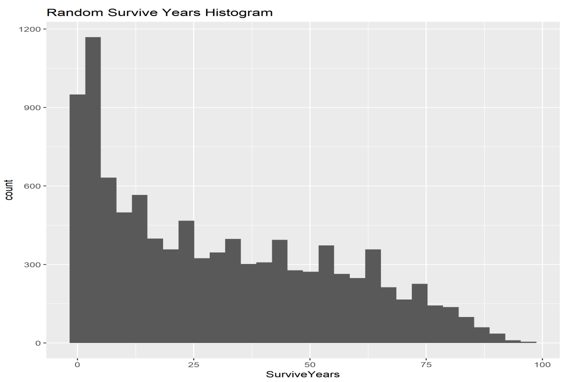

Figure 7: Random Survive Years Histogram

```
148 #------------------- print profit graph ------------------
149 meltProfitTable <- melt(profitTable, id="year")
150 x11()
151 myGraph <- ggplot(meltProfitTable, aes(x = year, y = value, colour =
        variable))
152 myGraph <- myGraph + geom_point() + labs(title="Company Profit Graph", y
        = "Money [$US]") +
153   scale_color_manual(labels = c("Profit", "Interest", "Payment"), values
        = c("Green","blue", "red"))
154 print(myGraph)
155 ggsave(filename = "images/Company Profit Graph.png", plot = myGraph)
156
157 endTime <- Sys.time() # calulating the ending time
158 totalTime <- endTime - startTime #calulating the total time
159 print(totalTime) # printing the total time
```

Line 153 uses one of the reshapes's library functions. Melt function which allows you to plot more that one thing on the same window. As mentioned earlier x11() function opens a new graph window before creating a new graph to avoid overwriting of previous plotted graphs. Line 155 sets the axis of the plot. Line 156 sets the title and the plot will be plotted as points. Line 157 sets the legend of the plot. Line 158 displays the graph on the screen. Line 159 saves the graph on images directory which should be created on the same working directory. Line 161 stores the time when the code is ended its execution. Line 162 calculates the total time . Line 163 prints the total time. Figure 8 shows the output of line 158.

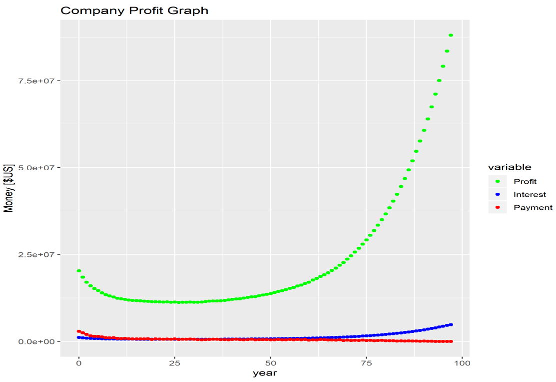

Figure 8: Company Profit Graph