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

February 10, 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.

Listing 1: Life Table File

```
startTime <- Sys.time()
setwd("/media/hermann/Tonpi/tonpi/Collegecourses/CWU/Graduate
    School/Winter 2019/CS 567/Projects/Project1/3R")
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
inputMortalityFile <- (inputs[inputs$label == "inputMortalityFile",
    c("value")])
print (inputMortalityFile)
data <- read.csv(inputMortalityFile, header = TRUE,
    stringsAsFactors=FALSE) #avoid convert string to factor
dataTotalRow <- nrow(data)
beginRowData = 1
```

Listing 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 sets the working directory. Line 3 read the inputs values from the life-table-inputs.txt file and store them in *inputs*. Line 4 gets the value of inputMortalityFile row from *inputs* data frame which is compare.csv and stores it in *inputMortalityFile*.Line 5 prints the value of *inputMortalityFile*. Line 6 reads the csv data of inputMortalityFile and store them in *data*. stringsAsFactors=FALSE means that do not covert the string data to factors. Line 7 counts the number of rows in *data*.

Listing 2: Life Table File

```
1 ages = as.numeric(data[beginRowData:dataTotalRow, 1])
2 mortality = as.numeric(data[beginRowData:dataTotalRow, 2])
3
4 lifeTable <- data.frame(ages,mortality)
5
6 lifeTableTotalRow = nrow(lifeTable)
7 lifeTable$t <- lifeTable$ages + 1
8 lifeTable$q <- lifeTable$mortality
9 lifeTable$p <- 1 - lifeTable$q
```

in Listing 2, line 1 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 2 does the same with the second column ,which contains the mortality, storing them as numeric values in *mortality*. Line 4 creates a data frame called *lifeTable* which contains for now two columns *ages* and *mortality*. Line 6 counts the number of rows in *lifeTable* and store them in *lifeTableTotalRow*. Line 7 adds new column to the data frame *lifeTable* the column is called t and contains the values ages+1. Line 8 adds new column to the data frame *lifeTable* the column is called q and contains the same data as *mortality* column. Line 9 as well adds new column to the data frame *lifeTable* the column is called p and contains the values of 1-q.

Listing 3: Life Table File

```
1 #calculate v^t
2 interest <- as.numeric(inputs[inputs$label == "interest", c("value")])
3 lifeTable$vt <- 1/(1+interest)^lifeTable$t
4
5 #calculate t_p_x | x=0
6 lifeTable$t_p_x0[1] <- lifeTable$p[1]
7 for (i in 2:lifeTableTotalRow) {
8   lifeTable$t_p_x0[i] <- lifeTable$t_p_x0[i-1] * lifeTable$p[i]
9 }
```

In Listing 3, lines 1 gets the value of interest from *inputs* and stores it in *interest*. Line 2 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 6-8 add new column to the data frame *lifeTable* the column is called ${}_tp_{x0}$ that contains the value of ${}_tp_x = lx_n/lx_{n-1}{=}p_x{*}_{t-1}p_{x=0}$.

Listing 4: Life Table File

```
1  #calculate t_E_x | x=0
2  lifeTable$t_E_x0 <- lifeTable$vt * lifeTable$t_p_x0
3
4  #calculate a_x
5  lifeTable$a_x[1] <- 1 + sum(lifeTable$t_E_x0)
6  for (i in 2:lifeTableTotalRow) {
7    lifeTable$a_x[i] <- 1 +
         sum(lifeTable$t_E_x0[i:lifeTableTotalRow])/lifeTable$t_E_x0[i-1]
8  }
9
10 #calculate A_x
11 d = interest/(1 + interest)
12 lifeTable$A_x <- 1 - d * lifeTable$a_x
```

In Listing 4, line 2 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 5-7 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 11-12 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.

Listing 5: Project1 File

```
setwd(C://Desktop//Statis//GroupProject1//3R)
inputsProject1 <- read.delim("project1_inputs.txt", header = TRUE, sep =
    "\t", dec = ".", stringsAsFactors=FALSE) #read the inputs values from
    the project1_inputs.txt file
print (inputsProject1)
source("life_table.R")
#install library if not exist
if (!require(ggplot2)) install.packages('ggplot2')
if (!require(reshape)) install.packages('reshape')
library(ggplot2)
library(reshape)
x11()
myGraph <- ggplot(lifeTable, aes(ages, A_x))
myGraph <- myGraph + geom_point()
print(myGraph)
ggsave(filename = "images/A_x Vs Ages.png", plot = myGraph)
```

In Listing 5, line 1 sets the working directory to the path of where the project and needed files are stored . Line 2 reads the inputs values from the 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 5-6 install the needed packages and call them with library function . Here we are calling ggplot2 and reshape libries in lines 8-9. Reshape library is used to cast a data frame into the reshaped form. The library ggplot2 is used for creating plots. Line 10 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 11 myGraph contains plots the *ages* and A_x cloumns from *lifeTable* data frame. Line 12 geom point function is used to create scatter points displaying the relationship between *ages* and A_x. Line 13 shows the plot on the screen. Line 14 saves the plot on images folder. Figure 1 shows the result of Line 12.

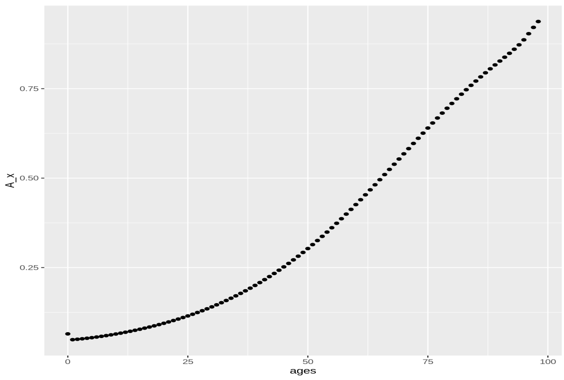

Figure 1: The Relationship between A_x and Ages

Listing 6: Project1 File

```
1 inputAges = as.numeric(inputsProject1[inputsProject1$label ==
    "inputAges", c("value")])
2 inputBenefit = as.numeric(inputsProject1[inputsProject1$label ==
    "inputBenefit", c("value")])
3 nsp <- lifeTable[lifeTable$ages == inputAges, c("A_x")]*inputBenefit
4 print(paste("input Ages: ", inputAges))
5 print(paste("Whole Life Net Single Premium: ", nsp))
```

In Listing 6, lines 1-2 get the values of inputAges and inputBenefit columns in *inputsProject1* convert them to numeric and store them in *inputAges* and *inputBenefit*. Line 3 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 4 prints the age. Line 5 prints the result of Net Single Premium.

Listing 7: Project1 File

```
#-------------------Calculating Probability of Dead Curve--------------
CalculateFinalAge <- function(inputAge) {
  #calculate probability of (x) lives t years t_p_x where x = inputAges
  previousAgeP <- lifeTable[ages == (inputAge - 1), c("t_p_x0")]
  if (inputAge > 0) {
    pLives <- lifeTable[ages >= inputAge, c("t_p_x0")]/previousAgeP
  } else {
    pLives <- lifeTable[ages >= inputAge, c("t_p_x0")]
  }

  pLivesAges <- lifeTable[ages >= inputAge, c("ages")]
  pCurveX <- data.frame(pLivesAges, pLives)

  #calculate probability tl1_q_x that x survives t years and dies within
     1 year
  pCurveXRow <- nrow(pCurveX)

  #probability of dead based on input age
  pCurveX$tl1_q_x[1] <- 1 - pCurveX$pLives[1] # firt data important,,, to
     avoid bug
  if(pCurveXRow > 1) { #bug -fixed, avoid enter for when nrow(pCurveX) ==
     0, this occur when inputAge is the last value of the tablelife
    for (j in 2:(pCurveXRow)) {

      #small bug which does not consider the 0|1_q_x data..... also add
         q_x of 1 in the last year automatically?
      pCurveX$tl1_q_x[j] <- pCurveX$pLives[j-1] - pCurveX$pLives[j]
      #pCurveX$tl1_q_x[j] <- pCurveX$pLives[j] - pCurveX$pLives[j+1] #i =
         u in the pdf 1 = t in pdf
    }
  }
  #generate random dies based on input age
  finalAge <- 0
  if(pCurveXRow > 1) { #bug -fixed, avoid enter when nrow(pCurveX) == 0,
     this occur when inputAge is the last value of the tablelife
    finalAge <- sample(pCurveX$pLivesAges, 1, replace = T,
       pCurveX$tl1_q_x)
  }else
  {
    finalAge <- pCurveX$pLivesAges[1] # only has 1 probability
  }
  return (finalAge)
}
```

Listing 7 Calculates Probability of Death using the defined function CalculateFinalAge.

Listing 8: Project1 File

```
CalculateFinalAgePerfectData <- function(inputAge) { #---only for testing

  #calculate probability of (x) lives t years t_p_x where x = inputAges
  previousAgeP <- lifeTable[ages == (inputAge - 1), c("t_p_x0")]
  if (inputAge > 0) {
    pLives <- lifeTable[ages >= inputAge, c("t_p_x0")]/previousAgeP
  } else {
    pLives <- lifeTable[ages >= inputAge, c("t_p_x0")]
  }

  pLivesAges <- lifeTable[ages >= inputAge, c("ages")]
  pCurveX <- data.frame(pLivesAges, pLives)

  #calculate probability tl1_q_x that x survives t years and dies within
      1 year
  pCurveXRow <- nrow(pCurveX)

  #probability of dead based on input age
  pCurveX$tl1_q_x[1] <- 1 - pCurveX$pLives[1] # firt data important,,, to
      avoid bug
  for (j in 2:(pCurveXRow)) {

    #small bug which does not consider the 0|1_q_x data..... also add q_x
        of 1 in the last year automatically?
    pCurveX$tl1_q_x[j] <- pCurveX$pLives[j-1] - pCurveX$pLives[j]
    #pCurveX$tl1_q_x[j] <- pCurveX$pLives[j] - pCurveX$pLives[j+1] #i = u
        in the pdf 1 = t in pdf
  }
  #pCurveX$tl1_q_x[pCurveXRow] <- pCurveX$pLives[pCurveXRow] # last line

  #bFAge <- sample(pCurveX$pLivesAges, inputNumberClients, replace = T,
      pCurveX$tl1_q_x)

  #------generate perfect pCurve data
  perfectData <- vector(mode="numeric", length=inputNumberClients) # it
      is better to initialize variable to make it faster

  bIndex <- 1 #begin index
  fIndex <- 0 #final index
  counter <- 0
  for(k in 1:pCurveXRow){

    nRepeat <- pCurveX$tl1_q_x[k]*inputNumberClients
    if(nRepeat%%1 >= 0.5){ # use this to avoid number of data > pCurveXRow
      if(counter%%2 == 0){
        nRepeat <- round(nRepeat)
      }else{
        nRepeat <- trunc(nRepeat) # use trunc to avoid number of data >
```

```
          pCurveXRow
      }
      counter <- counter +1
    }else{
      nRepeat <- trunc(nRepeat)
    }

    fIndex <- bIndex + nRepeat - 1
    perfectData[bIndex:fIndex] <- pCurveX$pLivesAges[k]
    bIndex <- bIndex + nRepeat
  }
  #nData <- length(perfectData)
  nFill <- inputNumberClients - fIndex

  #fill residual round values from nData to numberclient
  perfectData[(fIndex+1):inputNumberClients] <-
      sample(pCurveX$pLivesAges, nFill, replace = T, pCurveX$tl1_q_x)
  #perfectData[(nData+1):inputNumberClients] <- perfectData[nData]

  #return a vector
  return(perfectData)
}
```

Listing 8 is just for testing purposes, it generates final age based on a perfect probability curve of death.

Listing 9: Project1 File

```
CalculateFinalAgeSemiPerfectData <- function(inputAge) { #---only for testing

  #calculate probability of (x) lives t years t_p_x where x = inputAges
  previousAgeP <- lifeTable[ages == (inputAge - 1), c("t_p_x0")]
  if (inputAge > 0) {
    pLives <- lifeTable[ages >= inputAge, c("t_p_x0")]/previousAgeP
  } else {
    pLives <- lifeTable[ages >= inputAge, c("t_p_x0")]
  }

  pLivesAges <- lifeTable[ages >= inputAge, c("ages")]
  pCurveX <- data.frame(pLivesAges, pLives)

  #calculate probability tl1_q_x that x survives t years and dies within 1 year
  pCurveXRow <- nrow(pCurveX)

  #probability of dead based on input age
  pCurveX$tl1_q_x[1] <- 1 - pCurveX$pLives[1] # firt data important,,, to avoid bug
  for (j in 2:(pCurveXRow)) {

    #small bug which does not consider the 0|1_q_x data..... also add q_x of 1 in the last year automatically?
    pCurveX$tl1_q_x[j] <- pCurveX$pLives[j-1] - pCurveX$pLives[j]
    #pCurveX$tl1_q_x[j] <- pCurveX$pLives[j] - pCurveX$pLives[j+1] #i = u in the pdf 1 = t in pdf
  }
  #note: return a vector
  bFAge <- sample(pCurveX$pLivesAges, inputNumberClients, replace = T, pCurveX$tl1_q_x)


  return(bFAge)
}
```

Listing 9 calulates the probability of (x) lives t years using the defined function CalculateFinalAgeSemiPerfectData. It finds the probablity $_tp_{x0}$ of *inputAges - 1* and stores the result in *previousAgeP*. Then 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*. Then stores all the ages that are greater than or equal to *inputAges* in *pLivesAges*. After that, it creates a data frame called *pCurveX* that has two columns *pLivesAges* and *pLives*. After that, 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

$_{u|t}q_x =_u p_x$ - $_{u+t}p_x$. *pCurveXRow* stores the number of rows in *pCurveX* data frame. It iterates from 1 until *pCurveXRow*-1. 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, it stores the $_{u|t}q_x$ of the last row in *pCurveX* data frame.

Listing 10: Project1 File

```
#now creating a block of 10000 people who are in different ages and want
   different benefits
BussinessBlock <- function(rAge,rBenefit) {
  netSinglePremium <- lifeTable[lifeTable$ages == rAge, c("A_x")]*rBenefit
  return(netSinglePremium)
}

inputNumberClients <- as.numeric(inputsProject1[inputsProject1$label ==
   "inputNumberClients", c("value")])

bAge <- vector(mode="integer", length=inputNumberClients) #initialize
   variables
bBen <- vector(mode="integer", length=inputNumberClients)
bNps <- vector(mode="numeric", length=inputNumberClients)
bFAge <- vector(mode="integer", length=inputNumberClients)
maxAges <- max(lifeTable$ages)
lifeTableAges <- lifeTable$ages[ages < maxAges]# avoid picking max ages

#looping 10000
lifeTableAges <- data.frame(Age=ages) # creating a data frame that
   contains the ages
print("---------Calculating Lifetimes-----------")
print(paste("Number of clients: ", inputNumberClients))
lifeTableAsVector = as.vector(lifeTable[1,], mode = 'numeric') #create a
   vector representation of lifeTable so I can pass it to the c function
for (i in 2:lifeTableTotalRow)
{
  lifeTableAsVector = c(lifeTableAsVector, as.vector(lifeTable[i,], mode
    = 'numeric'))
}
dim = c(length(lifeTableAges$Age), 10, length(lifeTableAges$Age)) #bug if
   only put lifeTableAges -> length =1, 99 instead of 100
dyn.load("c_code.dll")
res = .C("for_loop", lifeTable=as.numeric(lifeTableAsVector), dim =
   as.integer(dim), bAge = as.numeric(bAge), bBen=as.integer(bBen), bNps
   = as.numeric(bNps), bFAge = as.numeric(bFAge), lifeTableAges =
   as.integer(lifeTableAges), inputNumberClients
   =as.integer(inputNumberClients))
bFAge = res[6]$bFAge
bAge = res[3]$bAge
bBen= res[4]$bBen
bNps= res[5]$bNps
```

Listing 10 creates a function named ***BussinessBlock*** that receives two parameters -age and benefit- and calculates and returns the Net Single Premium according to the formula $NetSinglePremium = Benefit * A_x$. Line 7 gets the population value which is 10000 from *inputsProject1* and stores it in *inputNumberClients*. Lines 9-12 creates new variables (arrays) and set them to zeros with same length of *inputNumberClients* for later use. Line 11 gets the maximum age from *lifeTable* data frame and stores in *maxAges*. Line 14 stores all the ages that are less than *maxAges* in *lifeTableAges*. Line 17 stores all the ages in *lifeTableAges*. The loop is made in a c code and called within R to reduce the time needed to loop over all 10000 clients.

Listing 11: Project1 File

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

investmentInterest <- as.numeric(inputsProject1[inputsProject1$label ==
    "investmentInterest", c("value")])

surviveYears <- 0
money <- sum(bDataFrame$NetSinglePremium)
earnInterest <- money*investmentInterest

benefitPayment <- 0
#find payment of the year
payment <- paymentTable[paymentTable$SurviveYears == 0, c("Benefit")]
if(length(payment) == 0){ #sometime there is no payment for specific
    year, so convert numeric(0) to 0
  benefitPayment <- 0
}else{
  benefitPayment <- payment
}
nYears <- max(paymentTable$SurviveYears)
for (i in 1:nYears) {
  surviveYears[i+1] <- i
  money[i+1] <- money[i] + earnInterest[i] - benefitPayment[i]
  if(money[i+1] > 0){ #only calculate earnInterest when money > 0
    earnInterest[i+1] <- money[i+1]*investmentInterest
  }else {
    earnInterest[i+1] <- 0
  }

```

```
32   #find payment of next year
33   payment <- paymentTable[paymentTable$SurviveYears == i, c("Benefit")]
34   if(length(payment) == 0){ #sometime there is no payment for specific
       year, so convert numeric(0) to 0
35     benefitPayment[i+1] <- 0
36   }else{
37     benefitPayment[i+1] <- payment
38   }
39 }
40 #benefitPayment[i+1] <- 0 #last year payment = 0
41 profitTable <- data.frame(surviveYears, money, earnInterest,
     benefitPayment)
```

In Listing 11, line 1 creates a data frame named *bDataFrame* that contains Age, Benefit, NetSinglePremium and Die columns. Line 2 adds new column to *bDataFrame* named *SurviveYears* that contains the difference between Die and Age column. Line 4 creates the CSV file, each time we run the code new file will be created and the older file will be replaced. Lines 5-6 creates a data frame named *paymentTable* that contains SurviveYears and Benefit columns from *bDataFrame* data frame. After that, we are gonna calculate and see whether the insurance company makes money or loses money with interest rate of 5.00%. Line 8 gets the interest value which is 0.05 from *inputsProject1* and stores it in *investmentInterest*. Line 10 sets the SurviveYear to zero which indicates the current year. Line 11 calculates the sum of NetSinglePremium column in *bDataFrame* data frame and stores the result in *money*. Line 12 multiplies *money* with *investmentInterest* and stores the result in *earnInterest*. Lines 16-29 calculates the payment of benefits in current year. Line 16 stores the benefits paid in the current year in *payment*. Lines 17-29 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*. Lines 32-39 calculates the benefit payment of next year. Line 33 gets the benefit that should be paid for i survived years. Lines 34-37 checks if the payment is equal to zero or not. If it is, the benefit payment for the next year will be zero and that meas nobody died this year. Otherwise, the benefit payment for the next year will be equal to *payment* that contains the benefit. Finally, Line 41 creates a data frame called *profitTable* that has four columns SurviveYear, money, earnInterest and benefitPayment.

As mentioned earlier, x11() function opens a new graph window before creating a new graph to avoid overwriting of previous plotted graphs. The following Listings 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 listing will have its output graph under it.

Listing 12: Project1 File

```
x11()
myGraph <- ggplot(bDataFrame, aes(Age))
myGraph <- myGraph + geom_histogram(binwidth=1, colour="black",
    fill="white") + labs(title = "Random Ages Histogram")
print(myGraph)
ggsave(filename = "images/Random Ages Histogram.png", plot = myGraph)
```

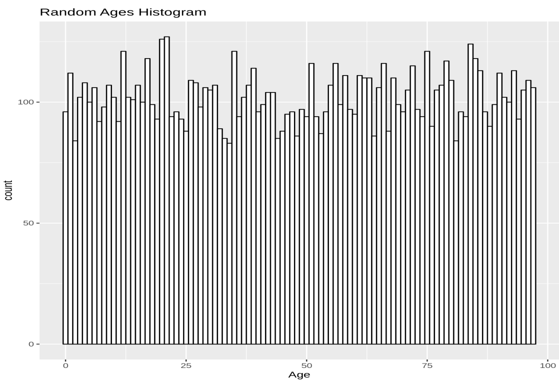

Figure 2: Random Ages Histogram

Listing 13: Project1 File

```
x11()
myGraph <- ggplot(bDataFrame, aes(Benefit))
myGraph <- myGraph + geom_histogram(colour="black", fill="white") +
    labs(title = "Random Benefit Histogram")
print(myGraph)
ggsave(filename = "images/Random Benefit Histogram.png", plot = myGraph)
```

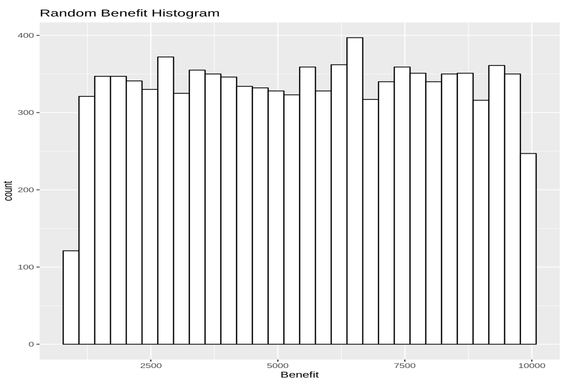

Figure 3: Random Benefit Histogram

Listing 14: Project1 File

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

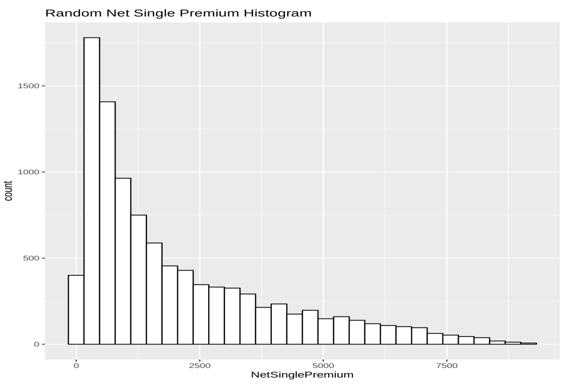

Figure 4: Random Net Single Premium Histogram

Listing 15: Project1 File

```
x11()
myGraph <- ggplot(bDataFrame, aes(Die))
myGraph <- myGraph + geom_histogram(binwidth=1, colour="black",
    fill="white") + labs(title = "Random Dead Ages Histogram")
print(myGraph)
ggsave(filename = "images/Random Dead Ages Histogram.png", plot = myGraph)
```

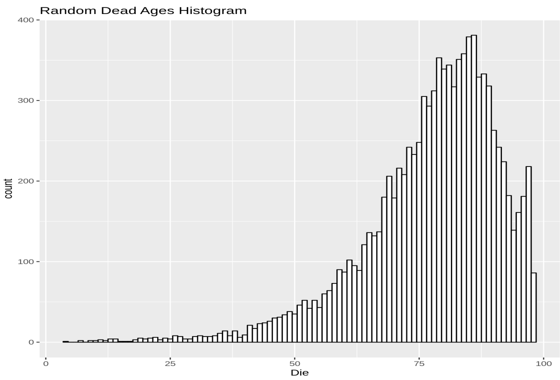

Figure 5: Random Dead Ages Histogram

Listing 16: Project1 File

```
x11()
myGraph <- ggplot(bDataFrame, aes(SurviveYears))
myGraph <- myGraph + geom_histogram(binwidth=1, colour="black",
    fill="white") + labs(title = "Random Survive Years Histogram")
print(myGraph)
ggsave(filename = "images/Random Survive Years Histogram.png", plot =
    myGraph)
```

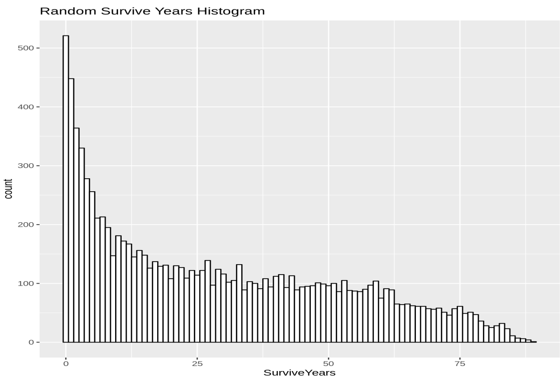

Figure 6: Random Survive Years Histogram

Listing 17: Project1 File

```
1 #------------------- print profit graph ------------------
2 meltProfitTable <- melt(profitTable, id="surviveYears")
3 x11()
4 myGraph <- ggplot(meltProfitTable, aes(x = surviveYears, y = value, colour = variable))
5 myGraph <- myGraph + geom_point() + labs(title="Company Profit Graph", y = "Money [$US]") + geom_line(linetype = "dashed") +
6   scale_color_manual(labels = c("Profit", "Interest", "Payment"), values = c("Green","blue", "red"))
7 print(myGraph)
8
9 ggsave(filename = "images/Company Profit Graph.png", plot = myGraph)
10
11 endTime <- Sys.time() # calulating the ending time
12 totalTime <- endTime - startTime #calulating the total time
13 print(totalTime) # printing the total time
```

Line 2 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 4 sets the axis of the plot. Line 5 sets the title and the plot will be plotted as lines and sets the legend of the plot. Line 7 displays the graph on the screen. Line 9 saves the graph on images directory which should be created on the same working directory. Line 11 stores the time when the code is ended its execution. Line 12 calculates the total time . Line 13 prints the total time. Figure 7 shows the output of line 7.

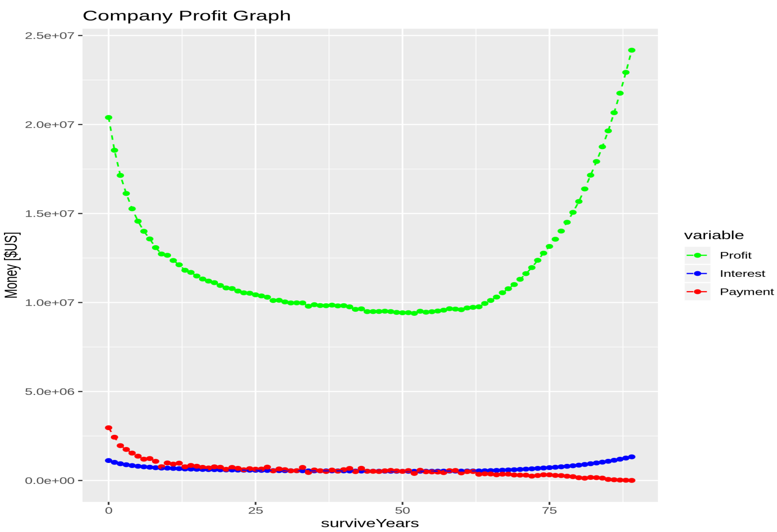

Figure 7: Company Profit Graph