**Data Management and Manipulation**

**Coursework**

Submitted by-

**Kaustubh Prakash Nanoti**

**20494432**

**Tables and Diagrams**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Table/Diagram** | **Table/Diagram Name** | **Page Number** |
| 1 | Diagram 1.3 | Tree Diagram to illustrate the scenario | 5 |
| 2 | Diagram 2.1 | Entity Relationship Diagram | 7 |
| 3 | Table 2.1 | Data Dictionary | 8 |

**Figures**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Figure Number** | **Figure Name** | **Page Number** |
| 1 | Fig. 1.1 | Implementation of Task1-a in R-Studio | 2 |
| 2 | Fig. 1.2 | Implementation of Task1-b in R-Studio | 4 |
| 3 | Fig. 1.3 | Implementation of Task1-c in R-Studio | 5 |
| 4 | Fig. 2.1 | Table Relationship Diagram | 9 |
| 5 | Fig. 2.2 | Implementation of Task2-b in R-Studio | 18 |
| 6 | Fig. 3.1 | Implementation of Task3 in R-Studio | 22 |

**Task 1**

1. A normal deck of cards has 52 cards equally divided into 4 suits. Two of the four suits consisting of 13 cards are black, and the remaining are red.

As per the task description, Let A has all the black cards and B has all the red cards. Therefore, both A and **B have 26** cards each, but as mentioned, B somehow manages to throw away 9 of the clubs from A’s cards, thus leaving **A with** only **25 cards**.

A and B randomly have to choose a card from their respective deck, and the one with the higher card value wins. For example, if A picks up a card with the value 1 and B picks up a card with the value 2, B wins the round. Repeat the same **1000 times** to determine the number of times B wins.

Following is the program in R to find out the same:

**Program:**

count <- 0 *#counter to determine the number of times B wins.*

*#Since A does not have 9 of the clubs, below is the list of cards with A.*

cardListA <- c(1,2,3,4,5,6,7,8,9,10,11,12,13,1,2,3,4,5,6,7,8,10,11,12,13)

*#Looping over 1000 times to find the winner*

for (i in 1:1000){

choiceA <- sample(cardListA, 1)  *#Ramdomly choosing a card from cardListA*

choiceB <- sample(1:13,1) *#Ramdomly choosing a card from a Red deck*

if(choiceA < choiceB){ *#Checking if B’s card is higher than that of A’s*

count = count + 1

}

i = i + 1

}

print(count)

**Output:**

451

1.

**Conclusion:**

After simulating for 1000 iterations, B wins 451 times. Calculating the probability of winning,

P(A) – Probability of A winning

P(B) – Probability of B winning

Thus, P(B) = Number of times B wins/Total number of iterations

P(B) = 451/1000 = 45.1%

P(A) = 1 – P(B) = 1 – 0.451 = 54.9%

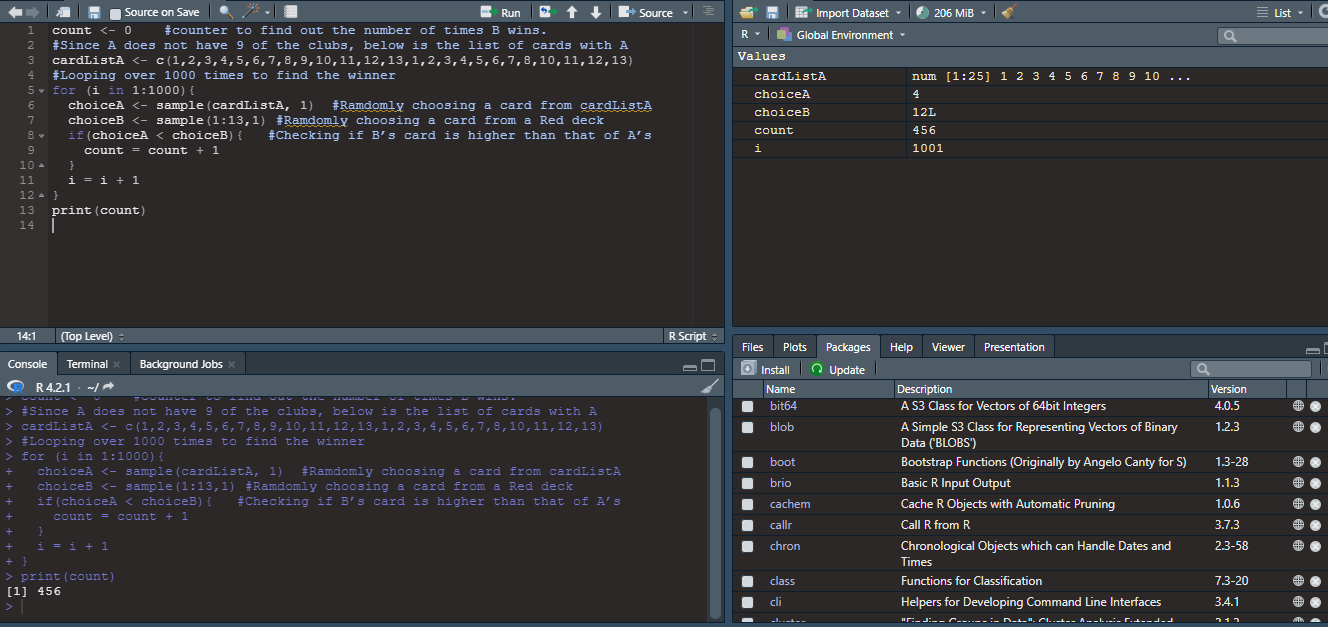


Fig. 1.1 Implementation of Task1-a in R-Studio

2.

1. The physics department tutor makes up the grades of every 10th student appearing for the exam without reading the paper. The University referred an external examiner to address this issue. The examiner randomly collects the sample of papers that equals the rounded square root of the students registered for the subject. The examiner can correctly identify the falsely marked exams 94% of the time and a marked exam as unmarked 0% of the time.

For the subject having 40, 80, and 160 students following is the program to estimate the probability tutor failing to escape.

**Program:**

students <- readline() *#Taking the number of students as input from the user.*

*#Number of students has to be an integer, converting the input to an integer.*

numOfStudents <- as.integer(students)

*#The examiner randomly selects the rounded-up square root of the students*

numOfRandomSample <- round(sqrt(numOfStudents))

falseMarked <- c() *#Let falseMarked be the list of students marked falsely.*

*#The number of times an examiner randomly selects falsely marked paper.*

count <- 0

for (i in 1: numOfStudents){ *#Iterating to find out the students marked falsely*

if(i %% 10 == 0){

falseMarked <- c(falseMarked, i)

}

}

sampleList <- c() *#List of randomly selected papers*

3.

for (k in 1: numOfRandomSample){ *#Finding the random samples for the examiner*

*#Selecting a random student from the class.*

sampleVal <- sample(1:numOfStudents, 1)

sampleList <- c(sampleList, sampleVal) *#List of random students selected*

*#Checking if the student selected by the examiner is falsely marked*

if(sampleVal %in% falseMarked){

*#The number of students the examiner has selected and are marked falsely*

count = count + 1

}

}*#The Examiner can identify falsely marked students 94% of the times*

probability <- 0.94\*(count/numOfStudents)

print(probability)

**Output:**

0.0235

**Conclusion:**

The tutor getting caught for his false deeds is rare, as the external examiner finds the falsely marked exam 2.35 times out of 100 for 40 students. The probability for 80 and 160 students is 11.75% and 5.875%, respectively. This value varies over time.

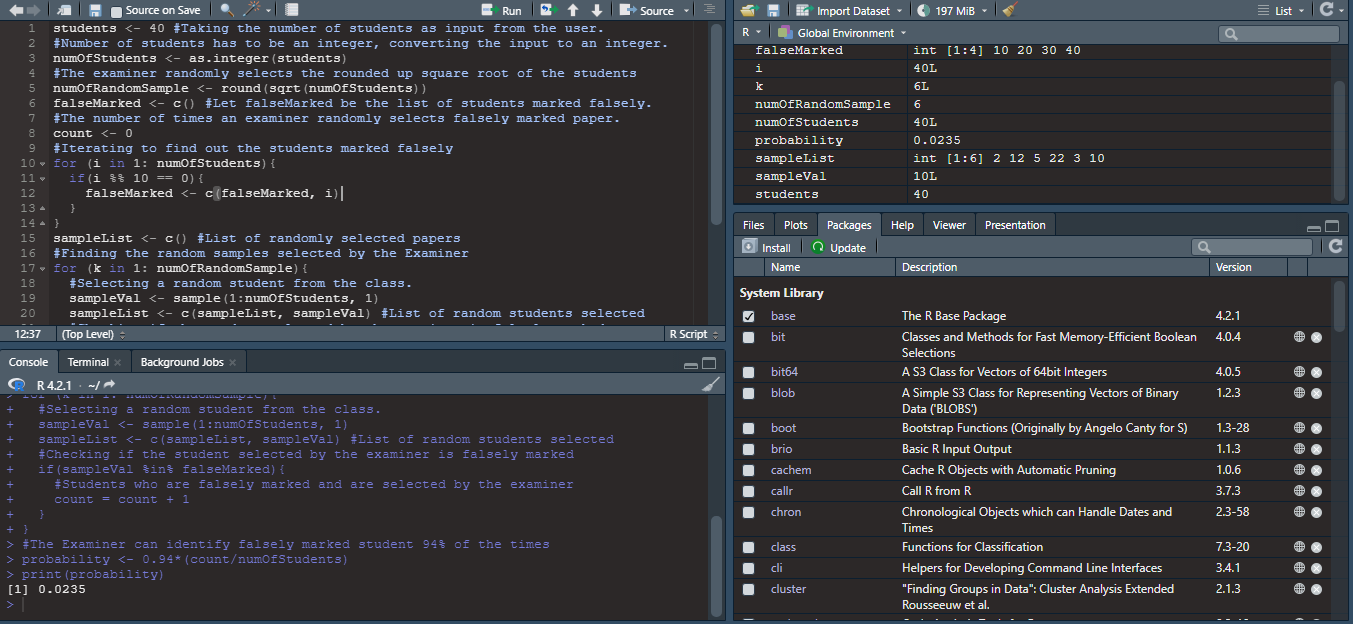


Fig. 1.2 Implementation of Task1-b in R-Studio

4.

1. A company's focus is estimating the probability of a product's success before its launch. R&D of a company (A) is working on a technology and has a 32% chance of perfecting it. However, a rival is working on similar technology with a 17% chance of perfecting and beating A in the market. If company A successfully perfects its technology and manages to launch it first, they have a 48% chance of success. If they fail to tackle first, this chance drops to 12%.

To better understand this problem, referring the Tree Diagram below.

**88%**

**12%**

Success

Failure

Launched Second

**52%**

**48%**

Failure

Success

Launched First

Perfecting

**68%**

**32%**

Not Perfecting

Product

Diagram 1.3 Tree Diagram to illustrate the scenario

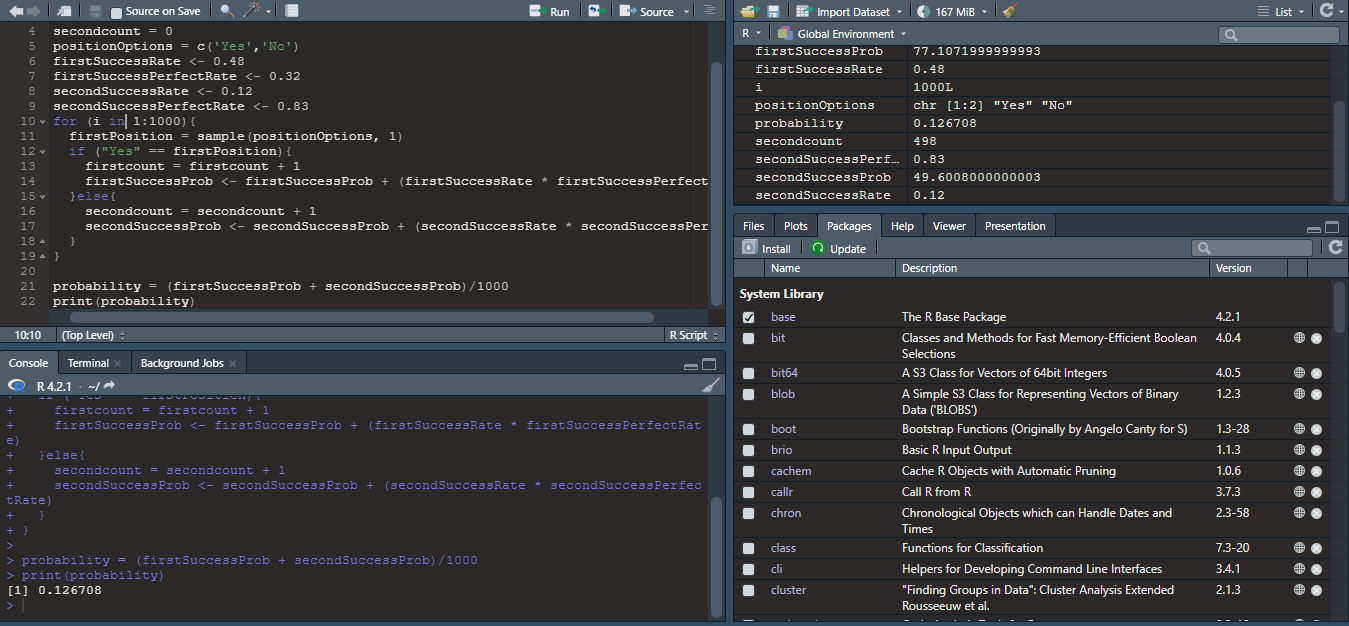


Fig. 1.3 Implementation of Task1-c in R-Studio

5.

**Program:**

firstSuccessProb <- 0 *#Probability of getting success after launching first*

secondSuccessProb <- 0 *#Probability of getting success after launching a second*

*#Options to decide whether the company can launch the product before its rival.*

positionOptions = c('Yes', 'No')

firstSuccessRate <- 0.48 *#Chances of success if the company can launch first.*

perfectingRate <- 0.32 *#Chances of perfecting the technology.*

secSuccessRate <- 0.12 *#Chances of success if the company fails to launch first.*

*#Looping over 1000 times to check the probability of the company managing to launch a product before the rival company*

for (i in 1:1000){

firstPosition = sample(positionOptions, 1) *#Choosing the position randomly*

if ("Yes" == firstPosition){ *#If Company launches first*

firstSuccessProb <- firstSuccessProb + (firstSuccessRate \* perfectingRate)

}else{ *#If Company fails to launch first*

secondSuccessProb <- secondSuccessProb + (secSuccessRate \* perfectingRate)

}

}

probability = ((firstSuccessProb + secondSuccessProb)/1000)\*100

print(probability)

**Output:**

12.6708

**Conclusion:**

On simulating, out of 1000 times, Company A manages to launch the product first 502 times, and thus, the success rate goes up to 48%, and therefore, the probability of a product being successful is 0.0766464, i.e., **7.2499%**. Simultaneously the likelihood of a product being successful when the company fails to launch the product before the rivals is 0.052588, i.e., **5.2588%**. Thus, company A's cumulative probability of the technology being successful is **12.6708%**.

6.

**Task 2**

Tiddly Inc is organizing its annual Tiddlywinks tournament with 32 players trying their best to win the prizes. The competition will take place in a single-elimination format where the players will be paired off and play in twos. Tiddly Inc wants to store the data of the winners of each game. All the players registering for the tournament must provide their name, sex, and phone number at the time of registration. The player winning their match will advance to the next stage. Tiddly Inc is interested in designing a data structure to store all the information about the tournament in the database.

1. As Tiddly Inc expects players’ details like Name, Gender, and Phone number at the time of registration, the **Players** entity expresses all these details about the player participating in the tournament. As the game progresses, the **Match** entity will represent all the match-related data, including the winner.

The following ER diagram represents the relationship between the two entities. The branches connected to the entities show the attributes of the entities. All the players participating in the tournament will play at least one match each.

Match

Players

Can play

Diagram 2.1 Entity Relationship Diagram

The diagram above illustrates the linkage between the two entities and shows **one-to-many** cardinalities. The player registering for the tournament can play more than one match, depending upon the match’s outcome in each round.

This cardinality is represented is following way:

One or more

Exactly one



7.

The entities from the ER Diagram 2.1 can be converted into the tables of the database with attributes as columns. Players\_Record can store the details of players registering for the tournament. The Match\_Records table can store the results of all the matches in various tournament rounds. Both these tables can be linked with each other using a **PlayerID**, i.e., a Primary Key of Players\_Record and **Winner** from Match\_Records which stores the ID of the player winning the match.

The following is the representation of the schema structure

**Players\_Record**

|  |  |  |  |
| --- | --- | --- | --- |
| PlayerID (PK) | PlayerName | Sex | PhoneNumber |

**Match\_Records**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| MatchNumber()PK | Round | Player1 | Player2 | Winner |

Following is the Data Dictionary that contains the information about the database.

Table 2.1 Data Dictionary –

**Players\_Record**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Field Length** | **Constraints** | **Description** |
| PlayerID | Number | 5 | Primary Key | Unique Identification Number of a Player |
| FullName | Varchar | 50 | Not Null | Name of the player participating, Last name followed by First name |
| Sex | Varchar | 25 | Not Null | Gender (Male/Female) |
| PhoneNumber | Number | 15 | Not Null | Contact Number of the player |

**Match\_Records**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Field Length** | **Constraints** | **Description** |
| MatchNumber | Number | 5 | Primary Key | Unique Identification Number of a Player |
| Round | Number | 5 | Not Null | Name of the player participating, Last name followed by First name |
| Player1 | Number | 5 | Not Null | Gender (Male/Female) |
| Player2 | Number | 5 | Not Null | Contact Number of the player |
| Winner | Number | 5 | Not Null | PlayerID of the winner |

8.

Fig. 2.2 is the representation of the relationship between the tables. The tables share one-to-many relationships. The winner’s details can be fetched from the Players\_Record table using the ID of a winner stored in the Match\_Records. The Unique ID of a winner is matched with the PlayerID from Players\_Record.

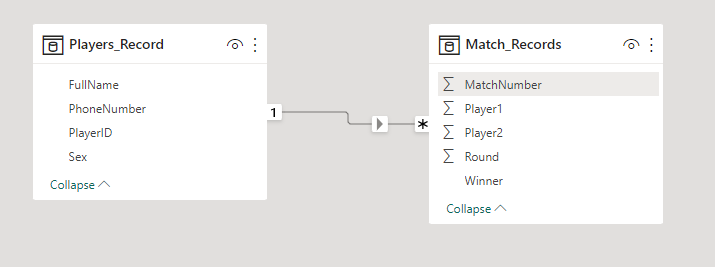


Fig. 2.1 Table Relationship Diagram

9.

b) Generating the data of 32 players and randomly choosing the winner of each game.

**Program:**

playersData = c() *#List to store players data*

fullNameList = c() *#List to store the name of the players*

sexList = c() *#List to store the gender of the players*

phoneNumList = c() *#List to store the phone number of the players*

playerIdList = c() *#List to store the Unique identification number of all the players*

for (i in 1:32){ *#Randomly generating the data of 32 players*

*#Selecting the gender of players*

gender <- sample(c("Male", "Female"),1)

*#Randomly selecting the name of the players*

fullName = randomNames::randomNames(1,gender,name.order="last.first",

name.sep=" ", sample.with.replacement=TRUE, return.complete.data=FALSE)

*#Selecting the phone number of the players*

phoneNum = sample(1000000000:9999999999, 1)

*#Storing the name of the player in fullNameList*

fullNameList = c(fullNameList, fullName)

*#Storing the gender of the player in sexList*

sexList = c(sexList, gender)

*#Storing the phone number of the player in phoneNumList*

phoneNumList = c(phoneNumList, phoneNum)

*#Storing the player Id in playerIdList*

playerIdList = c(playerIdList, i)

}

*#Storing all the player’s data in PlayersRecord and creating a dataFrame*

PlayersRecord = data.frame(PlayerID = playerIdList,FullName = fullNameList , Sex = sexList, PhoneNumber = phoneNumList)

write.csv(PlayersRecord, 'Players\_Record.csv')

print(PlayersRecord)

10.

matchRecord = c() *#List to store the match data*

player1List = c() *#List to store the data of player 1*

player2List = c() *#List to store the data of player 2*

matchNumList = c() *#List to store the match number*

*#List of players selected randomly out of the 32 players registered for the competition*

randListOfPlayers = c()

*#Selecting the random player out of 32 players registered*

player = sample(1:32,32, replace = FALSE)

*#Storing the randomly selected player in the randListOfPlayers*

randListOfPlayers = c(randListOfPlayers, player)

*#Printing the list of randomly selected players*

print(randListOfPlayers)

***#ROUND 1***

*#Randomly pairing off two players for competition and storing them in the list*

for (i in randListOfPlayers){

*#Storing the even-numbered players from randListOfPlayers in player2List and the odd-numbered players in player1List*

if(i %% 2 == 0){

player2List = c(player2List, randListOfPlayers[i])

}else{

player1List = c(player1List, randListOfPlayers[i])

}

}

*#List to store the winners of Round 1*

winnerList1 = c()

11.

for (i in 1:16){

*#Randomly selecting the winner of Round1 and storing it in winnerList1*

winner = sample(c(player1List[i], player2List[i]), 1)

winnerList1 = c(winnerList1, winner)

}

*#Randomly selecting the match number for Round 1*

matchNumList = sample(1:16, 16, replace = FALSE)

*#Storing the record of Round 1 in matchRecord1 and creating a dataFrame*

matchRecord1 = data.frame(Player1 = player1List , Player2 = player2List, MatchNumber = matchNumList, Winner = winnerList1)

print(matchRecord1)

***#ROUND 2***

*#List to store the data of Player 1 in Round 2*

player1List2 = c()

*#List to store the data of Player 2 in Round 2*

player2List2 = c()

*#Randomly storing the Player id of winners of round 1 in new list newWinnerList1*

newWinnerList1 = sample(winnerList1, 16, replace = FALSE)

print(newWinnerList1)

for (i in 1:16){

print(i)

*#Storing the even-numbered players from newWinnerList1* *in player2List2 and odd-numbered players in player1List2 for round 2*

if(i %% 2 == 0){

player2List2 = c(player2List2, newWinnerList1[i])

}else{

player1List2 = c(player1List2, newWinnerList1[i])

}

}

12.

*#List to store the player id of the winner of round 2*

winnerList2 = c()

for (i in 1:8){

*#Randomly selecting the winner out of two players for round 2 and storing it in the list*

winner = sample(c(player1List2[i], player2List2[i]), 1)

winnerList2 = c(winnerList2, winner)

}

*#Randomly selecting the match number for Round 2 and storing it in the matchNumList2*

matchNumList2 = sample(1:8, 8, replace = FALSE)

*#Storing the record of matches from Round 2 in matchRecord2 and creating a dataFrame*

matchRecord2 = data.frame(Player1 = player1List2 , Player2 = player2List2, MatchNumber = matchNumList2, Winner = winnerList2)

print(matchRecord2)

***#ROUND 3***

*#List to store the data of Player 1 in Round 3*

player1List3 = c()

*#List to store the data of Player 2 in Round 3*

player2List3 = c()

*#Randomly storing the Player id of winners of round 2 in new list newWinnerList2*

newWinnerList2 = sample(winnerList2, 8, replace = FALSE)

print(newWinnerList2)

13.

for (i in 1:8){

*#Storing the even-numbered players from newWinnerList2* *in player2List3 and odd-numbered players in player1List3 for round 3*

if(i %% 2 == 0){

player2List3 = c(player2List3, newWinnerList2[i])

}else{

player1List3 = c(player1List3, newWinnerList2[i])

}

}

*#List to store the player id of the winner of round 3*

winnerList3 = c()

for (i in 1:4){

*#Randomly selecting the winner out of two players for round 3 and storing it in the list*

winner = sample(c(player1List3[i], player2List3[i]), 1)

winnerList3 = c(winnerList3, winner)

}

*#Randomly selecting the match number for Round 3 and storing it in the matchNumList3*

matchNumList3 = sample(1:4, 4, replace = FALSE)

*#Storing the record of matches from Round 3 in matchRecord3 and creating a dataFrame*

matchRecord3 = data.frame(Player1 = player1List3 , Player2 = player2List3, MatchNumber = matchNumList3, Winner = winnerList3)

print(matchRecord3)

14.

***#ROUND 4***

*#List to store the data of Player 1 in Round 4*

player1List4 = c()

*#List to store the data of Player 2 in Round 4*

player2List4 = c()

*#Randomly storing the Player id of winners of round 3 in new list newWinnerList3*

newWinnerList3 = sample(winnerList3, 4, replace = FALSE)

print(newWinnerList3)

*#Storing the even-numbered players from newWinnerList3* *in player2List4 and odd-numbered players in player1List4 for round 4*

for (i in 1:4){

if(i %% 2 == 0){

player2List4 = c(player2List4, newWinnerList3[i])

}else{

player1List4 = c(player1List4, newWinnerList3[i])

}

}

*#List to store the player id of the winner of round 4*

winnerList4 = c()

for (i in 1:2){

*#Randomly selecting the winner out of two players for round 4 and storing it in the list*

winner = sample(c(player1List4[i], player2List4[i]), 1)

winnerList4 = c(winnerList4, winner)

}

15.

*#Randomly selecting the match number for Round 4 and storing it in the matchNumList4*

matchNumList4 = sample(1:2, 2, replace = FALSE)

*#Storing the record of matches from Round 4 in matchRecord4 and creating a dataFrame*

matchRecord4 = data.frame(Player1 = player1List4 , Player2 = player2List4, MatchNumber = matchNumList4, Winner = winnerList4)

print(matchRecord4)

***#ROUND 5***

*#List to store the data of Player 1 in Round 5*

player1List5 = c()

*#List to store the data of Player 2 in Round 5*

player2List5 = c()

*#Randomly storing the Player id of winners of round 4 in new list newWinnerList4*

newWinnerList4 = sample(winnerList4, 2, replace = FALSE)

print(newWinnerList4)

*#Storing the even-numbered players from newWinnerList4* *in player2List5 and odd-numbered players in player1List5 for round 5*

for (i in 1:2){

if(i %% 2 == 0){

player2List5 = c(player2List5, newWinnerList4[i])

}else{

player1List5 = c(player1List5, newWinnerList4[i])

}

}

*#List to store the player id of the winner of round 5*

winnerList5 = c()

16.

*#Randomly selecting the winner out of two players for round 5 and storing it in the list*

winner = sample(c(player1List5, player2List5), 1)

winnerList5 = c(winnerList5, winner)

*#Randomly selecting the match number for Round 5 and storing it in the matchNumList5*

matchNumList5 = sample(1:1, 1, replace = FALSE)

*#Storing the record of matches from Round 4 in matchRecord4 and creating a dataFrame*

matchRecord5 = data.frame(Player1 = player1List5 , Player2 = player2List5, MatchNumber = matchNumList5, Winner = winnerList5)

print(matchRecord5)

*#Combining all the round’s results in one data frame to form a table.*

finalMatchRecord = rbind(matchRecord1, matchRecord2, matchRecord3, matchRecord4,matchRecord5)

print(finalMatchRecord)

*#Creating a CSV file that stores the result of the tournament.*

write.csv(finalMatchRecord, 'Match\_Records.csv')

library(sqldf)

playerRecord <- read.csv("C:\\Users\\HP\\Documents\\Players\_Record.csv", sep = "," , header = TRUE)

finalResult <- read.csv("C:\\Users\\HP\\Documents\\Match\_Records.csv", sep = "," , header = TRUE)

*#Extracting the information of a winner combining the Player’s data and the Match results table*

winnerData = sqldf("SELECT pr.FullName, pr.Sex, pr.PhoneNumber FROM (SELECT \* FROM finalResult WHERE Round = 5) fr, playerRecord pr WHERE fr.Winner = pr.PlayerID")

print(winnerData)

**Output:**

"The details of the winner of the tournament are as follows:"

FullName Sex PhoneNumber

Martinez Jesus Male 7760031224

17.

**Conclusion:**

Tiddly Inc was interested in organizing a tournament with 32 players playing in a single-elimination format. The code above helps generate the data as expected by Tiddly Inc and store it in the database. The code randomly pairs off the candidates and predicts the face-off winner in each tournament round. The code ultimately indicates the tournament winner and returns that player's data. Here as the tournament was played between 32 applicants, the contest lasted for 5 rounds, and Martinez Jesus emerged as the winner.

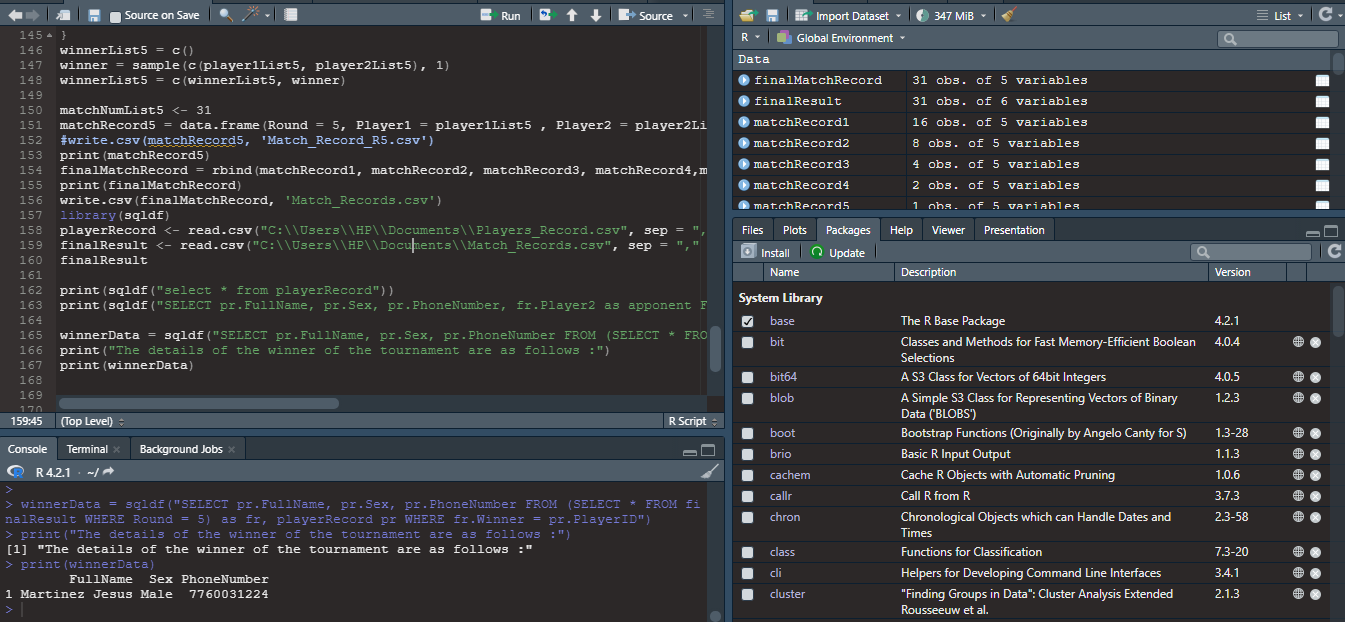


Fig. 2.2 Implementation of Task2-b in R-Studio

18.

**Task 3**

Diva wine company produces wines categorized as Red wine and White wine. The distribution of the quality of wine, when formulated by the Kernel-density estimator, is found to be between 4 to 8. The company's management is keen to identify the difference in the spread of the quality of the red and white wine measured by the standard deviation. The company has provided the data for all the wines in a CSV format.

The following program will first format the data and convert it into a usable form in R studio. Further, it will find the Standard deviation of both varieties of wines and the difference between them.

The code will generate 200 random records for each variety of wines 10 times to identify its relation with the given data.

**Program:**

*#Importing the given data frame in R and storing it in a vector wine*

wine <- read.csv("D:\\KAUSTUBH MS STUDY\\DATA MANAGEMENT AND MANIPULATION\\WINE.csv", sep = ";", header = TRUE)

*#Installing sqldf package to run SQL queries*

install.packages('sqldf')

install.packages("sfsmisc")

install.packages("ks") *#Installing ks package to kde plot*

library(sfsmisc)

library(ks)

library(sqldf)

*#Sagregatting the data based on the categories of the wine for simpler execution*

*#redWine vector stores all the data of Red Wine*

redWine <-sqldf("SELECT \* FROM wine WHERE COLOUR = 'red'")

*#whiteWine vector stores all the white wine data*

whiteWine <-sqldf("SELECT \* FROM wine WHERE COLOUR = 'white'")

*#The check the structure of newly created data Frames*

print(str(redWine))

print(str(whiteWine))

*#Both the data Frames have 200 records and 19 columns*

19.

*#Finding the Standard Deviation of quality of red wine and storing it in sdRedwine*

sdRedwine <- sd(redWine$quality)

*#Finding the SD of the quality of white wine and storing it in sdWhitewine*

sdWhitewine = sd(whiteWine$quality)

*#Calculating the range of sdRedwine and sdWhitewine and storing it in deviation*

*#Using the abs function to get the absolute value of the difference*

deviation = abs(sdWhitewine - sdRedwine)

*#Finding the distribution of the quality of the wine using kde*

d = wine$quality

hist(d) *#Plotting the histogram of the quality of wine*

kd=kde(d, H=2.25)

hist(d)

plot(kd)

*#Note: The quality of wine ranges from 4 to 8 as seen from the plot*

*#Storing the list of random values of quality of red wine in randomRed*

randomRed = c()

*#Storing the list of random values of quality of white wine in randomWhite*

randomWhite = c()

*#Storing the differences of SD of random values of quality of wines in randomDiffList*

randomDiffList = c()

*#Generating 200 records of the quality of wines for both Red and White 10 times to identify the randomness in the data.*

for (m in 1:10){  *#Generating the records 10 times to analyze randomness*

for (i in 1:200){ *#Generating 200 records for wines*

qualityRedWine = sample(4:8, 1) *#Randomly selecting quality of red wine*

qualityWhiteWine = sample(4:8, 1) *#Randomly selecting quality of white wine*

randomRed = c(randomRed, qualityRedWine) *#Storing data in the list*

randomWhite = c(randomWhite, qualityWhiteWine) *#Storing data in the list*

}

20.

randomSDRed = sd(randomRed) *#Calculating the SD of the quality of red wines*

*#Calculating the SD of the quality of white wines*

randomSDWhite = sd(randomWhite)

*#Calculating the difference between the SD of randomly generated samples of Red and White wine and storing it in the randomDiffList*

diff <- abs(randomSDRed – randomSDWhite)

randomDiff = diff

randomDiffList = c(randomDiffList, randomDiff)

}

greaterDiff <- 0 #If the random SD is greater than the actual SD

smallerDiff <- 0 #If the random SD is less than the actual SD

*#Looping over the list of random values to find out the count of values greater than actual and less than actual values.*

for (k in 1:10){

print(randomDiffList[k])

if(randomDiffList[k] >= deviation){

greaterDiff <- greaterDiff + 1

}

else{

smallerDiff <- smallerDiff + 1

}

}

print(randomDiffList)

print("The number of times random deviation is greater than the actual deviation is:")

print(greaterDiff)

print("The number of times random deviation is smaller than the actual deviation is :")

print(smallerDiff)

21.

**Output:**

The number of times random deviation is greater than the actual deviation is:

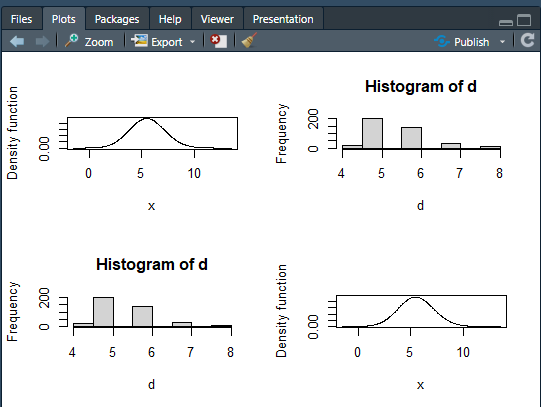
0

The number of times random deviation is less than the actual deviation is:

10

**Conclusion:**

After simulating 10 times for 200 random records of the quality of wine, generated over a distribution of 4 to 8, the difference in the standard deviation of actual data of red and white wine is not reproducible. The contrast of the standard deviation of random data is less than that of the real data. From the test above, it can be concluded that there **is no parent effect**. The deviation in actual data occurred only by chance and is random.



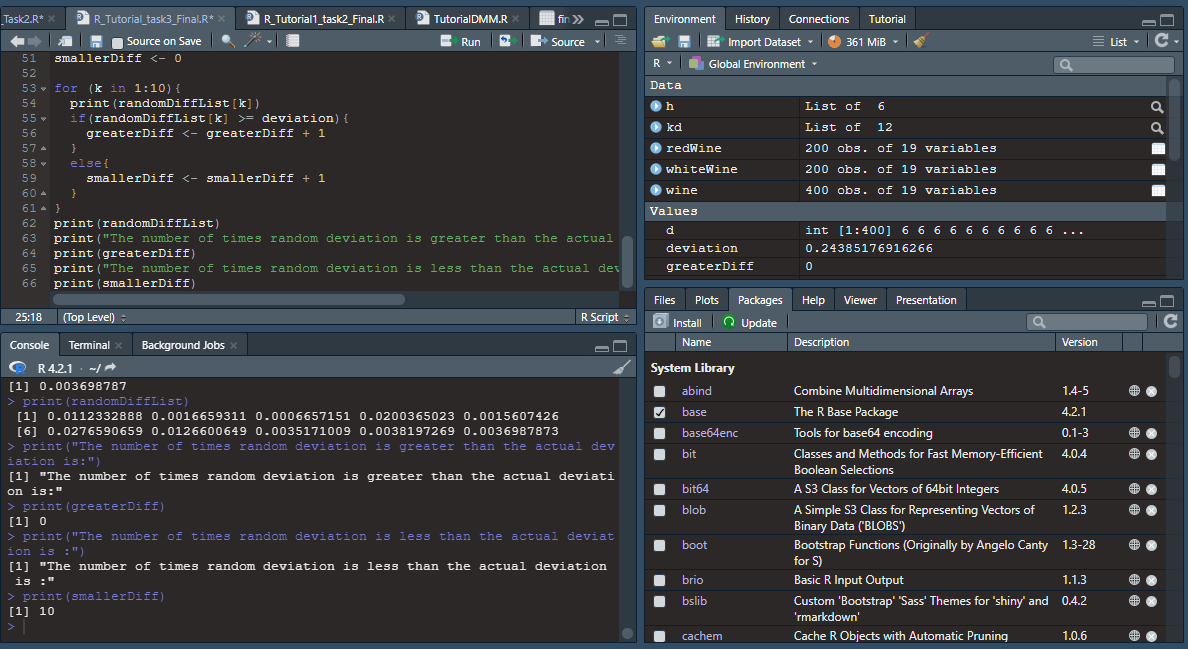


Fig. 3.1 Implementation of Task3 in R-Studio

22.