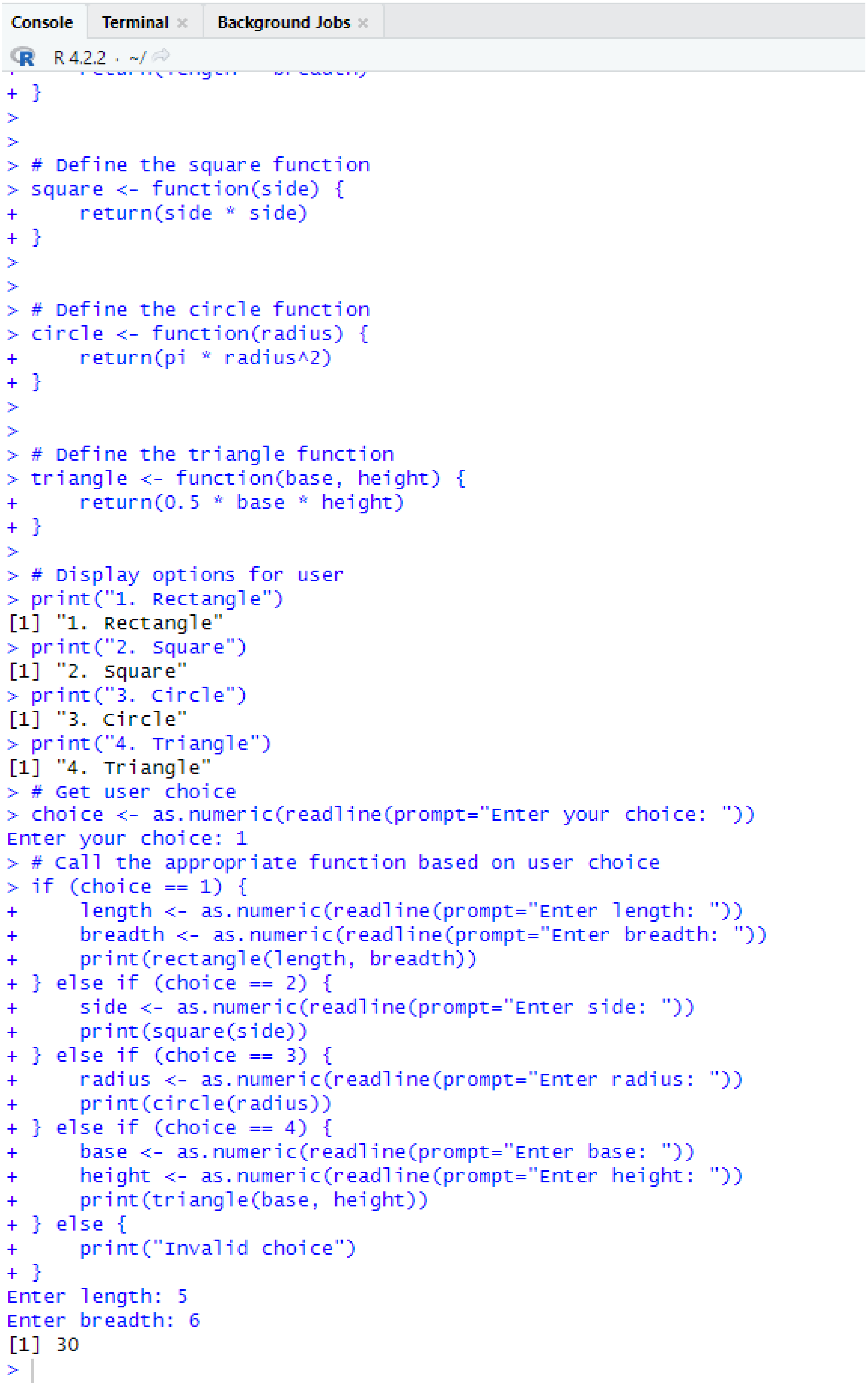
Data Mining Assignment 1

**Q1. Define four functions,
rectangle, square, circle and triangle, in one script for computing the area of respective figures. Rectangle function will take two arguments (length and breadth), square will consider only one argument (side), circle will consider one argument (radius) and triangle will consider two arguments
(base and height). Display the option 1. Rectangle, 2. Square, 3. Circle, 4. Triangle. Depending upon the user choice, the respective function will print the area of the desired shape.**

```r
# Define the rectangle function
rectangle <- function(length, breadth) {
  return(length * breadth)
}


# Define the square function
square <- function(side) {
  return(side * side)
}


# Define the circle function
circle <- function(radius) {
  return(pi * radius^2)
}


# Define the triangle function
```

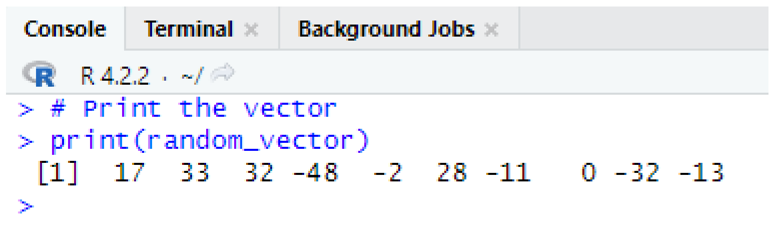
```r
triangle <- function(base, height) {
  return(0.5 * base * height)
}


# Display options for user
print("1. Rectangle")
print("2. Square")
print("3. Circle")
print("4. Triangle")


# Get user choice
choice <- as.numeric(readline(prompt="Enter your choice: "))


# Call the appropriate function based on user choice
if (choice == 1) {
  length <- as.numeric(readline(prompt="Enter length: "))
  breadth <- as.numeric(readline(prompt="Enter breadth: "))
  print(rectangle(length, breadth))
} else if (choice == 2) {
  side <- as.numeric(readline(prompt="Enter side: "))
  print(square(side))
} else if (choice == 3) {
  radius <- as.numeric(readline(prompt="Enter radius: "))
  print(circle(radius))
} else if (choice == 4) {
  base <- as.numeric(readline(prompt="Enter base: "))
  height <- as.numeric(readline(prompt="Enter height: "))
  print(triangle(base, height))
} else {
  print("Invalid choice")
}
```



```
+ }
>
>
> # Define the square function
> square <- function(side) {
+     return(side * side)
+ }
>
>
> # Define the circle function
> circle <- function(radius) {
+     return(pi * radius^2)
+ }
>
>
> # Define the triangle function
> triangle <- function(base, height) {
+     return(0.5 * base * height)
+ }
>
> # Display options for user
> print("1. Rectangle")
[1] "1. Rectangle"
> print("2. Square")
[1] "2. Square"
> print("3. Circle")
[1] "3. Circle"
> print("4. Triangle")
[1] "4. Triangle"
> # Get user choice
> choice <- as.numeric(readline(prompt="Enter your choice: "))
Enter your choice: 1
> # Call the appropriate function based on user choice
> if (choice == 1) {
+     length <- as.numeric(readline(prompt="Enter length: "))
+     breadth <- as.numeric(readline(prompt="Enter breadth: "))
+     print(rectangle(length, breadth))
+ } else if (choice == 2) {
+     side <- as.numeric(readline(prompt="Enter side: "))
+     print(square(side))
+ } else if (choice == 3) {
+     radius <- as.numeric(readline(prompt="Enter radius: "))
+     print(circle(radius))
+ } else if (choice == 4) {
+     base <- as.numeric(readline(prompt="Enter base: "))
+     height <- as.numeric(readline(prompt="Enter height: "))
+     print(triangle(base, height))
+ } else {
+     print("Invalid choice")
+ }
Enter length: 5
Enter breadth: 6
[1] 30
>
```

Q2. Write a R script to create a vector which contains 10 random integer values between -50 and +50.

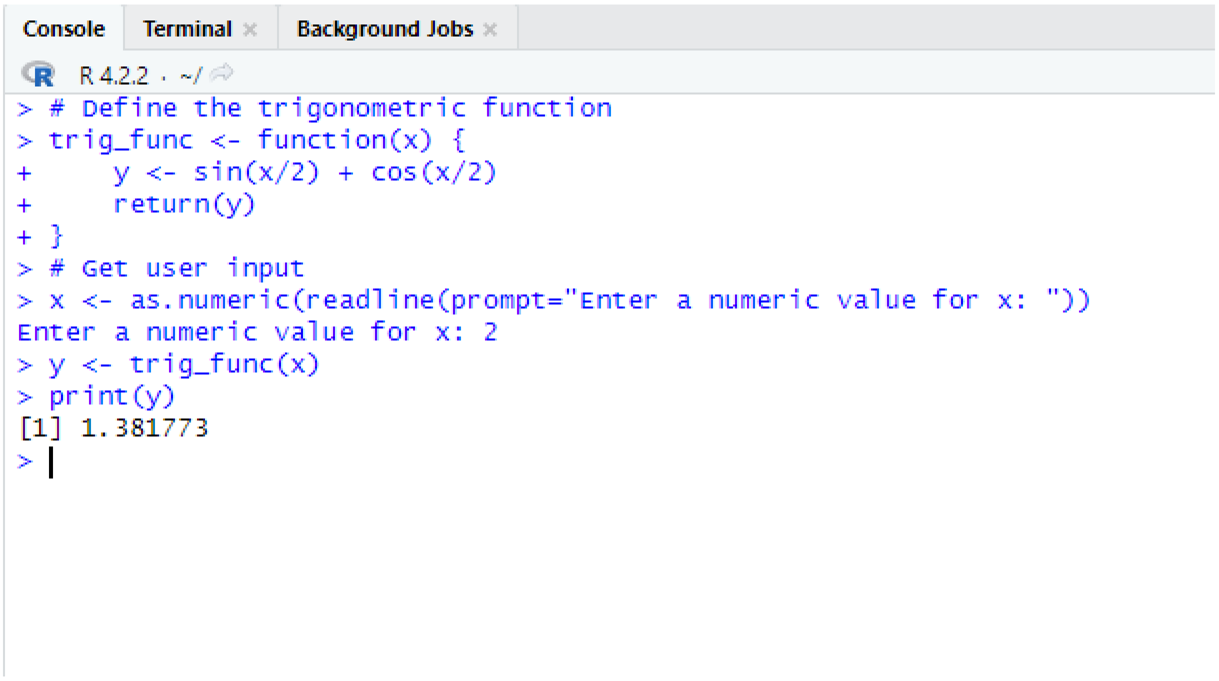
```
# Create a vector of 10 random integers between -50 and 50
random_vector <- sample(x = -50:50, size = 10, replace = TRUE)
# Print the vector
print(random_vector
```



```
> # Print the vector
> print(random_vector)
 [1]  17  33  32 -48  -2  28 -11   0 -32 -13
>
```

Q3. . y = sin(x/2) + cos(x/2) is a trigonometric function. Create a user-defined function – via function() - that implements the trigonometric function above, accepts numeric values, "x," calculates and returns values "y."

```
# Define the trigonometric function
trig_func <- function(x) {
 y <- sin(x/2) + cos(x/2)
 return(y)
}
# Get user input
x <- as.numeric(readline(prompt="Enter a numeric value for x: "))
# Calculate and print the result
y <- trig_func(x)
print(y)
```


Console Terminal Background Jobs

R 4.2.2 · ~/

```
> # Define the trigonometric function
> trig_func <- function(x) {
+     y <- sin(x/2) + cos(x/2)
+     return(y)
+ }
> # Get user input
> x <- as.numeric(readline(prompt="Enter a numeric value for x: "))
Enter a numeric value for x: 2
> y <- trig_func(x)
> print(y)
[1] 1.381773
> |
```

Data Mining Assignment 2

Q1. Write a R script to add 7 to each element in a given vector. Print the original and new vector.

```
> vector<-c(1,2,3,4)
> new_vector<-vector+7
> cat("Original Vector:",vector,"\n")
Original Vector: 1 2 3 4
> cat("New Vector:",new_vector,"\n")
New Vector: 8 9 10 11
> |
```

Q2. Write a R script to test whether the value of the element of a given vector greater than 65 or not. Return TRUE or FALSE.

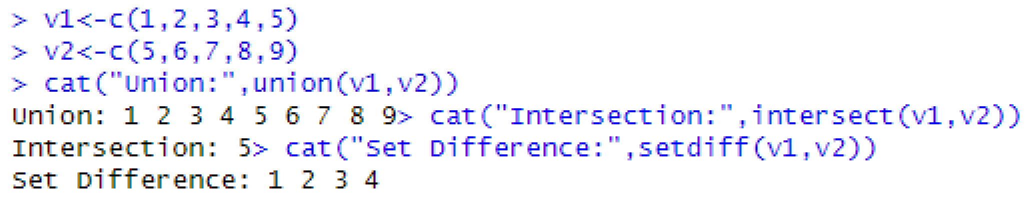
```
> vector<-c(60,62,65,67,69)
> cat("Result",vector>65)
Result FALSE FALSE FALSE TRUE TRUE
>
```

Q3. Write a R script to create a vector which contains 10 random integer values between -50 and +50.

```
> vector<-sample(seq(-50,50),10,replace=TRUE)
> cat("Vector:",vector)
Vector: -20 -19 20 -50 -44 -33 2 17 25 -37
```

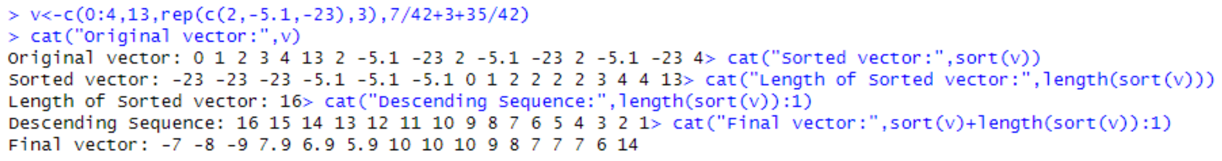
Q4. Read about and use the following functions on numeric vectors:
a. union

b. intersect
c. setdiff



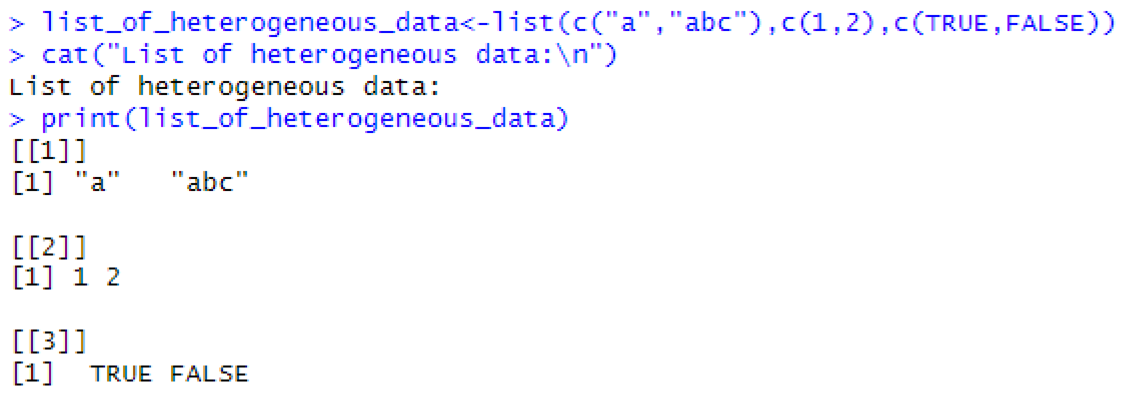
```
> v1<-c(1,2,3,4,5)
> v2<-c(5,6,7,8,9)
> cat("Union:",union(v1,v2))
Union: 1 2 3 4 5 6 7 8 9> cat("Intersection:",intersect(v1,v2))
Intersection: 5> cat("Set Difference:",setdiff(v1,v2))
Set Difference: 1 2 3 4
```

Q5. Write an R script for the following operations:-
a. Create a vector that contains the following, in this order, and output the final, resulting vector. Do not round any values, unless requested. * A sequence of integers from 0 to 4, inclusive. * The number 13 * Three repetitions of the vector c(2, -5.1, -23). * The arithmetic sum of 7/42, 3 and 35/42
b. Sort the vector created in (a) in ascending order. Output this result. Determine the length of the resulting vector and assign to “L”. Output L. Generate a descending sequence starting with L and ending with 1. Add this descending sequence arithmetically the sorted vector. This is vector addition, not vector combination. Output the contents. Do not round any values.



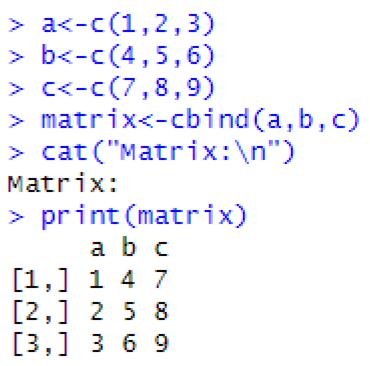
```
> v<-c(0:4,13,rep(c(2,-5.1,-23),3),7/42+3+35/42)
> cat("Original vector:",v)
Original vector: 0 1 2 3 4 13 2 -5.1 -23 2 -5.1 -23 2 -5.1 -23 4> cat("Sorted vector:",sort(v))
Sorted vector: -23 -23 -23 -5.1 -5.1 -5.1 0 1 2 2 2 2 3 4 4 13> cat("Length of Sorted vector:",length(sort(v)))
Length of Sorted vector: 16> cat("Descending Sequence:",length(sort(v)):1)
Descending Sequence: 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1> cat("Final vector:",sort(v)+length(sort(v)):1)
Final vector: -7 -8 -9 7.9 6.9 5.9 10 10 10 9 8 7 7 7 6 14
```

Q6. Write a R program to create a list of heterogeneous data, which include character, numeric and logical vectors. Print the lists.



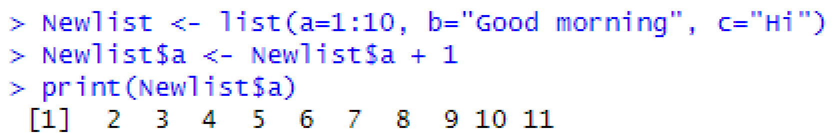
```
> list_of_heterogeneous_data<-list(c("a","abc"),c(1,2),c(TRUE,FALSE))
> cat("List of heterogeneous data:\n")
List of heterogeneous data:
> print(list_of_heterogeneous_data)
[[1]]
[1] "a"   "abc"

[[2]]
[1] 1 2

[[3]]
[1]  TRUE FALSE
```

Q7. Write a R program to create three vectors a,b,c with 3 integers. Combine the three vectors to become a 3×3 matrix where each column represents a vector. Print the content of the matrix.



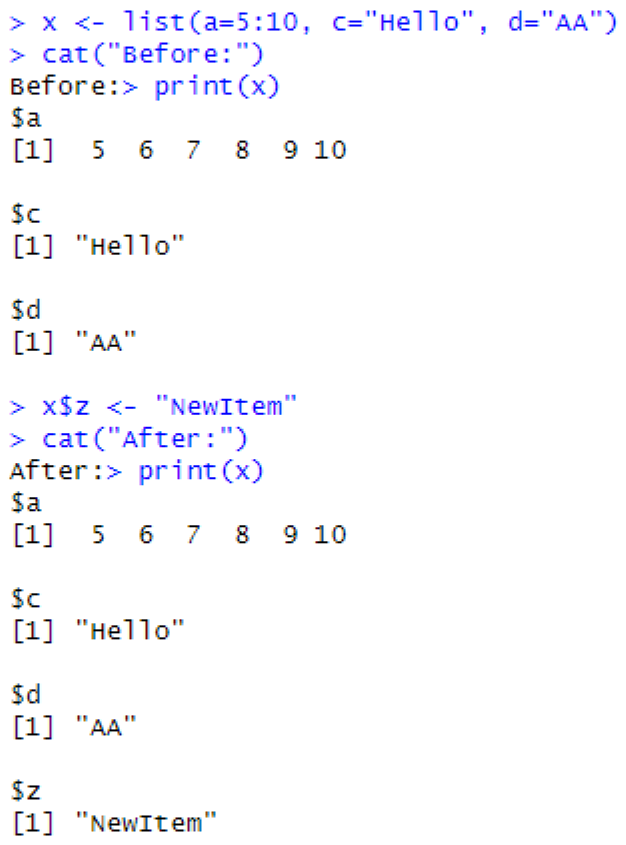
```
> a<-c(1,2,3)
> b<-c(4,5,6)
> c<-c(7,8,9)
> matrix<-cbind(a,b,c)
> cat("Matrix:\n")
Matrix:
> print(matrix)
     a b c
[1,] 1 4 7
[2,] 2 5 8
[3,] 3 6 9
```

Q8. If Newlist <- list(a=1:10, b="Good morning", c="Hi"), write an R statement that will add 1 to each element of the first vector in Newlist.



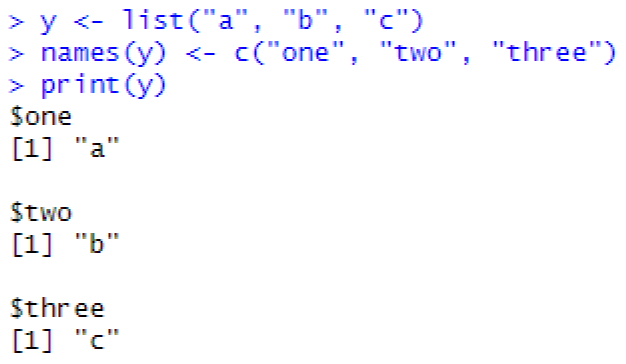
```
> Newlist <- list(a=1:10, b="Good morning", c="Hi")
> Newlist$a <- Newlist$a + 1
> print(Newlist$a)
 [1]  2  3  4  5  6  7  8  9 10 11
```

Q9. Let x <- list(a=5:10, c="Hello", d="AA"), write an R statement to add a new item z = "NewItem" to the list x.



```
> x <- list(a=5:10, c="Hello", d="AA")
> cat("Before:")
Before:> print(x)
$a
[1]  5  6  7  8  9 10

$c
[1] "Hello"

$d
[1] "AA"

> x$z <- "NewItem"
> cat("After:")
After:> print(x)
$a
[1]  5  6  7  8  9 10

$c
[1] "Hello"

$d
[1] "AA"

$z
[1] "NewItem"
```

Q10. Consider y <- list("a", "b", "c"), write an R statement that will assign new names "one", "two" and "three" to the elements of y.



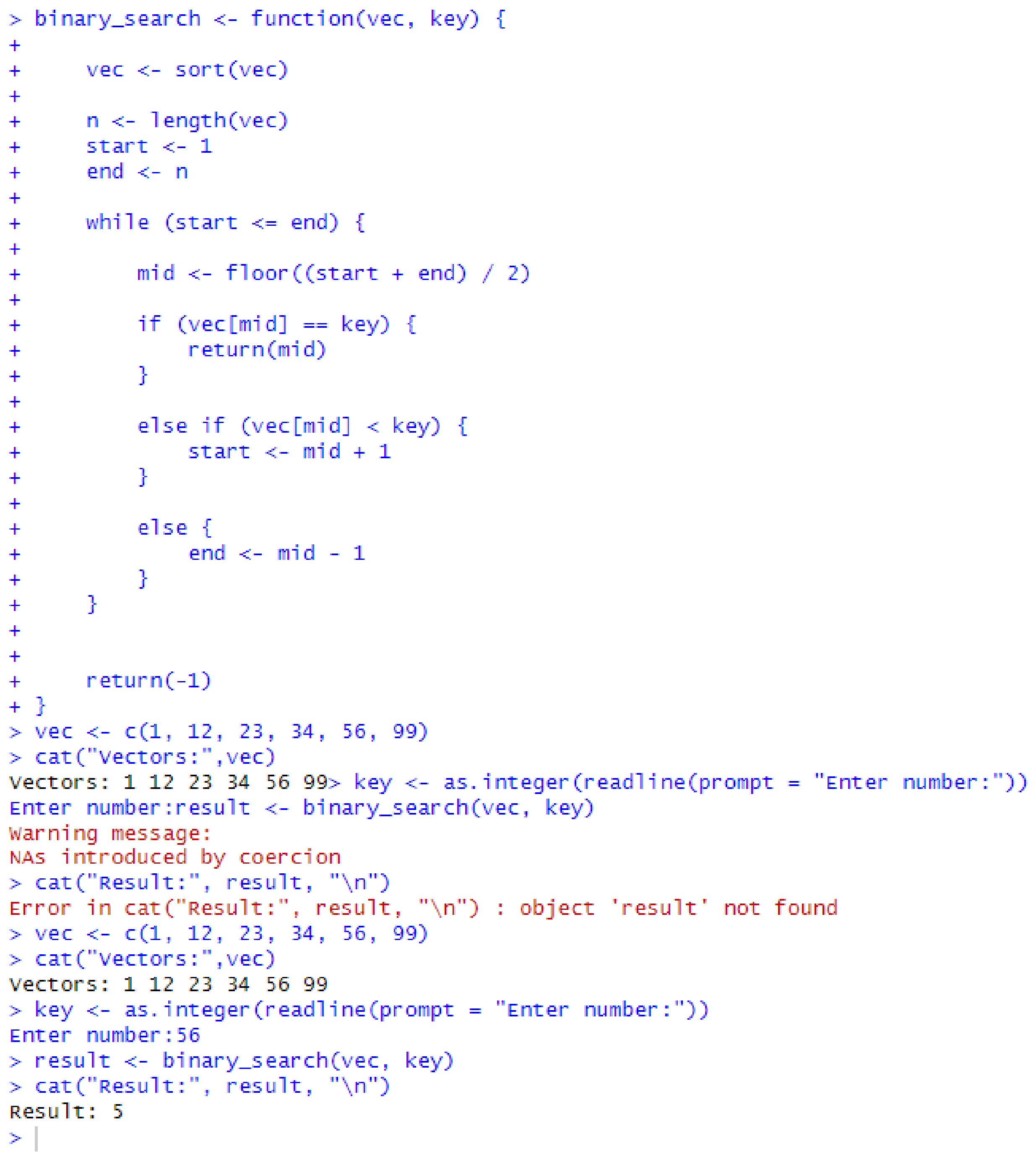
```
> y <- list("a", "b", "c")
> names(y) <- c("one", "two", "three")
> print(y)
$one
[1] "a"

$two
[1] "b"

$three
[1] "c"
```

Q11. Use R to create the following two matrices and do the indicated matrix multiplication.

$$\begin{bmatrix} 7 & 9 & 12 \\ 2 & 4 & 13 \end{bmatrix} \times \begin{bmatrix} 1 & 7 & 12 & 19 \\ 2 & 8 & 13 & 20 \\ 3 & 9 & 14 & 21 \end{bmatrix}$$

What is the resulting matrix?

Q12. Write a function to implement binary search(key) where key is taken as a user input. Consider the vector as (1,12,23,34,56,99).

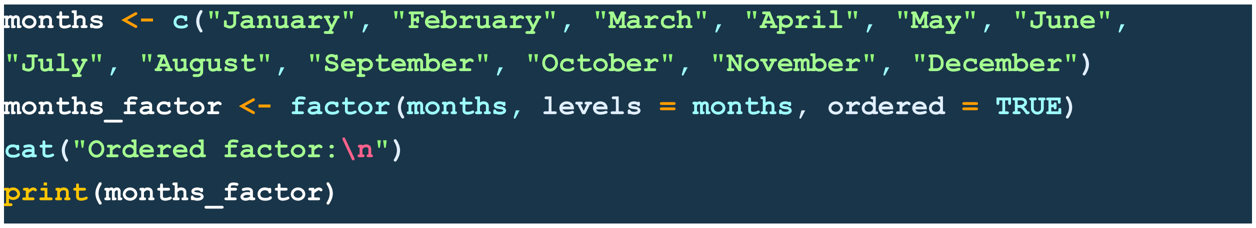


```
> binary_search <- function(vec, key) {
+ 
+     vec <- sort(vec)
+ 
+     n <- length(vec)
+     start <- 1
+     end <- n
+ 
+     while (start <= end) {
+ 
+         mid <- floor((start + end) / 2)
+ 
+         if (vec[mid] == key) {
+             return(mid)
+         }
+ 
+         else if (vec[mid] < key) {
+             start <- mid + 1
+         }
+ 
+         else {
+             end <- mid - 1
+         }
+     }
+ 
+ 
+     return(-1)
+ }
> vec <- c(1, 12, 23, 34, 56, 99)
> cat("Vectors:",vec)
Vectors: 1 12 23 34 56 99> key <- as.integer(readline(prompt = "Enter number:"))
Enter number:result <- binary_search(vec, key)
Warning message:
NAs introduced by coercion 
> cat("Result:", result, "\n")
Error in cat("Result:", result, "\n") : object 'result' not found
> vec <- c(1, 12, 23, 34, 56, 99)
> cat("Vectors:",vec)
Vectors: 1 12 23 34 56 99
> key <- as.integer(readline(prompt = "Enter number:"))
Enter number:56
> result <- binary_search(vec, key)
> cat("Result:", result, "\n")
Result: 5 
> 
```

Q13. Write a R program to create a two-dimensional 5x3 array of sequence of even integers greater than 50.

```
array <- matrix(seq(52, 92, by = 2), nrow = 5, ncol = 3, byrow = TRUE)
cat("Two-dimensional array:\n")
print(array)
```

Q14. Write a R program to create an ordered factor from data consisting of the names of months.



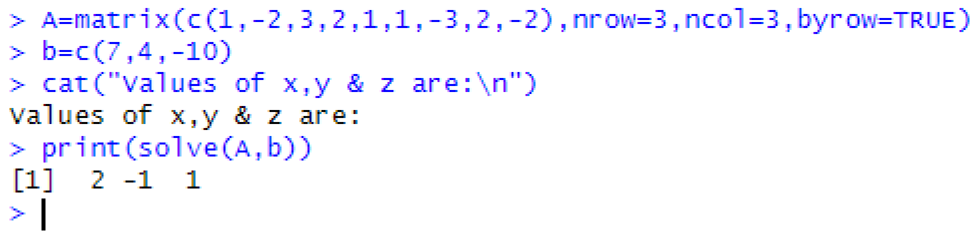
```r
months <- c("January", "February", "March", "April", "May", "June",
"July", "August", "September", "October", "November", "December")
months_factor <- factor(months, levels = months, ordered = TRUE)
cat("Ordered factor:\n")
print(months_factor)
```

Q15. Write a code in R to solve the following system of equations:

x – 2y + 3z = 7

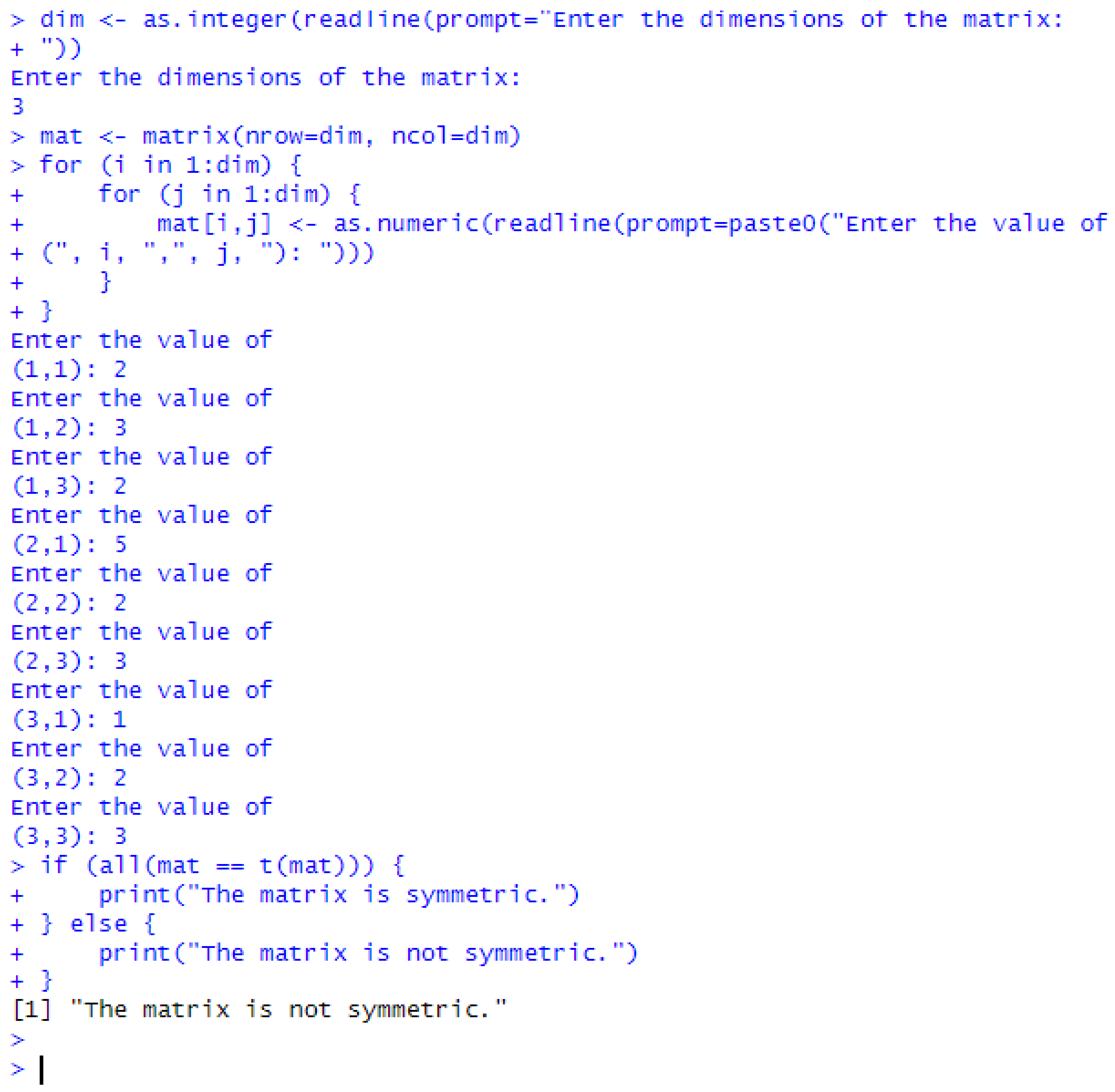
2x + y +z =4

-3x + 2y – 2z = -10



```
> A=matrix(c(1,-2,3,2,1,1,-3,2,-2),nrow=3,ncol=3,byrow=TRUE)
> b=c(7,4,-10)
> cat("Values of x,y & z are:\n")
Values of x,y & z are:
> print(solve(A,b))
[1]  2 -1  1
> |
```

Q16. Write a R code to take a matrix as an input from the user and print if it's a symmetric matrix or not. Also, take the dimension of the matrix as an input from the user.



```
> dim <- as.integer(readline(prompt="Enter the dimensions of the matrix:
+ "))
Enter the dimensions of the matrix:
3
> mat <- matrix(nrow=dim, ncol=dim)
> for (i in 1:dim) {
+     for (j in 1:dim) {
+         mat[i,j] <- as.numeric(readline(prompt=paste0("Enter the value of
+ (", i, ",", j, "): ")))
+     }
+ }
Enter the value of
(1,1): 2
Enter the value of
(1,2): 3
Enter the value of
(1,3): 2
Enter the value of
(2,1): 5
Enter the value of
(2,2): 2
Enter the value of
(2,3): 3
Enter the value of
(3,1): 1
Enter the value of
(3,2): 2
Enter the value of
(3,3): 3
> if (all(mat == t(mat))) {
+     print("The matrix is symmetric.")
+ } else {
+     print("The matrix is not symmetric.")
+ }
[1] "The matrix is not symmetric."
>
> |
```

Q17. Write statements in R to perform the following:
(i) Assume x<- c(0,1,1,2,3,5,8,13,21,34,NA,11,NA), find mean and median of x.
(ii) Create a 3X3 matrix with the values of square of 1st nine natural numbers.
(iii) To find transpose of a matrix.
(iv) To multiply two 3X3 matrices and store result in a matrix r.
(v) To concatenate two lists L1 and L2.

(vi) To write a loop to print all even elements of a vector
(vii) To sort a vector in ascending order

```
> #Q-17
> #(i) Mean and median of x:
> x <- c(0,1,1,2,3,5,8,13,21,34,NA,11,NA)
> mean_x <- mean(x, na.rm=TRUE)
> median_x <- median(x, na.rm=TRUE)
> print(paste("Mean of x:", mean_x))
[1] "Mean of x: 9"
> print(paste("Median of x:", median_x))
[1] "Median of x: 5"
> #(ii) 3X3 matrix of square of 1st nine natural numbers:
> matrix_sq <- matrix(data=1:9, nrow=3, ncol=3, byrow=TRUE)^2
> print("Matrix of square of 1st nine natural numbers:")
[1] "Matrix of square of 1st nine natural numbers:"
> print(matrix_sq)
     [,1] [,2] [,3]
[1,]    1    4    9
[2,]   16   25   36
[3,]   49   64   81
> #(iii) Transpose of a matrix:
> matrix_transpose<-t(matrix_sq)
> print("Transpose of the matrix:")
[1] "Transpose of the matrix:"
> print(matrix_transpose)
     [,1] [,2] [,3]
[1,]    1   16   49
[2,]    4   25   64
[3,]    9   36   81
> #(iv) Multiplying two 3X3 matrices:
> matrix1 <- matrix(1:9, nrow=3, ncol=3, byrow=TRUE)
> matrix2 <- matrix(2:10, nrow=3, ncol=3, byrow=TRUE)
> r <- matrix1 %*% matrix2
> print("Resultant matrix after multiplying two matrices:")
[1] "Resultant matrix after multiplying two matrices:"
> print(r)
     [,1] [,2] [,3]
[1,]   36   42   48
[2,]   81   96  111
[3,]  126  150  174
> #(v) Concatenating two lists L1 and L2:
> L1 <- list(a=1:10, b="Good morning")
> L2 <- list(c=2:5, d="Hi")
> L3 <- c(L1, L2)
> print("Concatenated list:")
[1] "Concatenated list:"
> print(L3)
$a
 [1]  1  2  3  4  5  6  7  8  9 10

$b
[1] "Good morning"

$c
[1] 2 3 4 5
```

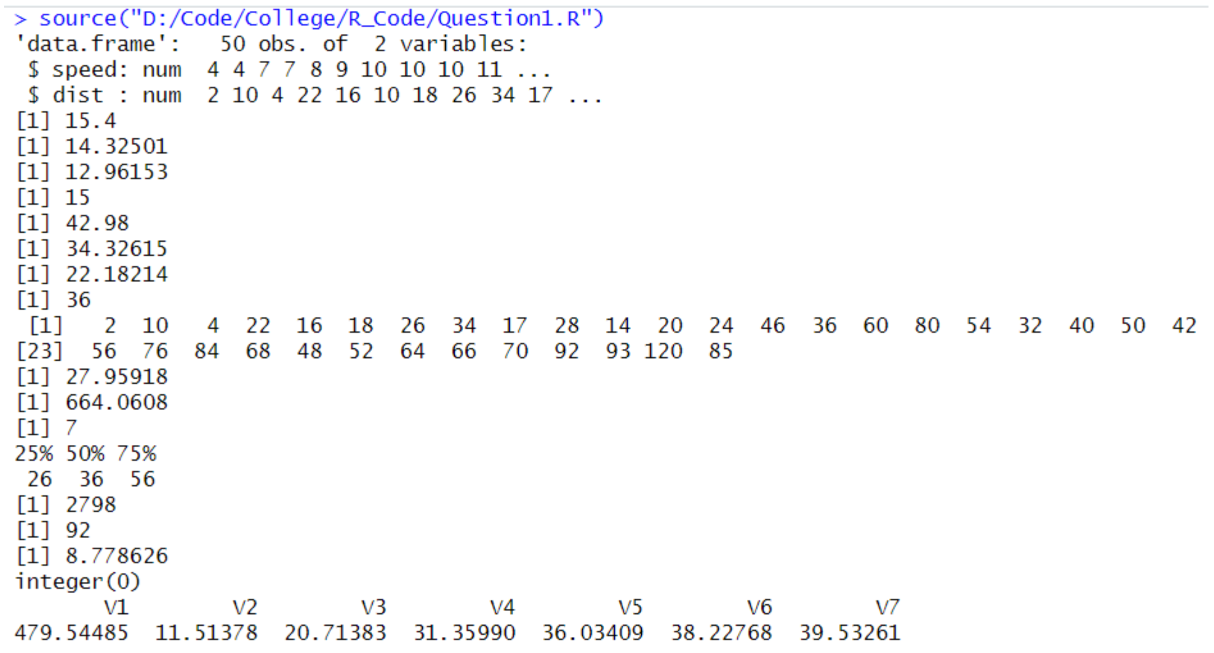
Data Preprocessing

```
# Question-1
# Load the dataset
data("cars")


# Display the structure of the dataset
str(cars)


# Compute the mean, geometric mean, harmonic mean, and median of speed and
dist columns
speed_mean <- mean(cars$speed)
print(speed_mean)
speed_gmean <- exp(mean(log(cars$speed)))
print(speed_gmean)
speed_hmean <- length(cars$speed) / sum(1/cars$speed)
print(speed_hmean)
speed_median <- median(cars$speed)
print(speed_median)


dist_mean <- mean(cars$dist)
print(dist_mean)
dist_gmean <- exp(mean(log(cars$dist)))
print(dist_gmean)
dist_hmean <- length(cars$dist) / sum(1/cars$dist)
print(dist_hmean)
dist_median <- median(cars$dist)
print(dist_median)


# Find the unique values of dist column
dist_unique <- unique(cars$dist)
print(dist_unique)


# Find the variance of both the columns
speed_var <- var(cars$speed)
print(speed_var)
```

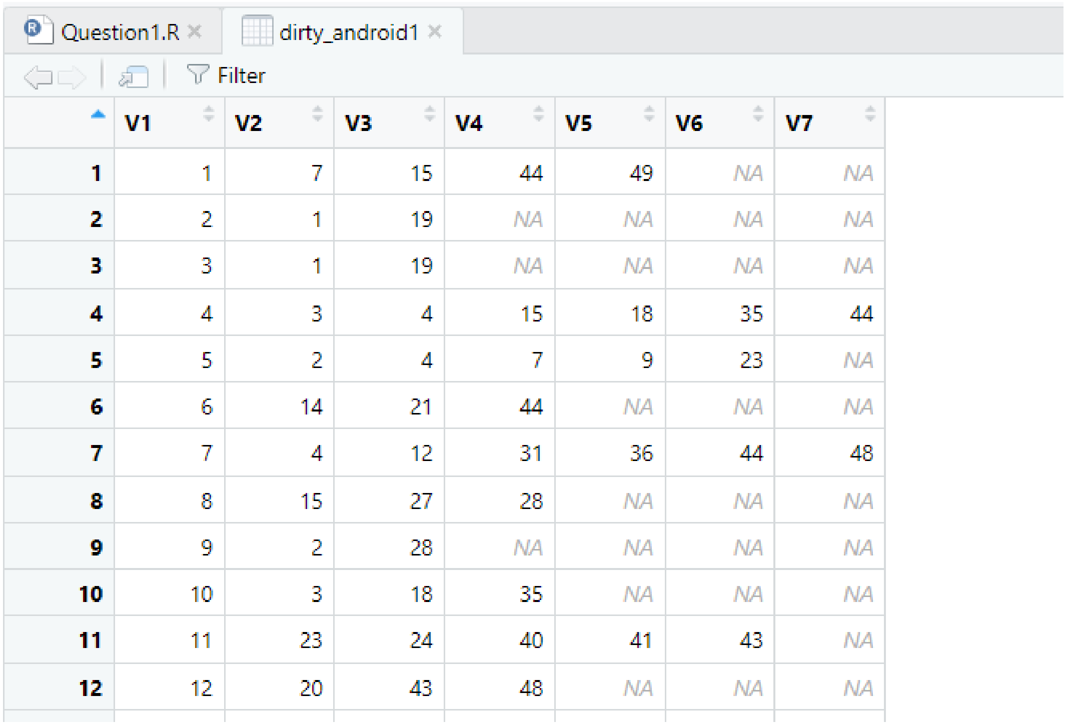
```r
dist_var <- var(cars$dist)
print(dist_var)


# Find the IQR of speed column
speed_IQR <- IQR(cars$speed)
print(speed_IQR)


# Create quartiles for dist column
dist_quartiles <- quantile(cars$dist, probs = c(0.25, 0.5, 0.75))
print(dist_quartiles)




# Question-2
# Load the dataset
dirty_android1 <- read.csv("C:/Users/DELL/Desktop/SEM-6/DM/R
Practicals/Dirty android1 data.csv", header=FALSE)
View(dirty_android1)


# Identify the count of NA's in the data frame
na_count <- sum(is.na(dirty_android1))
print(na_count)


# Calculate the number and percentage of observations that are complete
complete_observations <- sum(complete.cases(dirty_android1))
print(complete_observations)
complete_percentage <- complete_observations/nrow(dirty_android1) * 100
print(complete_percentage)


# Find the position of NA values in LCOM3 column
na_index <- which(is.na(dirty_android1$LCOM3))
print(na_index)
```

```r
# Report the mean values of each column
column_means <- colMeans(dirty_android1, na.rm = TRUE)
print(column_means)


# Replace all empty rows with mean values of the column
dirty_android1[is.na(dirty_android1)] <- lapply(dirty_android1,
function(x) ifelse(is.na(x), mean(x, na.rm = TRUE), x))
print(dirty_android1[is.na(dirty_android1)])


# Report the mean values of each column after replacing empty rows with
mean values
column_means_after_replacement <- colMeans(dirty_android1)
print(column_means_after_replacement)


# Read the dataset again in another data frame variable and omit all the
records with NA values
dirty_android1_omitted <- dirty_android1[complete.cases(dirty_android1),]
print(dirty_android1_omitted)
```

Output:



```
> source("D:/Code/College/R_Code/Question1.R")
'data.frame':	50 obs. of  2 variables:
 $ speed: num  4 4 7 7 8 9 10 10 10 11 ...
 $ dist : num  2 10 4 22 16 10 18 26 34 17 ...
[1] 15.4
[1] 14.32501
[1] 12.96153
[1] 15
[1] 42.98
[1] 34.32615
[1] 22.18214
[1] 36
 [1]   2  10   4  22  16  18  26  34  17  28  14  20  24  46  36  60  80  54  32  40  50  42
[23]  56  76  84  68  48  52  64  66  70  92  93 120  85
[1] 27.95918
[1] 664.0608
[1] 7
25% 50% 75% 
 26  36  56 
[1] 2798
[1] 92
[1] 8.778626
integer(0)
       V1        V2        V3        V4        V5        V6        V7 
479.54485  11.51378  20.71383  31.35990  36.03409  38.22768  39.53261 
```


Question1.R dirty_android1

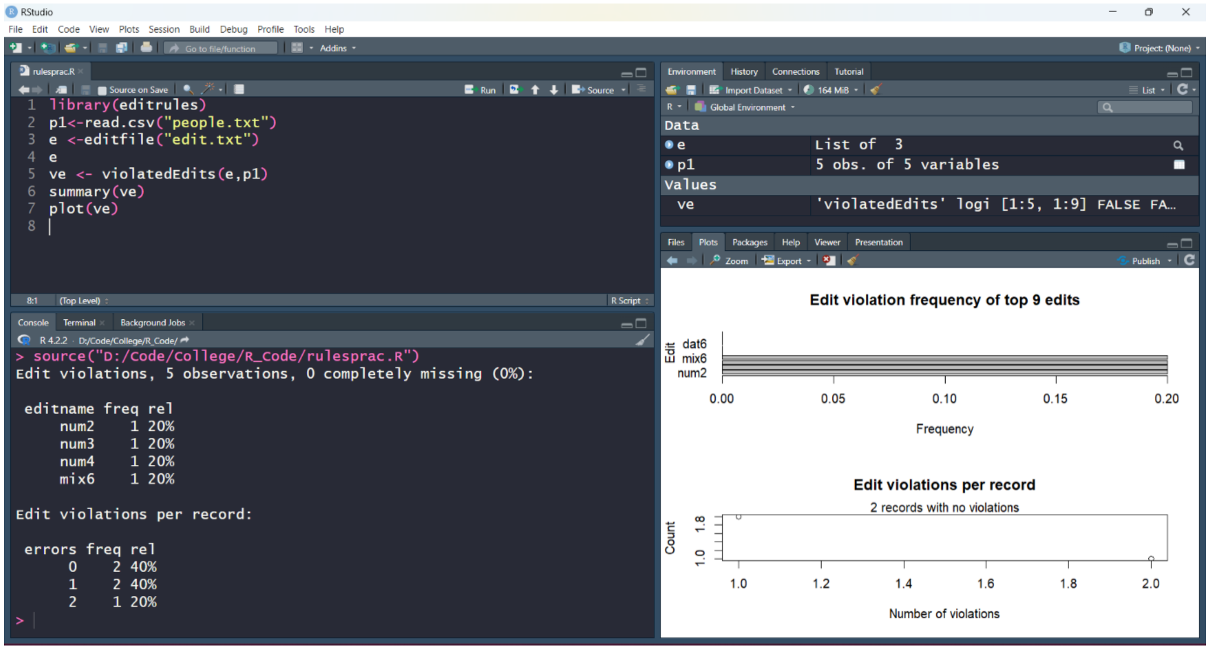
Filter

	V1	V2	V3	V4	V5	V6	V7
1	1	7	15	44	49	NA	NA
2	2	1	19	NA	NA	NA	NA
3	3	1	19	NA	NA	NA	NA
4	4	3	4	15	18	35	44
5	5	2	4	7	9	23	NA
6	6	14	21	44	NA	NA	NA
7	7	4	12	31	36	44	48
8	8	15	27	28	NA	NA	NA
9	9	2	28	NA	NA	NA	NA
10	10	3	18	35	NA	NA	NA
11	11	23	24	40	41	43	NA
12	12	20	43	48	NA	NA	NA

EditRules

```
# Q1
library(editrules)
p1<-read.csv("people.txt")
e <-editfile("edit.txt")
e
ve <- violatedEdits(e,p1)
summary(ve)
plot(ve)
```

Output:

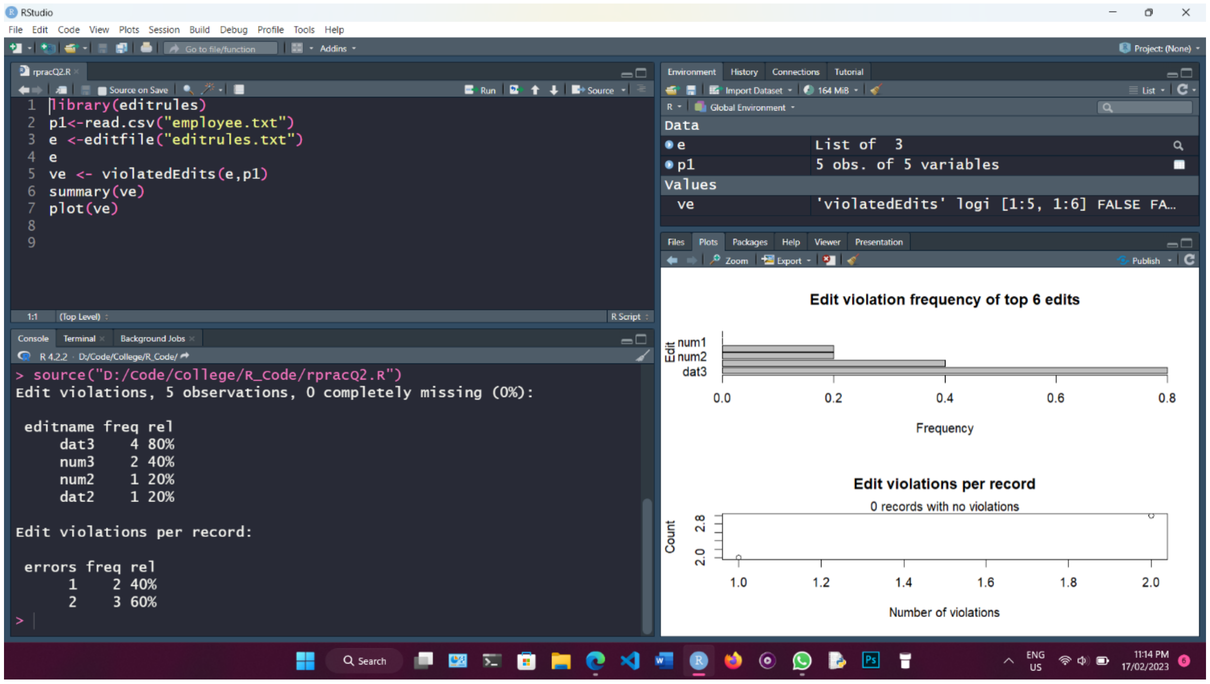


```r
library(editrules)
p1<-read.csv("people.txt")
e <-editfile("edit.txt")
e
ve <- violatedEdits(e,p1)
summary(ve)
plot(ve)
```

```
e     List of  3
p1    5 obs. of 5 variables
ve    'violatedEdits' logi [1:5, 1:9] FALSE FA…
```

```
> source("D:/Code/College/R_Code/rulesprac.R")
Edit violations, 5 observations, 0 completely missing (0%):

 editname freq rel
     num2    1 20%
     num3    1 20%
     num4    1 20%
     mix6    1 20%

Edit violations per record:

 errors freq rel
      0    2 40%
      1    2 40%
      2    1 20%
>
```

```r
# Q2
library(editrules)
p1<-read.csv("employee.txt")
e <-editfile("editrules.txt")
e
ve <- violatedEdits(e,p1)
summary(ve)
plot(ve)
```



```
library(editrules)
p1<-read.csv("employee.txt")
e <-editfile("editrules.txt")
e
ve <- violatedEdits(e,p1)
summary(ve)
plot(ve)
```

```
> source("D:/Code/College/R_Code/rpracQ2.R")
Edit violations, 5 observations, 0 completely missing (0%):

 editname freq rel
     dat3    4 80%
     num3    2 40%
     num2    1 20%
     dat2    1 20%

Edit violations per record:

 errors freq rel
      1    2 40%
      2    3 60%
>
```

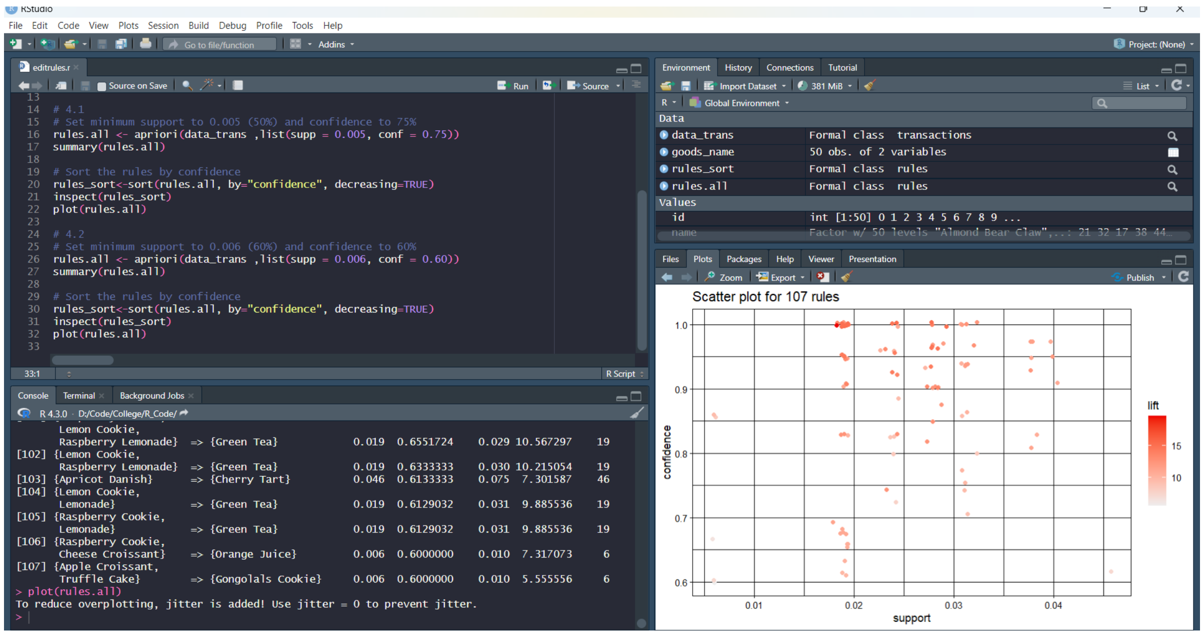
```
Data
e     List of  3
p1    5 obs. of 5 variables
Values
ve    'violatedEdits' logi [1:5, 1:6] FALSE FA…
```

Edit violation frequency of top 6 edits
Edit
num1
num2
dat3
0.0
0.2
0.4
0.6
0.8
Frequency
Edit violations per record
0 records with no violations
Count
2.0
2.8
1.0
1.2
1.4
1.6
1.8
2.0
Number of violations

Association Rule Mining

```
library(arules)
library(arulesViz)


data_trans <-
read.transactions("C:/Users/DELL/Desktop/SEM-6/DM/CompletePracticalFile/Q4
/1000-out1.csv",sep=",", cols = 1, rm.duplicates = FALSE)


id <- c(0:49)
name <- factor(c("Chocolate Cake","Lemon Cake","Casino Cake","Opera
Cake","Strawberry Cake","Truffle Cake","Chocolate Eclair","Coffee
Eclair","Vanilla Eclair","Napoleon Cake","Almond Tart","Apple Pie","Apple
Tart","Apricot Tart","Berry Tart","Blackberry Tart","Blueberry
Tart","Chocolate Tart","Cherry Tart","Lemon Tart","Pecan Tart","Ganache
Cookie","Gongolals Cookie","Raspberry Cookie","Lemon Cookie","Chocolate
Meringue","Vanilla Meringue","Marzipan Cookie","Tuile Cookie","Walnut
Cookie","Almond Croissant","Apple Croissant","Apricot Croissant","Cheese
Croissant","Chocolate Croissant","Apricot Danish","Apple Dansh","Almond
Twist","Almond Bear Claw","Blueberry Danish","Lemonade","Raspberry
Lemonade","Orange Juice","Green Tea","Bottled Water","Hot
Coffee","Chocolate Coffee","Vanilla Frappucino","Cherry Soda","Single
Espresso"))
goods_name <- data.frame(id, name)


data_trans@itemInfo$labels <-
goods_name$name[match(data_trans@itemInfo$labels,goods_name$id)]


summary(data_trans)


# 4.1
# Set minimum support to 0.005 (50%) and confidence to 75%
rules.all <- apriori(data_trans ,list(supp = 0.005, conf = 0.75))
summary(rules.all)


# Sort the rules by confidence
rules_sort<-sort(rules.all, by="confidence", decreasing=TRUE)
inspect(rules_sort)
plot(rules.all)


# 4.2
# Set minimum support to 0.006 (60%) and confidence to 60%
rules.all <- apriori(data_trans ,list(supp = 0.006, conf = 0.60))
summary(rules.all)


# Sort the rules by confidence
rules_sort<-sort(rules.all, by="confidence", decreasing=TRUE)
inspect(rules_sort)
plot(rules.all)
```

Output:



```
# 4.1
# Set minimum support to 0.005 (50%) and confidence to 75%
rules.all <- apriori(data_trans ,list(supp = 0.005, conf = 0.75))
summary(rules.all)

# Sort the rules by confidence
rules_sort<-sort(rules.all, by="confidence", decreasing=TRUE)
inspect(rules_sort)
plot(rules.all)

# 4.2
# Set minimum support to 0.006 (60%) and confidence to 60%
rules.all <- apriori(data_trans ,list(supp = 0.006, conf = 0.60))
summary(rules.all)

# Sort the rules by confidence
rules_sort<-sort(rules.all, by="confidence", decreasing=TRUE)
inspect(rules_sort)
plot(rules.all)
```

```
      Lemon Cookie,
      Raspberry Lemonade}  => {Green Tea}         0.019  0.6551724   0.029 10.567297    19
[102] {Lemon Cookie,
      Raspberry Lemonade}  => {Green Tea}         0.019  0.6333333   0.030 10.215054    19
[103] {Apricot Danish}     => {Cherry Tart}       0.046  0.6133333   0.075  7.301587    46
[104] {Lemon Cookie,
      Lemonade}            => {Green Tea}         0.019  0.6129032   0.031  9.885536    19
[105] {Raspberry Cookie,
      Lemonade}            => {Green Tea}         0.019  0.6129032   0.031  9.885536    19
[106] {Raspberry Cookie,
      Cheese Croissant}    => {Orange Juice}      0.006  0.6000000   0.010  7.317073     6
[107] {Apple Croissant,
      Truffle Cake}        => {Gongolals Cookie}  0.006  0.6000000   0.010  5.555556     6
> plot(rules.all)
To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.
>
```

```
Data
data_trans    Formal class  transactions
goods_name    50 obs. of 2 variables
rules_sort    Formal class  rules
rules.all     Formal class  rules
Values
id            int [1:50] 0 1 2 3 4 5 6 7 8 9 ...
name          Factor w/ 50 levels "Almond Bear Claw",..: 21 32 17 38 44...
```

Classification using Decision Tree

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score


# Load the Iris dataset
iris = load_iris()


# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(iris.data,
iris.target, test_size=0.2, random_state=42)


# Create a decision tree classifier
clf = DecisionTreeClassifier(random_state=42)


# Train the classifier
clf.fit(X_train, y_train)
```

```python
# Make predictions
y_pred = clf.predict(X_test)


# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

Output:

```
Accuracy: 1.0
```

Practical List Q2

Perform the following preprocessing tasks on the dirty_iris datasetii.

i) Calculate the number and percentage of observations that are complete.

ii) Replace all the special values in data with NA.

iii) Define these rules in a separate text file and read them.

(Use editfile function in R (package editrules). Use similar function in Python).

Print the resulting constraint object. – Species should be one of the following values: setosa, versicolor or virginica. – All measured numerical properties of an iris should be positive. – The petal length of an iris is at least 2 times its petal width. – The sepal length of an iris cannot exceed 30 cm. – The sepals of an iris are longer than its petals. iv)Determine how often each rule is broken (violatedEdits). Also summarize and plot the result. v) Find outliers in sepal length using boxplot and boxplot.stats

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

```
df=pd.read_csv("iris_dirty.csv")
t=df.shape[0]
df
```

	Unnamed: 0	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	1	5.1	3.5	1.4	0.2	Setosa
1	2	4.9	3.0	1.4	0.2	setosa
2	3	4.7	3.2	1.3	0.2	setosa
3	4	4.6	3.1	1.5	0.2	setosa
4	5	NaN	3.6	1.4	0.2	setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	virginica
146	147	6.3	2.5	5.0	1.9	virginica
147	148	6.5	3.0	5.2	2.0	virginica
148	149	6.2	3.4	5.4	2.3	virginica
149	150	NaN	3.0	5.1	1.8	virginica

150 rows × 6 columns

```
null_count=df.isnull().any(axis=1).sum()
print("null",null_count,"\npercentage",null_count/df.shape[0]*100)
```

```
null 19
percentage 12.666666666666668
```

```
df=df.replace('/W',np.nan)
```

```
c1=df[df['Species'].isin(['setosa', 'versicolor', 'virginica'])]
c1_f=c1.shape[0]
c1_f
```

```
145
```

```
c2=df[(df['Sepal.Length']>0)&(df['Sepal.Width']>0)&(df['Petal.Length']>0)&(df['Petal.Wid
c2_f=c2.shape[0]
c2_f
```

```
131
```

```
c3=df[(df['Petal.Length']>df['Petal.Width'])]
c3_f=c3.shape[0]
c3_f
```

```
150
```

```
c4=df[(df['Sepal.Length']<30)]
c4_f=c4.shape[0]
c4_f
```

```
142
```

```
c5=df[(df['Sepal.Length']>df['Petal.Length'])]
c5_f=c5.shape[0]
c5_f
```

```
142
```

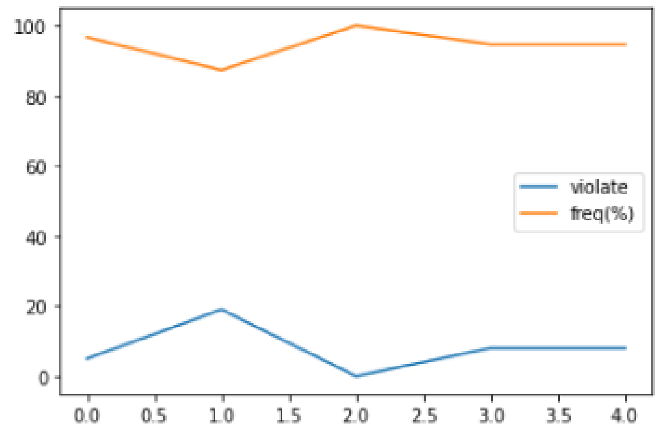
```
dic={'violate':[t-c1_f,t-c2_f,t-c3_f,t-c4_f,t-c5_f],"freq(%)":[(c1_f/t)*100,(c2_f/t)*100,
```

```
summary=pd.DataFrame(dic)
summary
```

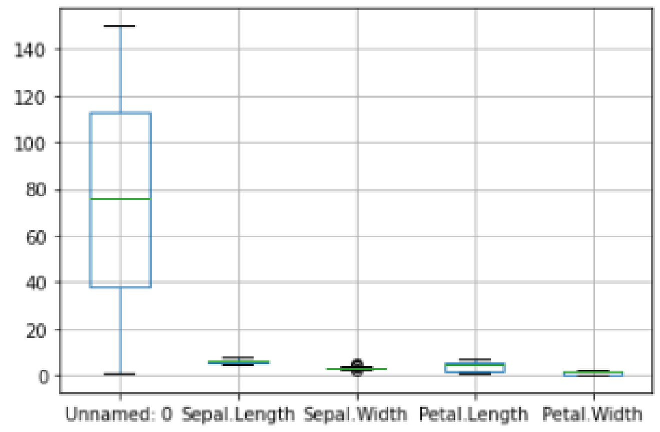
	violate	freq(%)
0	5	96.666667
1	19	87.333333
2	0	100.000000
3	8	94.666667
4	8	94.666667

```
summary.plot()
```

```
<AxesSubplot:>
```



```
x=df.boxplot()
```



Practical List Q3

Load the data from wine dataset. Check whether all attributes are standardized or not (mean is 0 and standard deviation is 1). If not, standardize the attributes.

```python
import pandas as pd
```

```python
df=pd.read_csv('wine.csv',header=None)
df
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
173	3	13.71	5.65	2.45	20.5	95	1.68	0.61	0.52	1.06	7.70	0.64	1.74	740
174	3	13.40	3.91	2.48	23.0	102	1.80	0.75	0.43	1.41	7.30	0.70	1.56	750
175	3	13.27	4.28	2.26	20.0	120	1.59	0.69	0.43	1.35	10.20	0.59	1.56	835
176	3	13.17	2.59	2.37	20.0	120	1.65	0.68	0.53	1.46	9.30	0.60	1.62	840
177	3	14.13	4.10	2.74	24.5	96	2.05	0.76	0.56	1.35	9.20	0.61	1.60	560

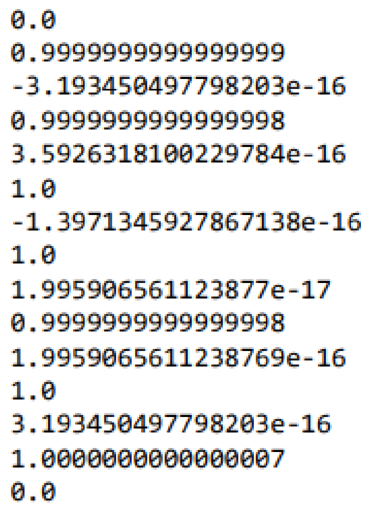
178 rows × 14 columns

:
```
for i in range(df.shape[1]):
    print(df[i].std(),df[i].mean(),sep='\n')
```

```
0.7750349899850565
1.9382022471910112
0.8118265380058577
13.00061797752809
1.1171460976144627
2.3363483146067416
0.2743440090608148
2.3665168539325845
3.3395637671735052
19.49494382022472
14.282483515295668
99.74157303370787
0.6258510488339891
2.295112359550562
0.9988586850169465
2.0292696629213487
0.12445334029667939
0.3618539325842696
0.5723588626747611
1.5908988764044945
2.318285871822413
5.058089882022472
0.22857156582982338
0.9574494382022471
0.7099904287650505
2.6116853932584267
314.9074742768489
746.8932584269663
```

:
```
for i in range(df.shape[1]):

    m=df[i].mean()
    std=df[i].std()

    df[i]=(df[i]-m)/std
    print(df[i].std(),df[i].mean(),sep='\n')
```

```
0.9999999999999981
0.0
0.9999999999999997
-8.781988868945058e-16
1.0
0.0
1.0
-7.996100660502532e-16
1.0000000000000002
-7.983626244495507e-17
0.9999999999999998
1.995906561123877e-17
1.0
```



```
0.0
0.9999999999999999
-3.193450497798203e-16
0.9999999999999998
3.5926318100229784e-16
1.0
-1.3971345927867138e-16
1.0
1.995906561123877e-17
0.9999999999999998
1.9959065611238769e-16
1.0
3.193450497798203e-16
1.0000000000000007
0.0
```

Practical List Q5

Use Naive bayes, K-nearest, and Decision tree classification algorithms and build classifiers. Divide the data set into training and test set. Compare the accuracy of the different classifiers under the following situations: 5.1 a) Training set = 75% Test set = 25% b) Training set = 66.6% (2/3rd of total), Test set = 33.3% 5.2 Training set is chosen by i) hold out method ii) Random subsampling iii) CrossValidation. Compare the accuracy of the classifiers obtained. 5.3 Data is scaled to standard format.

```
import sklearn
import sklearn.naive_bayes
import sklearn.neighbors
import sklearn.tree
import sklearn.model_selection
import sklearn.datasets
import sklearn.metrics
```

```
data=sklearn.datasets.load_iris()
X_train, X_test, y_train, y_test = sklearn.model_selection.train_test_split(data.da
```

```
model1=sklearn.tree.DecisionTreeClassifier(random_state=0)
```

```
model2=sklearn.neighbors.KNeighborsClassifier(n_neighbors=3)
```

```
model3=sklearn.naive_bayes.GaussianNB()
```

```
model1.fit(X_train,y_train)
model2.fit(X_train,y_train)
model3.fit(X_train,y_train)
```

```
GaussianNB()
```

```
y_pre_1=model1.predict(X_test)
y_pre_2=model2.predict(X_test)
y_pre_3=model3.predict(X_test)
```

```
sklearn.metrics.accuracy_score(y_test, y_pre_1)
```

```
0.96
```

```
sklearn.metrics.accuracy_score(y_test, y_pre_2)
```

```
0.98
```

```
sklearn.metrics.accuracy_score(y_test, y_pre_3)
```

```
0.96
```

```
sklearn.metrics.confusion_matrix(y_test, y_pre_3)
```

```
array([[19,  0,  0],
       [ 0, 14,  1],
       [ 0,  1, 15]], dtype=int64)
```

```
print(sklearn.model_selection.cross_val_score(model1, data.data, data.target, cv=3)
```

```
[0.98 0.94 0.98]
```

```
print(sklearn.model_selection.cross_val_score(model2, data.data, data.target, cv=3)
```

```
[0.98 0.96 0.98]
```

```
print(sklearn.model_selection.cross_val_score(model3, data.data, data.target, cv=3)
```

```
[0.92 0.94 0.96]
```

Practical List Q6

Use Simple Kmeans, DBScan, Hierachical clustering algorithms for clustering. Compare the performance of clusters by changing the parameters involved in the algorithms.

```
import pandas as pd
import numpy as np
import sklearn.cluster
import scipy.cluster.hierarchy
import matplotlib.pyplot as plt
```

```
df=pd.read_csv('iris.data',header=None)
```

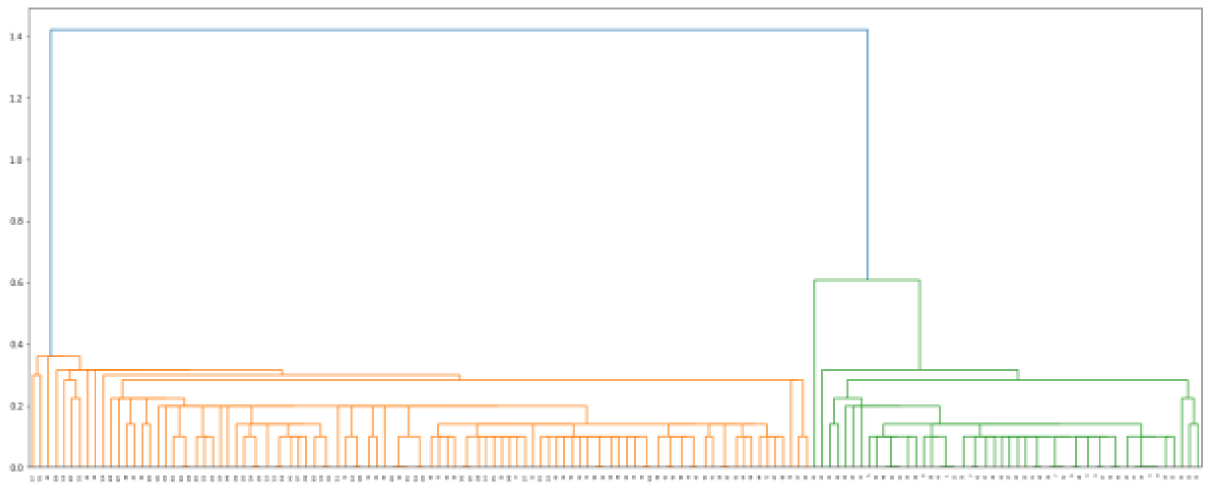
```
clustering = sklearn.cluster.DBSCAN(eps=0.1, min_samples=2).fit(df[[0,1]])
clustering.labels_
```

```
array([ 0,  1,  2, -1, -1,  3, -1,  4, -1,  5, -1,  6,  7, -1, -1, -1,  3,
        0, -1,  8,  9,  8, -1, 10,  6,  1,  4, -1, -1,  2, -1,  9, -1, -1,
        5, 10, -1,  5, -1,  4,  0, -1, -1,  0,  8,  7,  8, -1, -1, 10, 11,
       12, 13, -1, 14, 15, 16, -1, -1, -1, -1, 17, 18, 19, -1, 20, 21, 15,
       -1, 22, -1, 23, 24, 23, -1, 25, -1, 26, 19, 27, 28, 28, 15, -1, -1,
       -1, 20, -1, 21, 22, -1, 17, 27, -1, 29, -1, -1, 30, -1, 15, 16, 15,
       -1, 30, 25, -1, -1, -1, -1, -1, 12, 14, 26, -1, 15, 12, 25, -1, -1,
       18, 11, 29, -1, 31, 32, -1, 31, 17, 14, -1, -1, -1, 14, 31, -1, -1,
       33, -1, 17, 13, 20, 13, 15, -1, 32, 26, 24, 25, 33, 17],
      dtype=int64)
```

```
kmeans = sklearn.cluster.KMeans(n_clusters=2, random_state=0).fit(df[[0,1]])
kmeans.labels_
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1,
       0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0])
```

```
Z = scipy.cluster.hierarchy.linkage(df[[1,2]], 'single')
fig = plt.figure(figsize=(25, 10))
dn = scipy.cluster.hierarchy.dendrogram(Z)
plt.show()
```


1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0