2. Explain how to merge two data frames with example.

(only example 3)

Example 1: Combine Dataframe Vertically Using rbind() in R

If we want to combine two data frames vertically, the column name of the two data frames must be the same. For example,

```
# create a data frame
dataframe1 <- data.frame (
  Name = c("Juan", "Alcaraz"),
  Age = c(22, 15)
)

# create another data frame
dataframe2 <- data.frame (
  Name = c("Yiruma", "Bach", "Ludovico"),
  Age = c(46, 89, 72)
)

# combine two data frames vertically
updated <- rbind(dataframe1, dataframe2)
print(updated)
```

Output

```
      Name    Age
1       Juan    22
2    Alcaraz    15
3     Yiruma    46
4       Bach    89
5     Ludovico   72
```

In the above example,we have created a dataframe named `dataframe1`, with two elements for both the columns and `dataframe2`, with three elements for both the columns. And we have combined them together vertically using `rbind()`.
Since we want to combine two data frames vertically, we have provided the same column name `Name` and `Age` for both the data frames.

Example 2: Combine Dataframe Horizontally Using cbind() in R

The `cbind()` function combines two or more data frames horizontally. For example,

```
# create a data frame
dataframe1 <- data.frame (
  Name = c("Juan", "Alcaraz"),
  Age = c(22, 15)
)

# create another data frame
dataframe2 <- data.frame (
  Hobby = c("Tennis", "Piano")
)

# combine two data frames horizontally
updated <- cbind(dataframe1, dataframe2)
print(updated)
```

Output

```
     Name  Age  Hobby
1     Juan  22 Tennis
2 Alcaraz   15  Piano
```

Here, we have used `cbind()` to combine two dataframes: `dataframe1` and `dataframe2` horizontally.

Example 3: Combine Dataframe Vertically Using merge() in R

If we want to combine two data frames vertically, the column name of the two data frames must be the same. For example,

```
dataframe1 = data.frame(Id = c(101, 102, 103, 104, 105),
  Course = c("DSA", "Graphics", "Calculus-1", "Physics", "Network")
)

dataframe2 = data.frame(Id = c(101, 103, 105, 108, 110),
  Address = c("Nepal", "USA","Sweden", "Japan", "Norway")
)

# merge two dataframes based on Id
merge(dataframe1,dataframe2, by = "Id")
```

Output

```
  Id     Course Address
1 101       DSA   Nepal
2 103 Calculus-1     USA
```

```
3 105     Network   Sweden
```

In the above example, we have created two dataframes named: `dataframe1` and `dataframe2`. We have then used the `merge()` function to merge two dataframes.

Here, two dataframes whose `Id` value is the same are merged together.

2.Explain about sorting with suitable examples?

In R, there are several built-in functions for sorting vectors and data frames. The most commonly used functions are sort() and order().

The sort() function is used to sort a vector in ascending or descending order. By default, it sorts the vector in ascending order. Here's an example of sorting a vector of integers in R using sort():

Output:

```
[1] 1 2 3 4 5 8
```

To sort the vector in descending order, we can use the argument decreasing = TRUE as follows:

```
# Sort the vector in descending order

sorted_x_desc <- sort(x, decreasing = TRUE)

# Print the sorted vector

print(sorted_x_desc)
```

Output:

```
[1] 8 5 4 3 2 1
```

The order() function is used to return a permutation that would sort a given vector or data frame. It returns an integer index that can be used to reorder the original object. Here's an example of using order() to sort a data frame by one of its columns:

```
# Create a data frame with two columns: name and age

df <- data.frame(name = c("Alice", "Bob", "Charlie"),

           age = c(25, 30, 20))

# Order the data frame by age column in ascending order

ordered_df <- df[order(df$age), ]

# Print the ordered data frame

print(ordered_df)
```

Output:

```
   name age
```

3 Charlie 20

1 Alice 25

2 Bob 30

In this example, we use order(df$age) to get an index that sorts the rows of df by their age column. We then use this index to reorder the rows of df using [ ].

There are several other sorting algorithms available in R such as bubble sort, insertion sort, selection sort, merge sort and quicksort [1][2][3].

4 Explain about math function with two suitable. examples

S. No	Function	Description	Example

1.	abs(x)	It returns the absolute value of input x.	`x<- -4`
			`print(abs(x))`
			Output
			`[1] 4`

2.	sqrt(x)	It returns the square root of input x.	`x<- 4`
			`print(sqrt(x))`
			Output
			`[1] 2`

3.	ceiling(x)	It returns the smallest integer which is larger than or equal to x.	`x<- 4.5`
			`print(ceiling(x))`
			Output
			`[1] 5`

4.	floor(x)	It returns the largest integer, which is smaller than or equal to x.	`x<- 2.5`
			`print(floor(x))`
			Output
			`[1] 2`

5.	trunc(x)	It returns the truncate value of input x.	`x<-`
			`c(1.2,2.5,8.1)`
			`print(trunc(x))`
			Output
			`[1] 1 2 8`

6.	round(x, digits=n)	It returns round value of input x.	`x<- -4`
			`print(abs(x))`
			Output
			`4`

7.	cos(x), sin(x), tan(x)	It returns cos(x), sin(x) value of input x.	`x<- 4`
			`print(cos(x))`
			`print(sin(x))`
			`print(tan(x))`
			Output
			`[1] -06536436`

			`[2] -0.7568025`
			`[3] 1.157821`

8.	log(x)	It returns natural logarithm of input x.	`x<- 4`
			`print(log(x))`
			Output
			`[1] 1.386294`

9.	log10(x)	It returns common logarithm of input x.	`x<- 4`
			`print(log10(x))`
			Output
			`[1] 0.60206`

10.	exp(x)	It returns exponent.	`x<- 4`
			`print(exp(x))`
			Output
			`[1] 54.59815`

6. Explain about pie chart & bar chart with examples.

A pie-chart is a representation of values as slices of a circle with different colors. The slices are labeled and the numbers corresponding to each slice is also represented in the chart.

In R the pie chart is created using the **pie()** function which takes positive numbers as a vector input. The additional parameters are used to control labels, color, title etc.

Syntax

The basic syntax for creating a pie-chart using the R is –

```
pie(x, labels, radius, main, col, clockwise)
```

Following is the description of the parameters used –

- **x** is a vector containing the numeric values used in the pie chart.
- **labels** is used to give description to the slices.
- **radius** indicates the radius of the circle of the pie chart.(value between −1 and +1).
- **main** indicates the title of the chart.
- **col** indicates the color palette.
- **clockwise** is a logical value indicating if the slices are drawn clockwise or anti clockwise.

Example

A very simple pie-chart is created using just the input vector and labels. The below script will create and save the pie chart in the current R working directory.

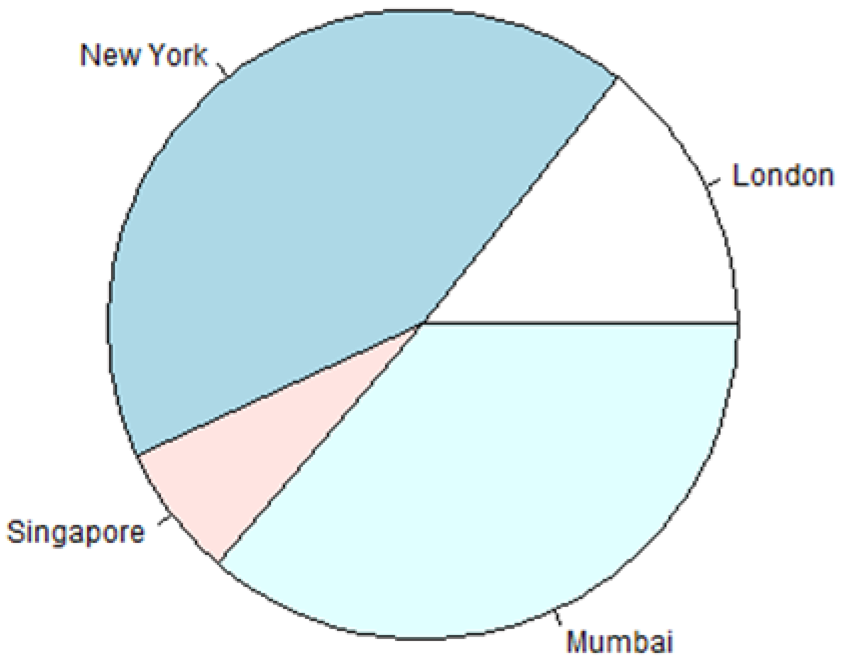
```
# Create data for the graph.
x <- c(21, 62, 10, 53)
labels <- c("London", "New York", "Singapore", "Mumbai")

# Give the chart file a name.
png(file = "city.png")

# Plot the chart.
pie(x,labels)
```

```
# Save the file.
dev.off()
```

When we execute the above code, it produces the following result –



Bar Charts

A bar chart represents data in rectangular bars with length of the bar proportional to the value of the variable. R uses the function **barplot()** to create bar charts. R can draw both vertical and Horizontal bars in the bar chart. In bar chart each of the bars can be given different colors.

Syntax

The basic syntax to create a bar-chart in R is –

```
barplot(H,xlab,ylab,main, names.arg,col)
```

Following is the description of the parameters used –

- **H** is a vector or matrix containing numeric values used in bar chart.
- **xlab** is the label for x axis.
- **ylab** is the label for y axis.
- **main** is the title of the bar chart.
- **names.arg** is a vector of names appearing under each bar.
- **col** is used to give colors to the bars in the graph.

Example

A simple bar chart is created using just the input vector and the name of each bar.

The below script will create and save the bar chart in the current R working directory.

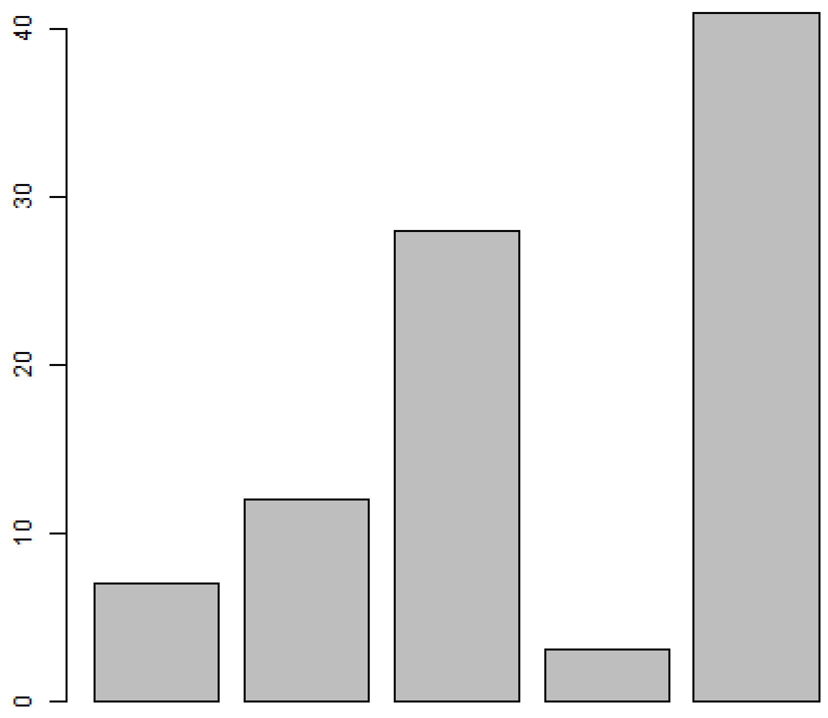
```
# Create the data for the chart
H <- c(7,12,28,3,41)
```

```
# Give the chart file a name
png(file = "barchart.png")

# Plot the bar chart
barplot(H)

# Save the file
dev.off()
```

When we execute above code, it produces following result –



7 Explain about applying functions to data frame.

In R, we can apply functions to data frames using various functions such as apply(), lapply(), sapply(), and more. These functions allow us to apply a function to each row or column of a data frame or to the entire data frame. Here are some examples:

1. Using apply(): The apply() function is used to apply a function to either the rows or columns of a matrix or data frame. Here's an example of using apply() to calculate the sum of each row in a data frame:

# Create a data frame with three columns: x, y, and z

df <- data.frame(x = c(1, 2, 3), y = c(4, 5, 6), z = c(7, 8, 9))

# Apply the sum function to each row of the data frame

row_sums <- apply(df, 1, sum)

```
# Print the row sums

print(row_sums)
```

Output:

```
[1] 12 15 18
```

In this example, we use apply(df, 1, sum) to calculate the sum of each row in the data frame.

2. Using lapply() and sapply(): The lapply() and sapply() functions are used to apply a function to each element of a list or vector. Here's an example of using lapply() and sapply() to calculate the square root of each element in a column of a data frame:

```
# Create a data frame with two columns: x and y

df <- data.frame(x = c(1, 4, 9), y = c(16, 25, 36))

# Use lapply() to calculate the square root of column y

sqrt_y_l <- lapply(df$y, sqrt)

# Use sapply() to calculate the square root of column y

sqrt_y_s <- sapply(df$y, sqrt)

# Print both results

print(sqrt_y_l)

print(sqrt_y_s)
```

Output:

```
[[1]]

[1] 4

[[2]]

[1] 5

[[3]]

[1] 6

[1] 4 5 6
```

In this example, we use lapply(df$y,sqrt) and sapply(df$y,sqrt)to calculate the square root for each element in column y.

These are just some examples of how we can apply functions on different parts of a dataframe in R

9 Explain about control statements in R Programming?

Control statements are expressions used to control the execution and flow of the program based on the conditions provided in the statements. These structures are used

to make a decision after assessing the variable. In this article, we'll discuss all the control statements with the examples.
In R programming, there are 8 types of control statements as follows:

- if condition
- if-else condition
- for loop
- nested loops
- while loop
- repeat and break statement
- return statement
- next statement

if condition

This control structure checks the expression provided in parenthesis is true or not. If true, the execution of the statements in braces {} continues.

Syntax:

```
if(expression){

    statements

    ....

    ....

}
```

Example:

```
x <- 100



if(x > 10){

print(paste(x, "is greater than 10"))

}
```

Output:

```
[1] "100 is greater than 10"
```

if-else condition

It is similar to **if** condition but when the test expression in if condition fails, then statements in **else** condition are executed.

Syntax:

```
if(expression){

    statements

    ....

    ....

}

else{

    statements

    ....

    ....
}
```

Example:

```
x <- 5


# Check value is less than or greater than 10

if(x > 10){

  print(paste(x, "is greater than 10"))

}else{

  print(paste(x, "is less than 10"))

}
```

Output:

```
[1] "5 is less than 10"
```

for loop

It is a type of loop or sequence of statements executed repeatedly until exit condition is reached.

Syntax:

```
for(value in vector){

    statements

    ....

    ....

}
```

Example:

```
x <- letters[4:10]


for(i in x){

  print(i)

}
```

Output:

```
[1] "d"

[1] "e"

[1] "f"

[1] "g"

[1] "h"

[1] "i"

[1] "j"
```

Nested loops

Nested loops are similar to simple loops. Nested means loops inside loop. Moreover, nested loops are used to manipulate the matrix.

Example:

```
# Defining matrix

m <- matrix(2:15, 2)


for (r in seq(nrow(m))) {

  for (c in seq(ncol(m))) {

    print(m[r, c])

  }

}
```

Output:

```
[1] 2

[1] 4

[1] 6

[1] 8

[1] 10

[1] 12

[1] 14

[1] 3

[1] 5

[1] 7

[1] 9

[1] 11

[1] 13

[1] 15
```

while loop

while loop is another kind of loop iterated until a condition is satisfied. The testing expression is checked first before executing the body of loop.

Syntax:

```
while(expression){

     statement

     ....

     ....

}
```

Example:

```
x = 1


# Print 1 to 5

while(x <= 5){

  print(x)

  x = x + 1

}
```

Output:

```
[1] 1

[1] 2

[1] 3

[1] 4

[1] 5
```

repeat loop and break statement

repeat is a loop which can be iterated many number of times but there is no exit condition to come out from the loop. So, break statement is used to exit from the loop. **break** statement can be used in any type of loop to exit from the loop.

Syntax:

```
repeat {

   statements

   ....

   ....

   if(expression) {

      break

   }

}
```

Example:

```
x = 1


# Print 1 to 5

repeat{

  print(x)

  x = x + 1

  if(x > 5){
    break

   }

}
```

Output:

```
[1] 1

[1] 2

[1] 3

[1] 4

[1] 5
```

return statement

return statement is used to return the result of an executed function and returns control to the calling function.

Syntax:

```
return(expression)
```

Example:

```
# Checks value is either positive, negative or zero

func <- function(x){

  if(x > 0){

    return("Positive")

  }else if(x < 0){

    return("Negative")

  }else{

    return("Zero")

  }

}


func(1)

func(0)

func(-1)
```

Output:

```
[1] "Positive"

[1] "Zero"

[1] "Negative"
```

next statement

next statement is used to skip the current iteration without executing the further statements and continues the next iteration cycle without terminating the loop.

Example:

```
# Defining vector

x <- 1:10


# Print even numbers

for(i in x){

  if(i%%2 != 0){

    next #Jumps to next loop

  }

  print(i)

}
```

Output:

```
[1] 2

[1] 4

[1] 6

[1] 8

[1] 10
```

10 a)Differentiate print() and cat().

In R, both print() and cat() are used to display output to the console or a file. However, there are some differences between these two functions.

print() function:

print() is a generic function in R that can be used to display a wide range of objects, including vectors, matrices, arrays, data frames, and lists.

The print() function automatically adds a newline character at the end of the output, which means that each call to print() generates a new line of output.

print() returns its input object as its output.

print() can be used without any arguments to print the last result stored in the R console.

Example:

x <- c(1, 2, 3)

print(x)   # Output: [1] 1 2 3

cat() function:

cat() is a function that concatenates and prints its arguments.

It is useful when you want to concatenate strings or other objects together and print them as a single output.

cat() does not add a newline character automatically, which means that you need to include it explicitly in the arguments if you want to start a new line of output.

cat() does not return any value, but it directly prints the concatenated strings or objects.

Example:

x <- c("hello", "world")

cat(x, "\n")   # Output: hello world

In summary, while both print() and cat() functions are used to display output, print() is used for general output display and adds a newline character automatically, while cat() is used for concatenating and printing objects without adding a newline character.

b) Write scan() and readline() functions with suitable example.

In R, scan() and readline() are functions used to read input from the console or a file.

1. **scan**(): The scan() function is used to read data into a vector or list from a file or the console. Here's an example of using scan() to read in a vector of numbers from the console:

```
# Read in a vector of numbers from the console

x <- scan()

# Print the vector

print(x)
```

When you run this code, R will prompt you to enter some numbers. Once you've entered them, press Enter and then Ctrl+D (or Ctrl+Z on Windows) to indicate that you're done entering input. R will then store your input in the variable x and print it out.

2. **readline**(): The readline() function is used to read in a single line of text from the console. Here's an example of using readline() to prompt the user for their name:

```
# Prompt the user for their name

name <- readline(prompt = "What is your name? ")

# Print a greeting using their name

cat("Hello,", name, "!\n")
```

When you run this code, R will prompt you with the message "What is your name?" You can then enter your name and press Enter. R will store your input in the variable name and use it to print out a greeting.

Overall, both functions are useful for reading input from users or files in R