R PROGRAMMING

1.Operations on vectors

a. Create vectors A and B with multiple element
b. Apply arithmetic operators +, -, *, /, and ^ on vectors A and B
c. Apply rep(), paste() and cumprod() functions on A and analyse the results.

Code for above program :

```
# Create vectors A and B with multiple elements

A <- c(1, 2, 3, 4, 5)

B <- c(6, 7, 8, 9, 10)

# Arithmetic operations on vectors A and B

sum_A_B <- A + B  # Addition

diff_A_B <- A - B  # Subtraction

prod_A_B <- A * B  # Element-wise multiplication

div_A_B <- A / B  # Element-wise division

pow_A_B <- A ^ B  # Element-wise exponentiation

# Print the results

cat("Sum of A and B:", sum_A_B, "\n")

cat("Difference of A and B:", diff_A_B, "\n")

cat("Product of A and B:", prod_A_B, "\n")

cat("Division of A and B:", div_A_B, "\n")

cat("Exponentiation of A and B:", pow_A_B, "\n")

# Apply rep(), paste(), and cumprod() functions on vector A

rep_A <- rep(A, times = 2)  # Repeat A twice

paste_A <- paste("A:", A, sep = " ")  # Concatenate "A:" with each element of A

cumprod_A <- cumprod(A)  # Cumulative product of A

# Print the results

cat("Repeated A:", rep_A, "\n")

cat("Concatenated A:", paste_A, "\n")
```

```
cat("Cumulative product of A:", cumprod_A, "\n")
```

Algorithm for the code:

1. Start
2. Create an empty vector A
3. Create an empty vector B
4. Add elements to vector A
5. Add elements to vector B
6. Perform arithmetic operations on vectors A and B:
   a. Add vector A and vector B, store the result in sum_A_B
   b. Subtract vector B from vector A, store the result in diff_A_B
   c. Multiply vector A and vector B element-wise, store the result in prod_A_B
   d. Divide vector A by vector B element-wise, store the result in div_A_B
   e. Raise vector A to the power of vector B element-wise, store the result in pow_A_B
7. Print the results of the arithmetic operations
8. Apply functions on vector A:
   a. Repeat vector A twice using the rep() function, store the result in rep_A
   b. Concatenate the string "A:" with each element of vector A using the paste() function, store the result in paste_A
   c. Calculate the cumulative product of vector A using the cumprod() function, store the result in cumprod_A
9. Print the results of the functions applied on vector A
10. End

Methods used in the above program:

Vector Creation: Two empty vectors, A and B, are created using the c() function. Elements can be added to these vectors by assigning values using the assignment operator <-.

Arithmetic Operations:

Various arithmetic operations are performed on vectors A and B:

Addition (+): The + operator is used to add corresponding elements of vectors A and B, resulting in a new vector representing the sum.

Subtraction (-): The - operator subtracts corresponding elements of vector B from vector A, generating a new vector representing the difference.

Multiplication (*): The * operator performs element-wise multiplication between vectors A and B, producing a new vector with the products.

Division (/): The / operator divides corresponding elements of vector A by vector B, resulting in a new vector representing the division.

Exponentiation (^): The ^ operator raises vector A to the power of vector B element-wise, generating a new vector with the exponentiated values.

Functions:

rep(): The rep() function repeats the elements of vector A a specified number of times, generating a new vector. It takes two arguments: the vector to be repeated and the number of repetitions.

paste(): The paste() function concatenates multiple strings or expressions into a single string. In the code, it is used to concatenate the string "A:" with each element of vector A. The sep argument specifies the separator between elements.

cumprod(): The cumprod() function calculates the cumulative product of a vector. It returns a vector where each element represents the product of all preceding elements in the original vector.

Printing Results: The cat() function is used to print the results of the arithmetic operations and functions applied on vector A. It takes one or more arguments and prints them to the console.

These methods enable vector manipulation, mathematical calculations, and analysis in R programming, providing flexibility and efficiency in data processing.

Output :

```
Sum of A and B: 7 9 11 13 15
```

```
Difference of A and B: -5 -5 -5 -5 -5
Product of A and B: 6 14 24 36 50
Division of A and B: 0.1666667 0.2857143 0.375 0.4444444 0.5
Exponentiation of A and B: 1 128 6561 262144 9765625
Repeated A: 1 2 3 4 5 1 2 3 4 5
Concatenated A: A: 1 A: 2 A: 3 A: 4 A: 5
Cumulative product of A: 1 2 6 24 120

[Execution complete with exit code 0]
```

2. Operations on data frames

1.Create the following data frame, later invert gender for all individuals.

Name	Age	Height	Weight	Gender
Ram	30	177	57	M
Alwin	35	164	48	F
Billy	22	155	45	M
Amera	16	180	60	F
Olive	42	124	52	F
Dora	59	150	55	F

2. Add this data frame column-wise to the previous one e (make sure you import the variable Working as character and not factor). How many rows and columns does the new data frame have? What class of data is in each column?

Name	Working
Ram	Y
Alwin	N
Billy	Y
Amera	N
Olive	Y
Dora	N

1. Code:
```
Name <- c("Ram", "Alwin", "Billy", "Amera", "Olive", "Dora")
Age <- c(30, 35, 22, 16, 42, 59)
Height <- c(177, 164, 155, 180, 124, 150)
Weight <- c(57, 48, 45, 60, 52, 55)
Gender <- as.factor(c("M", "F", "M", "F", "F", "F"))
df <- data.frame(Name, Age, Height, Weight, Gender)
# Invert gender for all individuals
levels(df$Gender) <- c("M", "F")
df$Gender <- as.factor(df$Gender)
df
```

Output:

Name	Age	Height	Weight	Gender
Ram	30	177	57	F
Alwin	35	164	48	M
Billy	22	155	45	F
Amera	16	180	60	M
Olive	42	124	52	M
Dora	59	150	55	M

```r
Name <- c("Ram", "Alwin", "Billy", "Amera", "Olive", "Dora")
Age <- c(30, 35, 22, 16, 42, 59)
Height <- c(177, 164, 155, 180, 124, 150)
Weight <- c(57, 48, 45, 60, 52, 55)
Gender <- as.factor(c("M", "F", "M", "F", "F", "F"))
df <- data.frame(Name, Age, Height, Weight, Gender)
# Invert gender for all individuals
levels(df$Gender) <- c("M", "F")
df$Gender <- as.factor(df$Gender)
df
```

```
Rscript /tmp/08R378qERm.r
Name    Age Height Weight Gender
1 Ram  30    177     57      F
2 Alwin  35    164     48      M
3 Billy  22    155     45      F
4 Amera  16    180     60      M
5 Olive  42    124     52      M
6  Dora  59    150     55      M
```

Algorithm:

1. Define the variables "Name", "Age", "Height", "Weight", and "Gender" as vectors containing the corresponding data for each individual.
2. Convert the "Gender" variable to a factor using the as.factor() function.
3. Combine the variables into a data frame using the data.frame() function.
4. Invert the levels of the "Gender" variable using the levels() function and reassign the levels to "M" and "F".
5. Assign the inverted "Gender" variable back to the data frame.
6. Print the modified data frame using the print() function.

Code:

```
Name <- c("Ram", "Alwin", "Billy", "Amera", "Olive", "Dora")
Age <- c(30, 35, 22, 16, 42, 59)
Height <- c(177, 164, 155, 180, 124, 150)
Weight <- c(57, 48, 45, 60, 52, 55)
Gender <- as.factor(c("M", "F", "M", "F", "F", "F"))
df <- data.frame(Name, Age, Height, Weight, Gender)
# Invert gender for all individuals
levels(df$Gender) <- c("M", "F")
df$Gender <- as.factor(df$Gender)
df
# Create the new data frame
Working <- c("Y", "N", "Y", "N", "Y", "N")
df2 <- data.frame(Working, stringsAsFactors = FALSE)
# Add the new data frame to the previous one
df <- cbind(df, df2)
# Print the modified data frame
df
```

Output:

Name	Age	Height	Weight	Gender
Ram	30	177	57	F
Alwin	35	164	48	M
Billy	22	155	45	F
Amera	16	180	60	M
Olive	42	124	52	M
Dora	59	150	55	M

Name	Age	Height	Weight	Gender	Working
Ram	30	177	57	F	Y
Alwin	35	164	48	M	N
Billy	22	155	45	F	Y
Amera	16	180	60	M	N
Olive	42	124	52	M	Y

Dora 59 150 55 M N

```
Name <- c("Ram", "Alwin", "Billy", "Amera", "Olive", "Dora")
Age <- c(30, 35, 22, 16, 42, 59)
Height <- c(177, 164, 155, 180, 124, 150)
Weight <- c(57, 48, 45, 60, 52, 55)
Gender <- as.factor(c("M", "F", "M", "F", "F", "F"))
df <- data.frame(Name, Age, Height, Weight, Gender)
# Invert gender for all individuals
levels(df$Gender) <- c("M", "F")
df$Gender <- as.factor(df$Gender)
df
# Create the new data frame
Working <- c("Y", "N", "Y", "N", "Y", "N")
df2 <- data.frame(Working, stringsAsFactors = FALSE)
# Add the new data frame to the previous one
df <- cbind(df, df2)
# Print the modified data frame
df
```

```
Rscript /tmp/O8R378qERm.r
Name Age Height Weight Gender
1 Ram  30    177     57      F
2 Alwin  35    164     48      M
3 Billy  22    155     45      F
4 Amera  16    180     60      M
5 Olive  42    124     52      M
6  Dora  59    150     55      M
Name Age Height Weight Gender Working
1 Ram  30    177     57      F       Y
2 Alwin  35    164     48      M       N
3 Billy  22    155     45      F       Y
4 Amera  16    180     60      M       N
5 Olive  42    124     52      M       Y
6  Dora  59    150     55      M       N
```

The new data frame has 6 rows and 6 columns. The class of data in each column is:

1. "Name": character
2. "Age": numeric
3. "Height": numeric
4. "Weight": numeric
5. "Gender": factor
6. "Working": character

Algorithm:

1. Create a new data frame "df2" containing the variable "Working".
2. Add the new data frame "df2" to the previous data frame "df" using the cbind() function.
3. Print the modified data frame using the print() function.

Here are the methods used in the code:

1. `c()` function: This function is used to concatenate the values of the variables "Name", "Age", "Height", "Weight", and "Gender" into vectors.
2. `as.factor()` function: This function is used to convert the "Gender" vector into a factor.

3. `data.frame()` function: This function is used to combine the vectors into a data frame.
4. `levels()` function: This function is used to access and modify the levels of the "Gender" factor.
5. `<-` operator: This operator is used to assign the modified levels of the "Gender" factor back to the data frame.
6. `print()` function: This function is used to display the modified data frame.

3. DATASETS

Consider built-in dataset state.x77.

a. Make sure the object is a data frame, if not change it to a data frame.

b. Find out how many states have an income of less than 5000.

c. Find out the state with the highest income and lowest income.

d.Rename the column state.region so that only the first 3 letters appear.

e. Remove the variable div, Life Exp, HS Grad, Frost, abb

f. Add a variable to the data frame which should categorize the level of illiteracy: [0,1) is low, [1,2) is some, [2, inf] is high.

g. Find out which state from the west, with low illiteracy, has the highest income, and what that income

ALGORITHM:

1.Check the class of the state.x77 object.

2.Convert the state.x77 object into a data frame named s77.

3.Check the class of the s77 data frame.

4.Generate a summary of the s77 data frame.

5.Count the number of rows in s77 where the value in the "Income" column is less than 5000.
6.Retrieve the row name of the row with the maximum value in the "Income" column of s77.
7.Retrieve the row name of the row with the minimum value in the "Income" column of s77.
8.Create a new data frame named df by combining specific columns from different objects and using row names from state.name.
9.Modify the column names of df by extracting a substring from the existing column names.
10.Combine the state.x77 and df data frames using the cbind() function to create a new data frame named new.df.
11.Remove the "div" column from new.df.
12.Subset new.df to remove columns 4, 6, 7, 9, and 10.
13.Create a new column named "Illiteracy.Levels" in new.df based on specific conditions using the ifelse() function.
14.Retrieve the values of the "Illiteracy.Levels" column in new.df.
15.Subset new.df to select rows where the "reg" column is "West" and the "Illiteracy.Levels" column is "Low" using the subset() function and store the result in x.
16.Retrieve the row name of the row with the maximum value in the "Income" column of x.
17.Retrieve the maximum value in the "Income" column of x.

Methods :

1.class() - This function is used to determine the class of an object, such as a data frame. It returns the class or type of the specified object.

2.data.frame() - This function is used to create a data frame from one or more vectors, matrices, or other data frames.

3.summary() - This method generates descriptive statistics of a data frame or specific columns within it. It provides a summary of the data, including minimum, 1st quartile, median, mean, 3rd quartile, and maximum values, as well as counts of missing values.

4.$ - The dollar sign operator is used to access a specific column or variable within a data frame. It allows you to retrieve or modify values in a specific column.

5.nrow() - This function is used to count the number of rows in a data frame**.**

6.row.names() - This method is used to retrieve the row names of a data frame or a subset of a data frame.

7.which.max() - This function is used to find the index of the maximum value within a vector or column of a data frame**.**

8.which.min() - This function is used to find the index of the minimum value within a vector or column of a data frame.

9.substr() - This function is used to extract a substring from a character vector or column names. It allows you to specify the starting and ending positions of the substring.

10.cbind() - This function is used to combine vectors, matrices, or data frames column-wise. It binds the columns of multiple objects together to create a new data frame.

11.NULL - The NULL keyword is used to remove a column from a data frame. Assigning NULL to a column name effectively removes that column from the data frame.

12.subset() - This function is used to create a subset of a data frame based on specified conditions. It allows you to extract rows or columns that meet specific criteria.

13.ifelse() - This function is used to perform conditional operations in R. It allows you to specify a condition and return different values based on the condition.

Code:

```
class(state.x77)

s77 <- data.frame(state.x77)
class(s77)
#B
summary(s77)
nrow(s77[s77$Income < 5000,])
#C
row.names(s77 [which.max(s77$Income),])

row.names(s77 [which.min(s77$Income),])
#d
df<-
data.frame(state.abb,state.area,state.division,state.region,ro
w.names=state.name)
colnames(df)<-substr(colnames(df),7,9)
new.df<-cbind(state.x77,df)
new.df
#e
new.df$div<-NULL
new.df<-subset(new.df,, -c(4,6,7,9,10))
new.df
#f
new.df$Illiteracy.Levels<-ifelse(new.df$Illiteracy >=0 &
new.df$Illiteracy < 1, "Low",
                    ifelse(new.df$Illiteracy >= 1 &
new.df$Illiteracy < 2,"Some",
                           "High"))
new.df$Illiteracy.Levels
```

#g
x<-subset(new.df,reg =="West" & Illiteracy.Levels =="Low")
row.names(x[which.max(x$Income),])

Output:

```
> class(state.x77)
[1] "matrix" "array"
>
> s77 <- data.frame(state.x77)
> class(s77)
[1] "data.frame"
> #B
> summary(s77)
   Population        Income       Illiteracy      Life.Exp        Murder          HS.Grad         Frost       
 Min.   :  365   Min.   :3098   Min.   :0.500   Min.   :67.96   Min.   : 1.400   Min.   :37.80   Min.   :  0.00  
 1st Qu.: 1080   1st Qu.:3993   1st Qu.:0.625   1st Qu.:70.12   1st Qu.: 4.350   1st Qu.:48.05   1st Qu.: 66.25  
 Median : 2838   Median :4519   Median :0.950   Median :70.67   Median : 6.850   Median :53.25   Median :114.50  
 Mean   : 4246   Mean   :4436   Mean   :1.170   Mean   :70.88   Mean   : 7.378   Mean   :53.11   Mean   :104.46  
 3rd Qu.: 4968   3rd Qu.:4814   3rd Qu.:1.575   3rd Qu.:71.89   3rd Qu.:10.675   3rd Qu.:59.15   3rd Qu.:139.75  
 Max.   :21198   Max.   :6315   Max.   :2.800   Max.   :73.60   Max.   :15.100   Max.   :67.30   Max.   :188.00  
      Area       
 Min.   :  1049  
 1st Qu.: 36985  
 Median : 54277  
 Mean   : 70736  
 3rd Qu.: 81163  
 Max.   :566432  
> nrow(s77[s77$Income < 5000,])
[1] 42
> #C
> row.names(s77 [which.max(s77$Income),])
[1] "Alaska"
>
> row.names(s77 [which.min(s77$Income),])
[1] "Mississippi"
> #d
> df<-data.frame(state.abb,state.area,state.division,state.region,row.names=state.name)
> colnames(df)<-substr(colnames(df),7,9)
> new.df<-cbind(state.x77,df)
> new.df
               Population Income Illiteracy Life Exp Murder HS Grad Frost   Area abb   are          div
Alabama              3615   3624        2.1    69.05   15.1    41.3    20  50708  AL  51609 East South Central
Alaska                365   6315        1.5    69.31   11.3    66.7   152 566432  AK 589757      Pacific
Arizona              2212   4530        1.8    70.55    7.8    58.1    15 113417  AZ 113909     Mountain
Arkansas             2110   3378        1.9    70.66   10.1    39.9    65  51945  AR  53104 West South Central
California          21198   5114        1.1    71.71   10.3    62.6    20 156361  CA 158693      Pacific
Colorado             2541   4884        0.7    72.06    6.8    63.9   166 103766  CO 104247     Mountain
Connecticut          3100   5348        1.1    72.48    3.1    56.0   139   4862  CT   5009  New England
Delaware              579   4809        0.9    70.06    6.2    54.6   103   1982  DE   2057 South Atlantic
Florida              8277   4815        1.3    70.66   10.7    52.6    11  54090  FL  58560 South Atlantic
Georgia              4931   4091        2.0    68.54   13.9    40.6    60  58073  GA  58876 South Atlantic
Hawaii                868   4963        1.9    73.60    6.2    61.9     0   6425  HI   6450      Pacific
Idaho                 813   4119        0.6    71.87    5.3    59.5   126  82677  ID  83557     Mountain
Illinois            11197   5107        0.9    70.14   10.3    52.6   127  55748  IL  56400 East North Central
```

R 4.3.0 · C:/Users/vishu/Desktop/R/

```
Ohio            10735   4561        0.8    70.82    7.4    53.2   124  40975  OH  41222 East North Central
Oklahoma         2715   3983        1.1    71.42    6.4    51.6    82  68782  OK  69919 West South Central
Oregon           2284   4660        0.6    72.13    4.2    60.0    44  96184  OR  96981           Pacific
Pennsylvania    11860   4449        1.0    70.43    6.1    50.2   126  44966  PA  45333    Middle Atlantic
Rhode Island      931   4558        1.3    71.90    2.4    46.4   127   1049  RI   1214        New England
South Carolina   2816   3635        2.3    67.96   11.6    37.8    65  30225  SC  31055     South Atlantic
South Dakota      681   4167        0.5    72.08    1.7    53.3   172  75955  SD  77047 West North Central
Tennessee        4173   3821        1.7    70.11   11.0    41.8    70  41328  TN  42244 East South Central
Texas           12237   4188        2.2    70.90   12.2    47.4    35 262134  TX 267339 West South Central
Utah             1203   4022        0.6    72.90    4.5    67.3   137  82096  UT  84916           Mountain
Vermont           472   3907        0.6    71.64    5.5    57.1   168   9267  VT   9609        New England
Virginia         4981   4701        1.4    70.08    9.5    47.8    85  39780  VA  40815     South Atlantic
Washington       3559   4864        0.6    71.72    4.3    63.5    32  66570  WA  68192           Pacific
West Virginia    1799   3617        1.4    69.48    6.7    41.6   100  24070  WV  24181     South Atlantic
Wisconsin        4589   4468        0.7    72.48    3.0    54.5   149  54464  WI  56154 East North Central
Wyoming           376   4566        0.6    70.29    6.9    62.9   173  97203  WY  97914           Mountain
                   reg
Alabama          South
Alaska            West
Arizona           West
Arkansas         South
California        West
Colorado          West
Connecticut  Northeast
Delaware         South
Florida          South
Georgia          South
Hawaii            West
```

R 4.3.0 · C:/Users/vishu/Desktop/R/

```
Idaho             West
Illinois  North Central
Indiana   North Central
Iowa      North Central
Kansas    North Central
Kentucky          South
Louisiana         South
Maine       Northeast
Maryland          South
Massachusetts Northeast
Michigan  North Central
Minnesota North Central
Mississippi       South
Missouri  North Central
Montana           West
Nebraska  North Central
Nevada            West
New Hampshire Northeast
New Jersey    Northeast
New Mexico        West
New York      Northeast
North Carolina    South
North Dakota  North Central
Ohio          North Central
Oklahoma          South
Oregon            West
Pennsylvania  Northeast
Rhode Island  Northeast
South Carolina    South
Wyoming           West
> #e
> new.df$div<-NULL
> new.df<-subset(new.df,, -c(4,6,7,9,10))
> new.df
              Population Income Illiteracy Murder   Area           reg
Alabama             3615   3624        2.1   15.1  50708         South
Alaska               365   6315        1.5   11.3 566432          West
Arizona             2212   4530        1.8    7.8 113417          West
Arkansas            2110   3378        1.9   10.1  51945         South
California         21198   5114        1.1   10.3 156361          West
Colorado            2541   4884        0.7    6.8 103766          West
Connecticut         3100   5348        1.1    3.1   4862     Northeast
Delaware             579   4809        0.9    6.2   1982         South
Florida             8277   4815        1.3   10.7  54090         South
Georgia             4931   4091        2.0   13.9  58073         South
Hawaii               868   4963        1.9    6.2   6425          West
Idaho                813   4119        0.6    5.3  82677          West
Illinois           11197   5107        0.9   10.3  55748 North Central
Indiana             5313   4458        0.7    7.1  36097 North Central
Iowa                2861   4628        0.5    2.3  55941 North Central
Kansas              2280   4669        0.6    4.5  81787 North Central
Kentucky            3387   3712        1.6   10.6  39650         South
Louisiana           3806   3545        2.8   13.2  44930         South
Maine               1058   3694        0.7    2.7  30920     Northeast
Maryland            4122   5299        0.9    8.5   9891         South
Massachusetts       5814   4755        1.1    3.3   7826     Northeast
Michigan            9111   4751        0.9   11.1  56817 North Central
```

```
Maryland          4122   5299        0.9    8.5   9891          South
Massachusetts     5814   4755        1.1    3.3   7826      Northeast
Michigan          9111   4751        0.9   11.1  56817 North Central
Minnesota         3921   4675        0.6    2.3  79289 North Central
Mississippi       2341   3098        2.4   12.5  47296          South
Missouri          4767   4254        0.8    9.3  68995 North Central
Montana            746   4347        0.6    5.0 145587          West
Nebraska          1544   4508        0.6    2.9  76483 North Central
Nevada             590   5149        0.5   11.5 109889          West
New Hampshire      812   4281        0.7    3.3   9027      Northeast
New Jersey        7333   5237        1.1    5.2   7521      Northeast
New Mexico        1144   3601        2.2    9.7 121412          West
New York         18076   4903        1.4   10.9  47831      Northeast
North Carolina    5441   3875        1.8   11.1  48798          South
North Dakota       637   5087        0.8    1.4  69273 North Central
Ohio             10735   4561        0.8    7.4  40975 North Central
Oklahoma          2715   3983        1.1    6.4  68782          South
Oregon            2284   4660        0.6    4.2  96184          West
Pennsylvania     11860   4449        1.0    6.1  44966      Northeast
Rhode Island       931   4558        1.3    2.4   1049      Northeast
South Carolina    2816   3635        2.3   11.6  30225          South
South Dakota       681   4167        0.5    1.7  75955 North Central
Tennessee         4173   3821        1.7   11.0  41328          South
Texas            12237   4188        2.2   12.2 262134          South
Utah              1203   4022        0.6    4.5  82096          West
Vermont            472   3907        0.6    5.5   9267      Northeast
Virginia          4981   4701        1.4    9.5  39780          South
Washington        3559   4864        0.6    4.3  66570          West
```

```
Console  Terminal  Background Jobs
R 4.3.0 · C:/Users/vishu/Desktop/R/
Oregon              2284   4660        0.6    4.2  96184          West
Pennsylvania       11860   4449        1.0    6.1  44966      Northeast
Rhode Island         931   4558        1.3    2.4   1049      Northeast
South Carolina      2816   3635        2.3   11.6  30225          South
South Dakota         681   4167        0.5    1.7  75955 North Central
Tennessee           4173   3821        1.7   11.0  41328          South
Texas              12237   4188        2.2   12.2 262134          South
Utah                1203   4022        0.6    4.5  82096          West
Vermont              472   3907        0.6    5.5   9267      Northeast
Virginia            4981   4701        1.4    9.5  39780          South
Washington          3559   4864        0.6    4.3  66570          West
West Virginia       1799   3617        1.4    6.7  24070          South
Wisconsin           4589   4468        0.7    3.0  54464 North Central
Wyoming              376   4566        0.6    6.9  97203          West
> #f
> new.df$Illiteracy.Levels<-ifelse(new.df$Illiteracy >=0 & new.df$Illiteracy < 1, "Low",
+                                  ifelse(new.df$Illiteracy >= 1 & new.df$Illiteracy < 2,"Some",
+                                         "High"))
> new.df$Illiteracy.Levels
 [1] "High" "Some" "Some" "Some" "Some" "Low"  "Some" "Low"  "Some" "High" "Some" "Low"  "Low"  "Low"  "Low"  "Low"  "Some"
[18] "High" "Low"  "Low"  "Some" "Low"  "Low"  "High" "Low"  "Low"  "Low"  "Low"  "Low"  "Some" "High" "Some" "Some" "Low"
[35] "Low"  "Some" "Low"  "Some" "Some" "High" "Low"  "Some" "High" "Low"  "Low"  "Some" "Low"  "Some" "Low"  "Low"
>
> #g
> x<-subset(new.df,reg =="West" & Illiteracy.Levels =="Low")
> row.names(x[which.max(x$Income),])
[1] "Nevada"
>
```

4. Loop implementation

a. Write a nested loop, where the outer for loop increments "a" 3 times, and the inner for loop increments "b" 3 times. The break statement exits the inner for loop after 2 incrementations. The nested loop prints the values of variables, "a" and "b".

b. The next statement is used within loops in order to skip the current evaluation, and instead proceed to the next evaluation. Therefore, write a while loop that prints the variable "; that is incremented from 2 - 5, and uses the next statement, to skip the printing of the number.

ALGORITHM: A: 1. Start

2. Set the initial value of "a" to 1.

3. Start the outer loop:

4. Repeat the following steps 3 times:

a. Set the initial value of "b" to 1.

b. Start the inner loop:

c. Print the values of "a" and "b".

d. Increment the value of "b" by 1.

e. If the value of "b" is equal to 2, break out of the inner loop.

f. Increment the value of "a" by 1.

5. End the outer loop.

6. End B.

1. Start

2. Set the initial value of the variable "x" to 2.

3. Start the while loop:

4. Repeat the following steps as long as the value of "x" is less than or equal to

5: a. If the value of "x" is equal to 4, use the "next" statement to skip the current iteration.

b. Print the value of "x".

c. Increment the value of "x" by 1.

5. End the while loop

. 6. End

METHODS USED IN THE CODE: 1. for()- used to repeat a specific block of code a known number of times. Syntax: for(expression1;expression2;expression3({ //code to be executed }

2. break()- a loop control statement that is used to terminate the loop. Syntax: break;

3. print()- used to print the statement given within the double codes. Syntax: print(“ “);

4. if()- a decision making statement that is used to execute a block of code based on the value of the given expression. Syntax: if(expression){ //code to be executed if condition is true }else{ //code to be executed if the condition is false }

5. while()- allows user to repeat a statement until a specified expression becomes false. Syntax: while(condition){ //code to be executed }

6. next()- a continue statement causes the iteration to go to the next statement of the loop, skipping the code in between. Syntax: continue; CODE: a. for (a in 1:3)

```
{
for (b in 1:3)
{
 print(paste("a =", a, "b =", b))
 if (b == 2)
 {
 break
 }
}
}
```

b.

```
i <- 2 while (i <= 5)
{
```

```
if (i == 4)
{
i <- i + 1 next
 }
 print(i)
 i <- i + 1
}
```

5. Create a function that given an integer will calculate how many divisors it has (other than 1 and itself). Print all the divisors and its count.

ALGORITHM

1. Start with an empty list of divisors.
2. Iterate through the numbers from 2 to (num - 1).
3. For each number, check if it divides the given integer evenly (i.e., the remainder of the division is 0).
4. If the number is a divisor, add it to the list of divisors.
5. After iterating through all the numbers, count the number of divisors in the list.
6. Print the list of divisors and the count.

PROGRAM CODE:

R program that calculates the number of divisors for a given integer and prints all the divisors along with their count:

```
countDivisors <- function(num) {
 divisors <- c()
 # Find divisors
 for (i in 2:(num-1)) {
  if (num %% i == 0) {
   divisors <- c(divisors, i)
  }
 }
```

```
  divisorCount <- length(divisors)

  # Print divisors and count

  cat("Divisors:", divisors, "\n")

  cat("Number of Divisors:", divisorCount, "\n")

}

# Read input from user

num <- as.integer(readline("Enter an integer: "))

# Call the function with user input

countDivisors(num)
```

FUNCTIONS EXPLAINATION:

1. countDivisors(num): This is the main function that calculates the number of divisors for a given integer. It takes an input parameter `num`, representing the integer for which we want to find the divisors. Here's a detailed explanation of the steps within this function:

- divisors: This is an empty vector that will store the divisors of the given integer `num`.
- for loop: It iterates through the numbers from 2 to `(num - 1)`. This loop will check each number to see if it divides `num` evenly.
- if statement: Within the loop, it checks if the current number `i` divides `num` evenly, which is determined by `num %% i == 0`. If the condition is true, it means that `i` is a divisor of `num`.
- Adding divisors: If `i` is a divisor, it is added to the `divisors` vector using the `c()` function. This appends the current divisor to the existing list of divisors.

divisorCount: After the loop completes, we determine the number of divisors by calculating the length of the `divisors` vector using the `length()` function.

Printing divisors and count: Finally, the function prints the list of divisors using the `cat()` function, followed by the count of divisors.

2. readline("Enter an integer: "): This function is used to read input from the user. It prompts the user to enter an integer and waits for the user's input. The input provided by the user is returned as a string.

3. as.integer(): This function is used to convert the user input (which is initially a string) into an integer data type. It takes a value as an argument and attempts to convert it into an integer. In this code, it is used to convert the user input string into an integer and store it in the variable `num`.

4. cat("Divisors:", divisors, "\n"): This function is used to print the divisors of the given integer. It takes three arguments: a string `"Divisors:"`, the `divisors` vector containing the divisors, and `"\n"` to add a newline character after printing the divisors.

5. cat("Number of Divisors:", divisorCount, "\n"): This function is used to print the count of divisors. It takes three arguments: a string `"Number of Divisors:"`, the `divisorCount` variable storing the count, and `"\n"` to add a newline character after printing the count.

These functions work together to calculate the divisors of a given integer, read user input, convert the input to an integer, and display the divisors along with their count.

OUTPUT:

```R
Enter an integer: 24
Divisors: 2 3 4 6 8 12
Number of Divisors: 6
```

In this example, the user inputs the number 24. The code then calculates the divisors of 24, which are 2, 3, 4, 6, 8, and 12. The code prints the divisors and the count, indicating that there are 6 divisors for the number 24.

6. Create a function that given a data frame, and a number or character. The function will return the data frame with the character or number changed to NA.

ALGORITHM:-

The algorithm used in the replace_with_na function is as follows:

The function replace_with_na takes two arguments: data_frame, which represents the input data frame, and value, which is the number or character to be replaced with NA.

The function loops through each column of the data frame using for (col in colnames(data_frame)).

Within the loop, the function checks if the elements in the current column (data_frame[[col]]) are equal to the specified value (value) using data_frame[[col]] == value.

If an element in the column matches the specified value, the function assigns NA to that element using the assignment operator <- NA.

After iterating through all the columns, the modified data frame is returned using return(data_frame).

METHODS USED IN THE CODE:-

1. for()- used to repeat a specific block of code a known number

of times.

Syntax: for(expression1;expression2;expression3(

```
{
//code to be executed
}
```

CODE:-

```
# Here we have created sample data frame
df <- data.frame(
  col1 = c(1, 2, 3, 4),
  col2 = c("a", "b", "c", "d"),
  col3 = c(5, 6, 7, 8)
)
# Print the original data frame
print(df);
```

OUTPUT:-

```
col1 col2 col3
1   1   a   5
2   2   b   6
3   3   c   7
4   4   d   8
```

After Using Replace using NA Function

Here's a function in R that takes a data frame and a number or character and replaces all occurrences of that number or character with NA

CODE:-

```
replace_with_na <- function(data_frame, value) {
  # Loop through each column of the data frame
  for (col in colnames(data_frame)) {
    # Replace the specified value with NA in the column
    data_frame[[col]][data_frame[[col]] == value] <- NA
  }

  return(data_frame)
```

```
}
# Create a sample data frame
df <- data.frame(
  col1 = c(1, 2, 3, 4),
  col2 = c("a", "b", "c", "d"),
  col3 = c(5, 6, 7, 8)
)

# Print the original data frame
print(df)

# Call the replace_with_na function to replace "b" with NA
df_new <- replace_with_na(df, "b")

# Print the modified data frame
print(df_new)
```

OUTPUT:-

```
col1 col2 col3
1    1    a    5
2    2    b    6
3    3    c    7
4    4    d    8


col1 col2 col3
1    1    a    5
2    2 <NA>    6
3    3    c    7
4    4    d    8
```

Here the function replace_with_na is called with the data frame df and the character “b” .it replaces all occurrences of ”b” in the data frame with NA and returns the modified data frame df_new.

7. Consider the file table0.txt containing the following contents.

Name	Age	Height	Weight	Gender
Ram	30	177	57	M
Alwin	35	164	48	F
Billy	--	155	45	M
Amera	16	*	60	F
Olive	42	124	**	F
Dora	59	150	55	F

a. Read the contents of the file.
b. Find the missing values.
c. Replace the missing values on column 4, with the average of the column values.

Step 1: Prepare the file

Create a file named table0.txt with the provided contents:

Name	Age	Height	Weight	Gender
Ram	30	177	57	M
Alwin	35	164	48	F
Billy	--	155	45	M
Amera	16	*	60	F
Olive	42	124	**	F
Dora	59	150	55	F

Step 2: Set up R environment

We have installed ‘**RStudio**’ for the implementation.

Step 3: Algorithm

1. Read the contents of the file "table0.txt" using the ‘read.delim()’ function and store it in the ‘contents’ data frame.

2. Find the missing values:

a. Create a subset of ‘contents’ using the ‘subset()’ function, filtering rows where ‘Age’, ‘Height’, or ‘Weight’ values are ‘—', ‘*’, or ‘**’.

b. Store the subset in ‘missing_values’.

c. Print “Missing values” and ‘missing_values’.

3. Replace the missing values with the average:

a. Convert the 'Weight' column of `contents` to numeric values using ‘as numeric(as.character())’.

b. Calculate the average of the numeric values using the ‘mean()’ function, excluding any missing values (‘NA’) by specifying ‘na.rm = TRUE’.

c. Assign the average value to the rows in the 'Weight' column where the value is ‘—' or ‘**’ using logical indexing.

d. Print “Modified contents” and contents.

Step 4: Implementation of the code

find_missing.r

```r
# Read the content of the file:
file_path <- "table0.txt"
contents <- read.delim(file_path, header = TRUE, sep = "\t")
print(contents)


#Find the missing values:
missing_values <- subset(contents, Age == "--" | Height == "--" | Weight == "--" | Age
print("Missing values:")
print(missing_values)


# Replace the missing values in column 4 with the average of the column values:
numeric_values <- as.numeric(as.character(contents$Weight))
average <- mean(numeric_values, na.rm = TRUE)

contents$Weight[contents$Weight == "--" | contents$Weight == "**"] <- average

print("Modified contents:")
print(contents)
```

Explanation:

a. Reading the contents of the file:

We use the read.table() function to read the contents of the file "table0.txt". We specify header = TRUE to indicate that the first row contains column names, and sep = "\t" to specify that the columns are separated by a tab character.

b. Finding the missing values:

We create a logical matrix missing_values to identify missing values in the data. In this case, we consider "--", "**", and "*" as missing values. The expression is.na(data) checks for NA values, and data == "--" checks for values equal to "--", and so on.

c. Replacing missing values in column 4:

We calculate the average of the values in the "Weight" column using the mean() function, and store it in the variable average_weight. Then, we replace the missing values in the "Weight" column with the calculated average by assigning average_weight to the corresponding elements of data$Weight.

Finally, we print the updated data using print(data).

We have to make sure that the file "table0.txt" is in the working directory of R environment or else we need to provide the full path to the file in the read.table() function.

Functions used in the Program:

1. read.delim(): This function is used to read tabular data from a file with delimiter-separated values. It is similar to `read.table()`, but specifically designed to read tab-separated files. In the code, it reads the contents of the file "table0.txt" and returns a data frame, which is stored in the variable `contents`.

2. subset(): This function is used to extract a subset of rows from a data frame based on specified conditions. In the code, it is used to create a subset `missing_values` by filtering rows where the values in the 'Age', 'Height', or 'Weight' columns are '--', '*', or '**'.

3. print(): This function is used to display output on the console. In the code, it is used to print the messages and the contents of the data frames (`missing_values` and `contents`).

4. as.numeric(): This function is used to convert a vector or a variable to a numeric type. In the code, it is used to convert the 'Weight' column of the 'contents' data frame to numeric values. 'as.character()' is used to convert the values to character type before converting them to numeric.

5. mean(): This function is used to calculate the mean (average) of a numeric vector or values. In the code, it is used to calculate the average of the numeric values in the 'Weight' column, excluding any missing values (`NA`). The `na.rm = TRUE` parameter is used to handle missing values and ignore them in the calculation.

6. Logical Indexing: Logical indexing is used to select specific rows in a data frame based on a condition. In the code, it is used to identify the rows in the 'Weight' column where the value is '--' or '**', and replace those values with the calculated average.

Step 5: Run the code.

```
> file_path <- "table0.txt"
> contents <- read.delim(file_path, header = TRUE, sep = "\t")
> print(contents)
   Name Age Height Weight Gender
1   Ram  30    177     57      M
2 Alwin  35    164     48      F
3 Billy  --    155     45      M
4 Amera  16      *     60      F
5 Olive  42    124     **      F
6  Dora  59    150     55      F
> missing_values <- subset(contents, Age == "--" | Height == "--" | Weight == "--" | Age == "*" | Height == "*" | Weight == "**")
> print("Missing values:")
[1] "Missing values:"
> print(missing_values)
   Name Age Height Weight Gender
3 Billy  --    155     45      M
4 Amera  16      *     60      F
5 Olive  42    124     **      F
> numeric_values <- as.numeric(as.character(contents$Weight))
Warning message:
NAs introduced by coercion
> average <- mean(numeric_values, na.rm = TRUE)
>
> contents$Weight[contents$Weight == "--" | contents$Weight == "**"] <- average
>
> print("Modified contents:")
[1] "Modified contents:"
> print(contents)
   Name Age Height Weight Gender
1   Ram  30    177     57      M
2 Alwin  35    164     48      F
3 Billy  --    155     45      M
4 Amera  16      *     60      F
5 Olive  42    124     53      F
6  Dora  59    150     55      F
```

8. . Read the file Table1.txt

Ram	30	177	57	M
Alwin	35	164	48	F
Billy	28	155	45	M
Amera	16	168	60	F
Olive	42	124	70	F
Dora	59	150	55	F

a. Change the names of the columns to Name, Age, Height, Weight and Gender

b. Change the row names so that they are the same as Name, and remove the variable Name.

Program code:

 # Read the file

```
data <- read.table("Table1.txt", header = FALSE)
```

 # Set the column names

```
colnames(data) <- c("Name", "Age", "Height", "Weight", "Gender")
```

 # Set the row names to the values in the first column (Name)

```
rownames(data) <- data$Name
```

 # Remove the Name variable

```
data <- data[, -1]
```

 # Print the modified data

```
print(data)
```

Title: R Program to Modify Column and Row Names in Table Data

Program Steps:

1. Reading the File:

The program starts by reading the contents of the "Table1.txt" file using the `read.table()` function. The `read.table()` function is a versatile tool in R used to read tabular data from files. In this case, we set the `header` parameter to `FALSE` since the file does not contain a header row.

2. Changing Column Names:

To change the column names, the program uses the `colnames()` function. By assigning a character vector with the desired column names to `colnames(data)`, the program modifies the column names to "Name," "Age," "Height," "Weight," and "Gender." This step ensures that the column names accurately represent the corresponding variables in the data.

3. Setting Row Names:

Next, the program sets the row names of the data using the `rownames()` function. To achieve this, the program assigns the values from the "Name" column to `rownames(data)`. By doing so, the row names become identical to the values in the "Name" column, enabling easier identification and indexing of rows.

4. Removing the Name Variable:

To remove the "Name" variable from the data, the program uses R's subsetting capabilities. By excluding the first column (Name) from the data, the program retains only the "Age," "Height," "Weight," and "Gender" variables. This step ensures that the modified data no longer contains the "Name" variable.

5. Printing the Modified Data:

Finally, the program prints the modified data to the console using the `print()` function. The resulting output displays the table with updated column names and row names, providing a clear representation of the modified data.

Algorithms:

1. Read the file:

   1. Start
   2. Read the file "Table1.txt" using the read.table() function and assign the data to a variable (e.g., data)
   3. Return the data
   4. Stop

2. Set the column names:
    1. Start
    2. Set the column names of the data using the colnames() function
    3. Assign the desired column names to colnames(data) as a character vector
    4. Return the modified data
    5. Stop
3. Set the Row names:
    1. Start
    2. Set the row names of the data using the rownames() function
    3. Assign the values from the "Name" column of the data to rownames(data)
    4. Return the modified data
    5. Stop
4. Removing the Name Variable:
    1. Start
    2. Remove the first column (Name) from the data using subsetting
    3. Assign the modified data to a new variable (e.g., data_modified)
    4. Return the data_modified
    5. Stop
5. Printing the modified data:
    1. Start
    2. Print the modified data using the print() function
    3. Stop

Outputs:

- **dataset**

    V1 V2 V3 V4 V5

    1 Ram 30 177 57 M

    2 Alwin 35 164 48 F

3 Billy 28 155 45 M

4 Amera 16 168 60 F

5 Olive 42 124 70 F

6 Dora 59 150 55 F

- Set the column names

```
> colnames(data) <- c("Name", "Age", "Height", "Weight", "Gender")
> data
```

Name Age Height Weight Gender

1 Ram 30 177 57 M

2 Alwin 35 164 48 F

3 Billy 28 155 45 M

4 Amera 16 168 60 F

5 Olive 42 124 70 F

6 Dora 59 150 55 F

- **Set the row names to the values in the first column (Name)**

```
> rownames(data) <- data$Name
> data
```

Name Age Height Weight Gender

Ram Ram 30 177 57 M

Alwin Alwin 35 164 48 F

Billy Billy 28 155 45 M

Amera Amera 16 168 60 F

Olive Olive 42 124 70 F

Dora Dora 59 150 55 F

- **Remove the Name variable**

```
> data <- data[, -1]
> # Print the modified data
> print(data)
```

	Age	Height	Weight	Gender
Ram	**30**	**177**	**57**	**M**
Alwin	**35**	**164**	**48**	**F**
Billy	**28**	**155**	**45**	**M**
Amera	**16**	**168**	**60**	**F**
Olive	**42**	**124**	**70**	**F**
Dora	**59**	**150**	**55**	**F**

9. Plot Data

a. Plot Miles/(US) gallon versus Rear axle ratio by plot(mpg,drat).

b. Change type of visualisation to scatterplot.

Algorithm:

Load the mtcars dataset.

Plot the Miles/(US) gallon versus Rear axle ratio using the plot() function with 'mpg' and 'drat' columns.

Customize the plot by setting the title, x-axis label, and y-axis label.

Display the line plot.

Modify the plot by changing the visualization type to a scatter plot.

Display the scatter plot.

Methods Used in the Code:

a. plot() - Used to create a basic line or scatter plot.

b. data() - Used to load the mtcars dataset.

c. main, xlab, ylab - Arguments used to set the title, x-axis label, and y-axis label in the plot.

Code:

```
# Load the mtcars dataset

data(mtcars)

# Plot Miles/(US) gallon versus Rear axle ratio as a line plot

plot(mtcars$mpg, mtcars$drat, main = "Miles/(US) gallon versus Rear axle ratio",

    xlab = "Miles/(US) gallon", ylab = "Rear axle ratio")


# Change the visualization to a scatter plot

plot(mtcars$mpg, mtcars$drat, main = "Miles/(US) gallon versus Rear axle ratio (Scatter Plot)",

    xlab = "Miles/(US) gallon", ylab = "Rear axle ratio", type = "p")
```

Step 1: Load the dataset

To begin, we load the mtcars dataset into our R environment. The following code accomplishes this task:

```
data(mtcars)
```

Step 2: Plotting Miles/(US) gallon versus Rear axle ratio as a line plot

For the initial line plot, we make use of the plot() function, considering the 'mpg' and 'drat' columns from the mtcars dataset. Our code implementation is as follows:

```
plot(mtcars$mpg, mtcars$drat, main = "Miles/(US) gallon versus Rear axle ratio",

    xlab = "Miles/(US) gallon", ylab = "Rear axle ratio")
```

This code generates a line plot where the x-axis represents Miles/(US) gallon, while the y-axis represents Rear axle ratio. The main, xlab, and ylab arguments are utilized to set the title, x-axis label, and y-axis label, respectively.

Output:

The output of the above code is a line plot showing the relationship between Miles/(US) gallon and Rear axle ratio.

Step 3: Changing the visualization to a scatter plot

To transform the line plot into a scatter plot, we modify the code by including the type argument within the plot() function. Here is the updated code:

```
plot(mtcars$mpg, mtcars$drat, main = "Miles/(US) gallon versus Rear axle ratio (Scatter Plot)",
    xlab = "Miles/(US) gallon", ylab = "Rear axle ratio", type = "p")
```

This code produces a scatter plot where each data point corresponds to a car from the mtcars dataset. The title, x-axis label, and y-axis label are appropriately specified.

Output:

The output of the modified code is a scatter plot illustrating the relationship between Miles/(US) gallon and Rear axle ratio.

Explanation:

In our solution, we followed a step-by-step approach to plot the Miles/(US) gallon versus Rear axle ratio using the mtcars dataset. Firstly, we loaded the mtcars dataset using the data(mtcars) command.

Then, we created a line plot by utilizing the plot() function with the 'mpg' and 'drat' columns from the mtcars dataset. The resulting plot demonstrated the relationship between the two variables.

To change the visualization to a scatter plot, we updated the code by incorporating the type = "p" argument within the plot() function. This alteration transformed the line plot into a scatter plot, providing a different representation of the data.

10. Select four different continuous variables from the mtcars dataset.

a. Plot with lines and points for different variables with different colors.

b. Choose one variable from each and highlight it with different color.

```
# Function to create the employee data frame
  createEmployeeDataFrame <- function() {
    # Read the number of employees from the user
    numEmployees <- readline("Enter the number of employees: ")
    numEmployees <- as.integer(numEmployees)
    # Initialize empty vectors for each field
    empID <- vector("integer", numEmployees)
    empName <- vector("character", numEmployees)
    doj <- vector("character", numEmployees)
    dept <- vector("character", numEmployees)
    desig <- vector("character", numEmployees)

    # Loop to read employee details
    for (i in 1:numEmployees) {
      cat(paste("Enter details for Employee", i, ":\n"))

      # Read employee details from the user
      empID[i] <- readline("EmpID: ")
      empName[i] <- readline("EmpName: ")
      doj[i] <- readline("DOJ: ")
      dept[i] <- readline("Dept: ")
      desig[i] <- readline("Desig: ")
    }

    # Create the data frame
    employeeData <- data.frame(EmpID = empID, EmpName = empName, DOJ = doj, Dept = dept,
Desig = desig)
```

```
  return(employeeData)
}

# Function to write the employee data frame to CSV
writeDataFrameToCSV <- function(employeeData, fileName) {
  write_csv(employeeData, fileName)
  cat(paste("Employee data written to", fileName, "successfully!\n"))
}

# Function to read and display the contents of the CSV file
readAndDisplayCSV <- function(fileName) {
  employeeData <- read_csv(fileName)
  cat(paste("Contents of", fileName, ":\n"))
  print(employeeData)
}

# Function to append a new row to the CSV file
appendRowToCSV <- function(employeeData, fileName) {
  newEmployeeData <- createEmployeeDataFrame()
  appendedData <- rbind(employeeData, newEmployeeData)
  write_csv(appendedData, fileName)
  cat(paste("New row appended to", fileName, "successfully!\n"))
}

# Main program
csvFileName <- "check.csv"

# Step 1: Create the employee data frame
employeeData <- createEmployeeDataFrame()

# Step 2: Store the data frame in the CSV file
```

```
writeDataFrameToCSV(employeeData, csvFileName)


# Step 3: Read and display the contents of the CSV file

readAndDisplayCSV(csvFileName)


# Step 4: Append a new row to the CSV file

appendRowToCSV(employeeData, csvFileName)
```

ALGORITHM

1. The `createEmployeeDataFrame` function prompts the user to enter the number of employees. It then initializes empty vectors for each employee field (EmpID, EmpName, DOJ, Dept, Desig). It uses a loop to read employee details from the user and stores them in the respective vectors. Finally, it creates a data frame using the collected employee data and returns it.

2. The `writeDataFrameToCSV` function takes an employee data frame and a file name as inputs. It writes the data frame to a CSV file using the `write_csv` function from the appropriate library. It displays a success message after the operation.

3. The `readAndDisplayCSV` function takes a file name as an input. It reads the contents of the CSV file using the `read_csv` function and displays them using the `print` function.

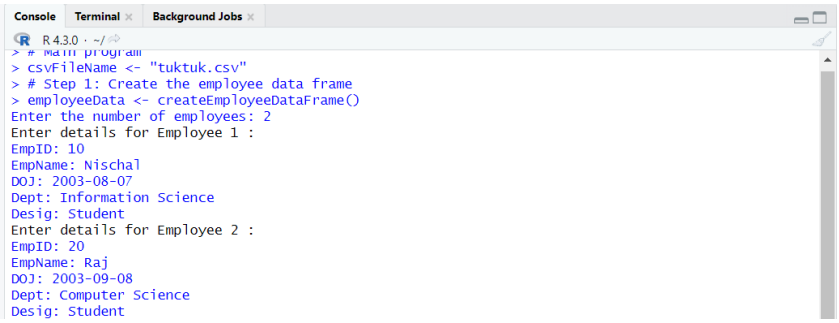
4. The `appendRowToCSV` function takes an existing employee data frame and a file name as inputs. It calls the `createEmployeeDataFrame` function to collect details for a new employee. It then appends the new employee data to the existing data frame using the `rbind` function. The updated data frame is written to the CSV file using the `write_csv` function, and a success message is displayed.

METHODS USED

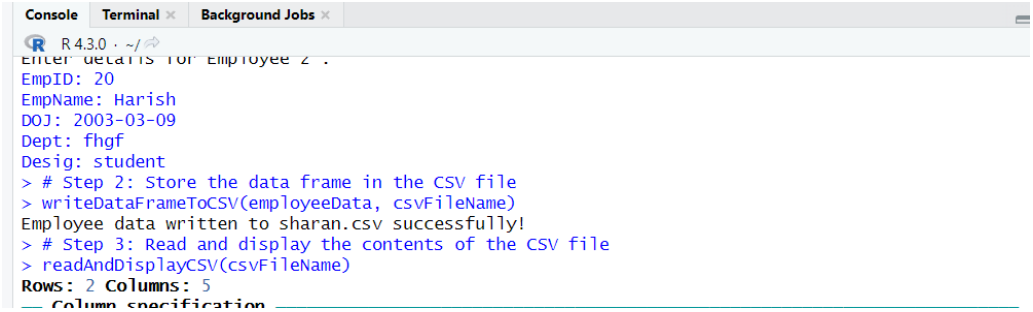
- 'readline': Used to read user input for the number of employees and employee details.
- 'vector': Used to initialize empty vectors for each employee field.
- 'for' loop: Used to iterate over the employee details input process.
- 'rbind': Used to append a new row to the employee data frame.
- 'write_csv': Used to write the employee data frame to a CSV file.
- 'read_csv': Used to read the contents of the CSV file.
- 'print': Used to display the contents of the data frame.

OUTPUT OF THE PROGRAM

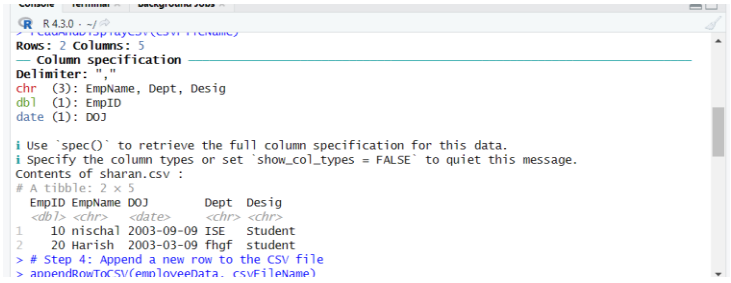
- Upon execution, the program prompts the user to enter the number of employees and employee details for each employee.



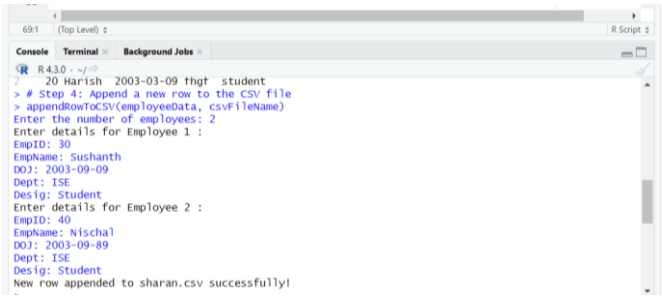
- It writes the employee data to a CSV file and displays a success message.



- It reads the contents of the CSV file and displays them.



- It prompts the user to enter details for a new employee and appends the new row to the CSV file.



After each operation, appropriate success messages or contents of the data frame are displayed.