Name of Student: Pushkar Sane		
Roll Number: 45	**Lab Assignment Number: 2**	
Title of Lab Assignment: Write commands for following cbind-ing and rbinding, Reading and Writing data. setwd(), getwd(), data(), rm(), Attaching and Detaching data. Reading data from the consol. Loading data from different data sources.(CSV, Excel).		
DOP: 19-09-2023	**DOS: 20-09-2023**	
CO Mapped: **CO1**	**PO Mapped:** **PO1, PO2, PO3, PO4, PO6, PO7, PSO1, PSO2**	**Signature:**

<u>Aim:</u> Write commands for following cbind-ing and rbind-ing, Reading and Writing data. setwd(), getwd(), data(), rm(), Attaching and Detaching data. Reading data from the console, Loading data from different data sources.(CSV, Excel).

<u>Description:</u>

1. **<u>Cbind-ing and Rbind-ing:</u>** In R, `cbind` and `rbind` are two functions used for combining data frames or matrices either by columns (`cbind`) or by rows (`rbind`). These functions are essential for data manipulation and aggregation tasks. Let's take a closer look at each of them:
   a. **`cbind` (Column Bind):**
      i. `cbind` stands for "column bind," and it is used to combine objects (usually data frames or matrices) by adding their columns side by side.
      ii. Syntax: `cbind(object1, object2, ...)`
      iii. Each object can be a data frame, matrix, or vector.
      iv. If you provide multiple objects to `cbind`, it will combine them by columns, aligning the columns from each object together
      v. If the objects have different numbers of rows, `cbind` will use recycling to match the shorter object's length to the longer one.
      vi. Example:

```
# Creating two data frames
 df1 <- data.frame(Name = c("Alice", "Bob", "Charlie"),
 Age = c(25, 30, 22))
 df2 <- data.frame(Score = c(95, 88, 75))
 # Combining them using cbind
 combined <- cbind(df1, df2)
 print(combined)

 Output:
     Name     Age    Score
 1   Alice     25     95
 2   Bob      30     88
 3   Charlie  22     75
```

b. `rbind` (Row Bind):

i. `rbind` stands for "row bind," and it is used to combine objects by stacking them on top of each other, creating a new object with more rows.
ii. Syntax: `rbind (object1, object2, ...)`
iii. Each object can be a data frame, matrix, or vector.
iv. If you provide multiple objects to `rbind`, it will combine them by rows, aligning the rows from each object together.
v. Like `cbind`, if the objects have different numbers of columns, `rbind` will use recycling to match the shorter object's width to the longer one.
vi. Example:

```
# Creating two data frames
df1 <- data. frame(Name = c("Alice", "Bob"),
Age = c(25, 30))
df2 <- data.frame(Name = c("Charlie"),
Age = c(22))
# Combining them using rbind
combined <- rbind(df1, df2)
print(combined)
```

Output:

```
   Name     Age
 1 Alice     25
 2 Bob      30
 3 Charlie  22
```

vii. In summary, `cbind` and `rbind` are versatile functions in R that allow you to combine data frames or matrices either by columns or by rows, depending on your data manipulation needs.
viii. These functions are commonly used in data preprocessing, data merging, and reshaping tasks.

2. **<u>Reading and Writing Data</u>**

Reading and writing data are fundamental tasks in data analysis using R. R provides various functions and packages to handle different data formats, allowing you to import and export data efficiently. Here's a detailed overview of reading and writing data using R:

<u>Reading Data:</u>

a. **Reading from Text Files:**
   i. `read.csv()`, `read.table()`, and `read.delim()` are commonly used functions to read data from text files (e.g., CSV, TSV, plain text).
   ii. Example:
```
# Read data from a CSV file
 data <- read.csv("data.csv")
 # Read data from a tab-separated file (TSV)
 data <- read.table("data.tsv", sep = "\t", header = TRUE)
```
   iii. You can specify the delimiter and whether the file has a header row using parameters like `sep` and `header`.

b. **Reading from Excel Files:**
   i. R has packages like `readxl` and `openxlsx` for reading data from Excel files.
   ii. Example:
```
library(readxl)
 # Read data from an Excel file
 data <- read_excel("data.xlsx")
```

c. **Reading from Other Data Formats:**
   i. R supports a wide range of data formats. To read from formats like JSON, XML, HDF5, or databases, you'll need specific packages (e.g., `jsonlite`, `XML`, `hdf5r`, or database-specific packages like `DBI`).
   ii. Example (using `jsonlite` for JSON):
```
library(jsonlite)
data <- fromJSON("data.json")          # Read data from a JSON file
```

<u>Writing Data:</u>

a. **Writing to Text Files:**
   i. For other data formats (e.g., JSON, XML, HDF5), you'll need specific packages similar to reading.
   ii. Example (using `jsonlite` for JSON):
   ```
   library(jsonlite) # Write data to a JSON file
   toJSON(data, file = "output.json")
   ```

3. **<u>Function (setwd, getwd, data, rm etc):</u>**

In R, several functions and commands are used to manage the working directory, load datasets, and remove objects. Here's a detailed explanation of each of these functions and commands along with examples:

a. **`setwd()` (Set Working Directory):**
   i. `setwd()` is used to set the current working directory to a specified path. The working directory is the folder from which R reads and writes files by default.
   ii. Example:
   ```
   # Set the working directory to a specific path
   setwd("C:/Users/YourUserName/Documents/R_Projects")
   ```

b. **`getwd()` (Get Working Directory):**
   i. `getwd()` is used to retrieve the current working directory.
   ii. Example:
   ```
   # Get the current working directory
   current_dir <- getwd()
   print(current_dir)
   ```

c. **`data()` (Load Datasets):**
   i. `data()` is used to load built-in datasets that come with R packages. These datasets can be used for practice and learning.
   ii. Example:
   ```
   data(iris)       # Load the built-in 'iris' dataset
   head(iris)       # Use the 'iris' dataset in your R session
   ```

d. **`rm()` (Remove Objects):**

   i. `rm()` is used to remove objects (variables, data frames, etc.) from the R workspace. You can specify one or more objects to be removed.

   ii. Example:

```
# Create a variable 'x'
x <- c(1, 2, 3, 4, 5)

# Remove the variable 'x'
rm(x)

# Check if 'x' has been removed
print(exists("x"))
```

   You can also remove all objects in the workspace by using `rm(list = ls())`.

```
# Remove all objects in the workspace
rm(list = ls())
```

   iii. These functions and commands are useful for managing your R environment and working with data. Setting the working directory helps you read and write files from the correct location, `data()` makes it easy to access built-in datasets, and `rm()` is essential for cleaning up your workspace by removing unnecessary objects.

4. **Attaching and Detaching data:**

In R, you can attach and detach datasets to make it easier to work with the variables in those datasets. Attaching a dataset allows you to access its variables directly without specifying the dataset's name each time. Detaching a dataset removes it from the search path.

a. **Attaching Data:**

   i. To attach a dataset in R, you can use the `attach()` function. This function makes the dataset's variables available for direct access without specifying the dataset name.

   ii. Example:

```
# Create a simple dataset
my_data <- data.frame(
Name = c("Alice", "Bob", "Charlie"),
Age = c(25, 30, 22)
)
# Attach the dataset
attach(my_data)
# Now you can access variables directly
print(Name)
print(Age)
```

b. **Detaching Data:**

i. To detach a dataset, you can use the `detach()` function. Detaching a dataset removes it from the search path, making its variables inaccessible without specifying the data set's name.

ii. Example:

```
# Detach the dataset
detach(my_data)
# Attempting to access variables without specifying the dataset will result in an error.
# print(Name) # This will throw an error
```

iii. Note: It's important to be cautious when using `attach()` and `detach()`. While they can make code more concise, they can also lead to ambiguity and unexpected behavior, especially in larger and more complex scripts. It's often considered good practice to avoid using `attach()` and `detach()` in favor of using the `$` operator or the `with()` function for specific situations.

c. **Using `$` operator:**

i. You can access variables in a dataset directly using the `$` operator.

ii. Example:

```
# Create a dataset
my_data <- data.frame(
Name = c("Alice", "Bob", "Charlie"),
Age = c(25, 30, 22))
```

```
# Access variables directly using the $ operator
print(my_data$Name)
print(my_data$Age)
```

d. **Using `with()` Function:**
   i. The `with()` function allows you to temporarily work within a specific environment, making it easier to access variables.
   ii. Example:

```
# Create a dataset
my_data <- data.frame(
 Name = c("Alice", "Bob", "Charlie"),
 Age = c(25, 30, 22)
)
# Use with() to access variables
with(my_data, {
 print(Name)
 print(Age)
})
```

5. **Reading data from the console:**

Reading data from the console in R is a common task when you want to interactively input data while running your script or when you want to read user inputs during the execution of a program. You can use the `readLines()` and `scan()` functions to read data from the console. Here's a detailed explanation of how to do this:

a. **`readLines()` Function:**
   i. The `readLines()` function reads lines of text input from the console and stores them in a character vector. You can specify the number of lines to read or read until an empty line is encountered. It's commonly used for reading multiple lines of text.
   ii. Example:

```
# Read multiple lines of text until an empty line is encountered
input_lines <- readLines(con = "stdin", n = 0L)
```

```
print(input_lines)        # Print the input lines
```

iii. In the above example, `con = "stdin"` specifies that you are reading from the console, and `n = 0L` means to read until an empty line is entered. You can change the value of `n` to read a specific number of lines.

b. `scan()` Function:

i. The `scan()` function is used to read data elements separated by a specified delimiter (default is whitespace) from the console. It's commonly used for reading numeric or character data.

ii. Example:

```
# Read space-separated numeric values from the console
numeric_values <- scan(text = "", what = numeric())
# Print the numeric values
print(numeric_values)
```

iii. In this example, `text = ""` specifies that you are reading from the console, and `what = numeric()` indicates that you want to read numeric values. You can change `what` to `character()` if you want to read character data.

c. Reading Data Line by Line:

i. You can read data line by line by using a loop and `readLines()`. This is useful when you want to read and process multiple lines of input sequentially.

ii. Example:

```
# Initialize an empty vector to store lines of text
lines <- character(0)
# Read lines of text until an empty line is encountered
while (TRUE) {
line <- readLines(con = "stdin", n = 1L)
if (length(line) == 0) break # Exit the loop on an empty line
lines <- c(lines, line)
}
# Print the lines of text
print(lines)
```

Script (Code):

```
# CBinding Vector to DataFrame using Cbind
data_1 <- data.frame(x1 = c(7, 3, 2, 9, 0),          # Column1 of data frame 1
                x2 = c(4, 4, 1, 1, 8),               # Column2 of data frame 1
                x3 = c(5, 3, 9, 2, 4))               # Column3 of data frame 1
y1 <- c(9, 8, 7, 6, 5)                               # Create vector
data_new1 <- cbind(data_1, y1)                       # cbind vector to data frame
data_new1

# CBinding 2 dataframes using Cbind
data_2 <- data.frame(z1 = c(1, 5, 9, 4, 0),          # Column 1 of data frame 2
                z2 = c(0, 9, 8, 1, 6))               # Column 2 of data frame 2
data_new2 <- cbind(data_1, data_2)                   # cbind two data frames in R
data_new2

# R BINDING USING R
# RBINDING VECTOR TO DATAFRAME
x1 <- c(7, 4, 4, 9)                                  # Column 1 of data frame 1
x2 <- c(5, 2, 8, 9)                                  # Column 2 of data frame 1
x3 <- c(1, 2, 3, 4)                                  # Column 3 of data frame 1
data_1 <- data.frame(x1, x2, x3)                     # Create example data frame
vector_1 <- c(9, 8, 7)                               # Create example vector
rbind(data_1, vector_1)                              # rbind vector to data frame

# RBINDING 2 DATAFRAMES
x1 <- c(7, 1)                            # Column 1 of data frame 2
x2 <- c(4, 1)                            # Column 2 of data frame 2
x3 <- c(4, 3)                            # Column 3 of data frame 2
data_2 <- data.frame(x1, x2, x3)         # Create second data frame
rbind(data_1, data_2)                                # rbind two data frames in R

# Reading files using R ReadLines()
con <- file("F:/Pushkar/MCA/Sem-1/DAR/readnew.txt", "r")
```

```
w <- readLines(con)
close(con)
w[2]
w[3]
w[4]

# Writing files using R WriteLines()
sample <- c("Class,Alcohol,Malic acid,Ash","1,14.23,1.71,2.43","1,13.2,1.78,2.14")
writeLines(sample,"F:/Pushkar/MCA/Sem-1/DAR/sample.csv")
a <- read.csv("F:/Pushkar/MCA/Sem-1/DAR/sample.csv")
a

# Function in R (Setwd, getwd, data, rm)
setwd("D:/Users/Asus/Desktop/MCA FY/DataAnalytics with R")
getwd()
x <- runif(20)
summary(x)
hist(x)
list.files()                                    #Lists all the files in working directory
#q()                                            #Quit R. You'll get a prompt to save the
workspace.
ls()
# R program to illustrate
# attach function
# Create example data
txt <- data.frame(
   c1 = c(1, 2, 3, 4, 5),
   c2 = c(6, 7, 8, 9, 0),
   c3 = c(1, 2, 5, 4, 5))

# Try to print c1
c1
# Error: object 'c1' not found
# attach data
attach(txt)
c1
detach(txt)
c1

# Reading data from console in R
# Taking user input readline()
val <- readline(prompt = "Enter the number: ")

# Printing type of variable
print(paste("Old datatype: ",typeof(val)))

# Converting into integer type
val <- as.integer(val)

# Printing the type of variable
print(paste("New datatype: ",typeof(val)))

# Printing the variable
print(val)

# Reading first input using scan()
cat("Enter the number of rows: ")
nrows <- scan(n = 1, what = integer())

cat("Enter the number of columns: ")
ncols <- scan(n = 1, what = integer())

# Read matrix elements
cat("Enter the matrix elements:\n")
elements <- scan(n = nrows * ncols)
```

```
# Step 3: Reshape into a matrix
matrix_data <- matrix(elements, nrow = nrows, ncol = ncols)
```

Console (Output):

```
> # C BINDING USING R
> # CBinding Vector to DataFrame using Cbind
> data_1 <- data.frame(x1 = c(7, 3, 2, 9, 0),          # Column1 of data frame 1
+                  x2 = c(4, 4, 1, 1, 8),              # Column2 of data frame 1
+                  x3 = c(5, 3, 9, 2, 4))              # Column3 of data frame 1
> y1 <- c(9, 8, 7, 6, 5)                               # Create vector
> data_new1 <- cbind(data_1, y1)                       # cbind vector to data frame
> data_new1
  x1 x2 x3 y1
1  7  4  5  9
2  3  4  3  8
3  2  1  9  7
4  9  1  2  6
5  0  8  4  5
>
> # CBinding 2 dataframes using Cbind
> data_2 <- data.frame(z1 = c(1, 5, 9, 4, 0),          # Column 1 of data frame 2
               + z2 = c(0, 9, 8, 1, 6))                # Column 2 of data frame 2
> data_new2 <- cbind(data_1, data_2)              # cbind two data frames in R
> data_new2
  x1 x2 x3 z1 z2
1  7  4  5  1  0
2  3  4  3  5  9
3  2  1  9  9  8
4  9  1  2  4  1
5  0  8  4  0  6
>
> # R BINDING USING R
> # RBINDING VECTOR TO DATA FRAME
> x1 <- c(7, 4, 4, 9)                          # Column 1 of data frame 1
> x2 <- c(5, 2, 8, 9)                          # Column 2 of data frame 1
> x3 <- c(1, 2, 3, 4)                          # Column 3 of data frame 1
> data_1 <- data.frame(x1, x2, x3)             # Create example data frame
> vector_1 <- c(9, 8, 7)                       # Create example vector
> rbind(data_1, vector_1)                      # rbind vector to data frame
  x1 x2 x3
1  7  5  1
2  4  2  2
3  4  8  3
4  9  9  4
5  9  8  7
>
> # RBINDING 2 DATAFRAMES
> x1 <- c(7, 1)                                # Column 1 of data frame 2
> x2 <- c(4, 1)                                # Column 2 of data frame 2
> x3 <- c(4, 3)                                # Column 3 of data frame 2
> data_2 <- data.frame(x1, x2, x3)             # Create second data frame
> rbind(data_1, data_2)                        # rbind two data frames in R
  x1 x2 x3
1  7  5  1
2  4  2  2
3  4  8  3
4  9  9  4
5  7  4  4
6  1  1  3
>
> # Reading files using R ReadLines()
> con <- file("F:/Pushkar/MCA/Sem-1/DAR/readnew.txt", "r")
> w <- readLines(con)
> close(con)
> w[2]
[1] "This is the second line."
```

```
> w[3]
[1] "This is the third line."
> w[4]
[1] "This is the forth line."
>
> # Writing files using R WriteLines()
> sample <- c("Class,Alcohol,Malic acid,Ash","1,14.23,1.71,2.43","1,13.2,1.78,2.14")
> writeLines(sample,"F:/Pushkar/MCA/Sem-1/DAR/sample.csv")
> a <- read.csv("F:/Pushkar/MCA/Sem-1/DAR/sample.csv")
> a
  Class Alcohol Malic.acid  Ash
1     1   14.23       1.71 2.43
2     1   13.20       1.78 2.14
>
> # Function in R (Setwd, getwd, data, rm)
> setwd("F:/Pushkar/MCA/Sem-1/DAR")
> getwd()
[1] "F:/Pushkar/MCA/Sem-1/DAR"
> x <- runif(20)
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.1463  0.4837  0.6256  0.6125  0.8105  0.9854
> hist(x)
> list.files()                                    #Lists all the files in working directory
[1]  "Journal"          "Practical  1.R"    "Practical  2.R"    "readnew.txt"       "SalesData.xlsx"
"SalesData1.csv"
[7] "sample.csv"
> # q()                               #Quit R. You'll get a prompt to save the workspace.
> ls()
 [1] "a"          "b"          "con"        "data_1"     "data_2"     "data_new1"  "data_new2"  "df"
"difference"
[10] "elements"   "gender"     "is_valid"   "mat"        "my_list"    "name"       "ncols"      "nrows"
"nums"
```

```
[19]  "power"        "product"     "quotient"    "remainder"   "SalesData"   "SalesData1" "sample"
"sum_result" "today"
[28] "txt"        "val"       "vector_1"   "w"         "x"         "x1"        "x2"        "x3"        "y"
[37] "y1"        "z"
>
> # R program to illustrate
> # Attach function
> # Create example data
> txt <- data.frame(
+     c1 = c(1, 2, 3, 4, 5),
+     c2 = c(6, 7, 8, 9, 0),
+     c3 = c(1, 2, 5, 4, 5))
>
> # Try to print c1
> c1
Error: object 'c1' not found
> # Error: object 'c1' not found
> # Attach data
> attach(txt)
> c1
[1] 1 2 3 4 5
> detach(txt)
> c1
Error: object 'c1' not found
> # Reading data from console in R
> # Taking user input readline()
> val <- readline(prompt = "Enter the number: ")
Enter the number: 5
> # Printing type of variable
> print(paste("Old datatype: ",typeof(val)))
[1] "Old datatype:  character"
>
> # Converting into integer type
> val <- as.integer(val)
>
> # Printing the type of variable
> print(paste("New datatype: ",typeof(val)))
[1] "New datatype:  integer"
>
> # Printing the variable
> print(val)
[1] 5
> # Reading first input using scan()
> cat("Enter the number of rows: ")
Enter the number of rows: > nrows <- scan(n = 1, what = integer())
1: 3
Read 1 item
> cat("Enter the number of columns: ")
Enter the number of columns: > ncols <- scan(n = 1, what = integer())
1: 3
Read 1 item
> # Read matrix elements
> cat("Enter the matrix elements:\n")
Enter the matrix elements:
> elements <- scan(n = nrows * ncols)
1: 2
2: 3
3: 4
4: 5
5: 6
6: 7
7: 8
8: 9
9: 2
Read 9 items
> # Step 4: Display the matrix
> print(matrix_data)
     [,1] [,2] [,3]
[1,]    2    5   89
[2,]    3    6    9
[3,]    4    7    8
>
```

Conclusion: In this practical we learned different commands and functions. These commands and functions are fundamental for data analysis and manipulation in R. Properly managing the working directory, loading data, writing results, and handling data objects are crucial skills for any R user which we learned during the practical.