1. Implement data frames in R. Write a program to join columns and rows in a data frame using c bind()and r bind() in R.

R-CODE:

```
df1 <- data.frame(ID = c(1, 2, 3), Name = c("A", "B", "C"))
df2 <- data.frame(Age = c(25, 30, 28), Score = c(90, 85, 88))

df_col_bind <- cbind(df1, df2)
print(df_col_bind)

df3 <- data.frame(ID = c(4, 5), Name = c("D", "E"), Age = c(26, 29), Score = c(92, 87))
df_row_bind <- rbind(df_col_bind, df3)
print(df_row_bind)
```

OUTPUT:

```
> print(df_col_bind)
  ID Name Age Score
1  1    A  25    90
2  2    B  30    85
3  3    C  28    88
> df3 <- data.frame(ID = c(4, 5), Name = c("D", "E"), Age = c(26, 29), Score = c(92, 87))
> df_row_bind <- rbind(df_col_bind, df3)
> print(df_row_bind)
  ID Name Age Score
1  1    A  25    90
2  2    B  30    85
3  3    C  28    88
4  4    D  26    92
5  5    E  29    87
>
```

2. Implement different String Manipulation functions in R.

R-CODE:

```
text <- "Hello R Programming"

substr_result <- substr(text, 1, 5)
print(substr_result)

tolower_result <- tolower(text)
print(tolower_result)

toupper_result <- toupper(text)
print(toupper_result)

nchar_result <- nchar(text)
print(nchar_result)

paste_result <- paste("Welcome", "to", "R", "Programming", sep = " ")
print(paste_result)

gsub_result <- gsub("R", "Python", text)
print(gsub_result)

split_result <- strsplit(te
```

OUTPUT:

```
R 4.4.2 · ~/
> text <- "Hello R Programming"
> substr_result <- substr(text, 1, 5)
> print(substr_result)
[1] "Hello"
> tolower_result <- tolower(text)
> print(tolower_result)
[1] "hello r programming"
> toupper_result <- toupper(text)
> print(toupper_result)
[1] "HELLO R PROGRAMMING"
> nchar_result <- nchar(text)
> print(nchar_result)
[1] 19
> paste_result <- paste("Welcome", "to", "R", "Programming", sep = " ")
> print(paste_result)
[1] "Welcome to R Programming"
> gsub_result <- gsub("R", "Python", text)
> print(gsub_result)
[1] "Hello Python Programming"
> split_result <- strsplit(te
+
```

3. Write R program to find Correlation and Covariance and Write R program for Regression Modeling.

R-CODE:

```
Source on Save
1  data(mtcars)
2
3  correlation_matrix <- cor(mtcars)
4  print(correlation_matrix)
5
6  covariance_matrix <- cov(mtcars)
7  print(covariance_matrix)
8
9  model <- lm(mpg ~ hp + wt, data = mtcars)
10 summary(model)
11
12 new_data <- data.frame(hp = c(100, 150), wt = c(2.5, 3.0))
13 predictions <- predict(model, new_data)
14 print(predictions)
15
16
```

print(x, ...)

OUTPUT:

```
                am         gear         carb
mpg     1.80393145   2.1356855 -5.36310484
cyl    -0.46572581  -0.6491935  1.52016129
disp -36.56401210 -50.8026210 79.06875000
hp     -8.32056452  -6.3588710 83.03629032
drat    0.19015121   0.2759879 -0.07840726
wt     -0.33810484  -0.4210806  0.67579032
qsec   -0.20495968  -0.2804032 -1.89411290
vs      0.04233871   0.0766129 -0.46370968
am      0.24899194   0.2923387  0.04637097
gear    0.29233871   0.5443548  0.32661290
carb    0.04637097   0.3266129  2.60887097
> model <- lm(mpg ~ hp + wt, data = mtcars)
> summary(model)

Call:
lm(formula = mpg ~ hp + wt, data = mtcars)

Residuals:
   Min     1Q Median     3Q    Max
-3.941 -1.600 -0.182  1.050  5.854

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 37.22727    1.59879  23.285  < 2e-16 ***
hp          -0.03177    0.00903  -3.519  0.00145 **
wt          -3.87783    0.63273  -6.129 1.12e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.593 on 29 degrees of freedom
Multiple R-squared:  0.8268,    Adjusted R-squared:  0.8148
F-statistic: 69.21 on 2 and 29 DF,  p-value: 9.109e-12

> new_data <- data.frame(hp = c(100, 150), wt = c(2.5, 3.0))
> predictions <- predict(model, new_data)
> print(predictions)
       1        2
24.35540 20.82784
>
```

4. Write R program to build classification model using KNN algorithm

R-CODE:

```
library(class)

data(iris)

set.seed(123)
index <- sample(1:nrow(iris), 0.8 * nrow(iris))
train_data <- iris[index, ]
test_data <- iris[-index, ]

train_labels <- train_data$Species
test_labels <- test_data$Species

train_features <- train_data[, -5]
test_features <- test_data[, -5]

k <- 5
predictions <- knn(train = train_features, test = test_features, cl = train_labels, k = k)

accuracy <- sum(predictions == test_labels) / length(test_labels)
print(accuracy)
```

OUTPUT:

```
R 4.4.2 · ~/
> library(class)
> data(iris)
> set.seed(123)
> index <- sample(1:nrow(iris), 0.8 * nrow(iris))
> train_data <- iris[index, ]
> test_data <- iris[-index, ]
> train_labels <- train_data$Species
> test_labels <- test_data$Species
> train_features <- train_data[, -5]
> test_features <- test_data[, -5]
> k <- 5
> predictions <- knn(train = train_features, test = test_features, cl = train_labels, k = k)
> accuracy <- sum(predictions == test_labels) / length(test_labels)
> print(accuracy)
[1] 0.9666667
>
```

5. Write R program to build clustering model using K-mean algorithm.

R-CODE:

```
data(iris)

features <- iris[, -5]

set.seed(123)
kmeans_model <- kmeans(features, centers = 3, nstart = 25)

print(kmeans_model$centers)
print(table(kmeans_model$cluster, iris$Species))
```

OUTPUT:

```
> data(iris)
> features <- iris[, -5]
> set.seed(123)
> kmeans_model <- kmeans(features, centers = 3, nstart = 25)
> print(kmeans_model$centers)
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.006000    3.428000     1.462000    0.246000
2     5.901613    2.748387     4.393548    1.433871
3     6.850000    3.073684     5.742105    2.071053
> print(table(kmeans_model$cluster, iris$Species))

    setosa versicolor virginica
  1     50          0         0
  2      0         48        14
  3      0          2        36
>
```

6. Write a R program to create three vectors numeric data, character data and logical data. Display the content of the vectors and their type.

R-CODE:

```
num_vector <- c(10, 20, 30, 40, 50)
char_vector <- c("A", "B", "C", "D", "E")
log_vector <- c(TRUE, FALSE, TRUE, FALSE, TRUE)

print(num_vector)
print(typeof(num_vector))

print(char_vector)
print(typeof(char_vector))

print(log_vector)
print(typeof(log_vector))
```

OUTPUT:

```
Console  Terminal  Background Jobs
R 4.4.2 · ~/
> num_vector <- c(10, 20, 30, 40, 50)
> char_vector <- c("A", "B", "C", "D", "E")
> log_vector <- c(TRUE, FALSE, TRUE, FALSE, TRUE)
> print(num_vector)
[1] 10 20 30 40 50
> print(typeof(num_vector))
[1] "double"
> print(char_vector)
[1] "A" "B" "C" "D" "E"
> print(typeof(char_vector))
[1] "character"
> print(log_vector)
[1]  TRUE FALSE  TRUE FALSE  TRUE
> print(typeof(log_vector))
[1] "logical"
>
```

7. Write a R program to create a 5 x 4 matrix , 3 x 3 matrix with labels and fill the matrix by rows and 2 × 2 matrix with labels and fill the matrix by columns.
   R-CODE:

```
matrix_5x4 <- matrix(1:20, nrow = 5, ncol = 4, byrow = TRUE)
print(matrix_5x4)

matrix_3x3 <- matrix(1:9, nrow = 3, ncol = 3, byrow = TRUE, dimnames = list(c("R1", "R2", "R3"), c("C1", "C2",
print(matrix_3x3)

matrix_2x2 <- matrix(1:4, nrow = 2, ncol = 2, byrow = FALSE, dimnames = list(c("R1", "R2"), c("C1", "C2")))
print(matrix_2x2)
```

   OUTPUT:

```
> matrix_5x4 <- matrix(1:20, nrow = 5, ncol = 4, byrow = TRUE)
> print(matrix_5x4)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
[4,]   13   14   15   16
[5,]   17   18   19   20
> matrix_3x3 <- matrix(1:9, nrow = 3, ncol = 3, byrow = TRUE, dimnames = list(c("R1", "R2", "R3"), c("C1", "C2", "C
3")))
> print(matrix_3x3)
   C1 C2 C3
R1  1  2  3
R2  4  5  6
R3  7  8  9
> matrix_2x2 <- matrix(1:4, nrow = 2, ncol = 2, byrow = FALSE, dimnames = list(c("R1", "R2"), c("C1", "C2")))
> print(matrix_2x2)
   C1 C2
R1  1  3
R2  2  4
>
```

8. Write a R program to create an array, passing in a vector of values and a vector of dimensions. Also provide names for each dimension.
   R-CODE:

```
values <- 1:12
dimensions <- c(3, 2, 2)

array_data <- array(values, dim = dimensions, dimnames = list(c("Row1", "Row2", "Row3"), c("Col1", "Col2"), c(

print(array)
```

   OUTPUT:

```
> array_data <- array(values, dim = dimensions, dimnames = list
le1", "Table2")))
> print(array)
function (data = NA, dim = length(data), dimnames = NULL)
{
    if (is.atomic(data) && !is.object(data))
        return(.Internal(array(data, dim, dimnames)))
    data <- as.vector(data)
    if (is.object(data)) {
        dim <- as.integer(dim)
        if (!length(dim))
            stop("'dim' cannot be of length 0")
        vl <- prod(dim)
        if (length(data) != vl) {
            if (vl > .Machine$integer.max)
                stop("'dim' specifies too large an array")
            data <- rep_len(data, vl)
        }
        if (length(dim))
            dim(data) <- dim
        if (is.list(dimnames) && length(dimnames))
            dimnames(data) <- dimnames
        data
    }
    else .Internal(array(data, dim, dimnames))
}
<bytecode: 0x000001fe0c6c5400>
<environment: namespace:base>
>
>
```

9. .Write a R program to create an array with three columns, three rows, and two "tables", taking two vectors as input to the array. Print the array.

   R-CODE:

```
Source on Save
1  vector1 <- c(1:9)
2  vector2 <- c(10:18)
3
4  array_data <- array(c(vector1, vector2), dim = c(3, 3, 2))
5
6  print(array_data)
7
8
```

   OUTPUT:

```
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

, , 2

     [,1] [,2] [,3]
[1,]   10   13   16
[2,]   11   14   17
[3,]   12   15   18

>
```

10. .Write a R program to draw an empty plot and an empty plot specify the axes limits of the graphic

R-CODE:

```r
plot(1, type = "n")

plot(1, type = "n", xlim = c(0, 10), ylim = c(0, 10))
```

OUTPUT: