1.a

Importing Data from a CSV File:

R

```
# Set the working directory to the folder where your CSV file is located
setwd("your_directory_path")

# Read data from a CSV file
data <- read.csv("your_file.csv")

# Display the structure of the imported data
str(data)

# View the first few rows of the data
head(data)
```

Replace "your_directory_path" with the path to the folder where your CSV file is located, and "your_file.csv" with the name of your CSV file.

Exporting Data to a CSV File:

R

```
# Assume 'data' is your data frame that you want to export

# Write data to a CSV file
write.csv(data, file = "output_file.csv", row.names = FALSE)
```

1.b

```
# Create a sample matrix
matrix_data <- matrix(c(1, 5, 3, 8, 2, 7, 4, 6, 9), nrow = 3, byrow = TRUE)

# Display the matrix
cat("Original Matrix:\n")
print(matrix_data)

# Find the row and column indices of the maximum value
max_value <- max(matrix_data)
max_index <- which(matrix_data == max_value, arr.ind = TRUE)

cat("\nMaximum Value and Indices:\n")
cat("Value:", max_value, "\n")
cat("Row Index:", max_index[1, 1], "\n")
cat("Column Index:", max_index[1, 2], "\n")

# Find the row and column indices of the minimum value
min_value <- min(matrix_data)
min_index <- which(matrix_data == min_value, arr.ind = TRUE)

cat("\nMinimum Value and Indices:\n")
cat("Value:", min_value, "\n")
cat("Row Index:", min_index[1, 1], "\n")
cat("Column Index:", min_index[1, 2], "\n")
```

2.a

```
# Generate some sample data
set.seed(123)
data <- rnorm(100)

# Calculate statistical measures
mean_value <- mean(data)
median_value <- median(data)
range_value <- range(data)[2] - range(data)[1]

# Create a boxplot
boxplot(data, main = "Statistical Measures Visualization", ylab = "Values")

# Add mean, median, and range to the boxplot
abline(h = mean_value, col = "red", lty = 2, lwd = 2, label = "Mean")
abline(h = median_value, col = "blue", lty = 2, lwd = 2, label = "Median")
abline(h = range(data)[1], col = "green", lty = 2, lwd = 2, label = "Min")
abline(h = range(data)[2], col = "purple", lty = 2, lwd = 2, label = "Max")

# Add legend
legend("topright", legend = c("Mean", "Median", "Min", "Max"), col = c("red", "blue",
"green", "purple"), lty = 2, lwd = 2)

# Print statistical measures
cat("Mean:", mean_value, "\n")
cat("Median:", median_value, "\n")
cat("Range:", range_value, "\n")
```

2.b

```
# Create a sample data frame
data_frame <- data.frame(
  Name = c("John", "Jane", "Bob", "Alice"),
  Age = c(25, 30, 22, 28),
  Score = c(90, 85, 78, 92)
)

# Display the original data frame
cat("Original Data Frame:\n")
print(data_frame)

# Save the data frame to a CSV file
write.csv(data_frame, file = "output_file.csv", row.names = FALSE)

# Display the information of the saved file
saved_data <- read.csv("output_file.csv")
cat("\nData Frame from Saved File:\n")
print(saved_data)
```

3.a

```
# Generate sample data
set.seed(123)
data <- data.frame(
  Value1 = rnorm(100, mean = 0, sd = 1),
  Value2 = rnorm(100, mean = 5, sd = 2)
)

# Histogram
par(mfrow = c(2, 2))  # Set up a 2x2 layout for multiple plots
hist(data$Value1, main = "Histogram for Value1", xlab = "Value1", col = "lightblue", border = "black")
hist(data$Value2, main = "Histogram for Value2", xlab = "Value2", col = "lightgreen", border = "black")

# Boxplot
boxplot(data, main = "Boxplot for Value1 and Value2", col = c("lightblue", "lightgreen"), names = c("Value1", "Value2"))

# Scatter Plot
plot(data$Value1, data$Value2, main = "Scatter Plot", xlab = "Value1", ylab = "Value2", col = "blue", pch = 16)

# Add a regression line to the scatter plot
abline(lm(data$Value2 ~ data$Value1), col = "red")

# Reset the layout to default
par(mfrow = c(1, 1))
```

3.b

```
# Create sample data
data <- matrix(1:24, nrow = 3, ncol = 4)

# Create a 3D array using the array() function
my_array <- array(data, dim = c(3, 4, 2))

# Display the content of the array
cat("Content of the 3D Array:\n")
print(my_array)
```

4.a

```
# Create a data frame
my_data <- data.frame(
  Name = c("John", "Jane", "Bob", "Alice"),
  Age = c(25, 30, 22, 28),
  Score = c(90, 85, 78, 92),
  Grade = c("A", "B", "C", "A")
)

# Display the data frame
cat("Original Data Frame:\n")
print(my_data)

# a. Extract two column names using column name.
selected_columns <- colnames(my_data)[1:2]
cat("\nSelected Columns:\n")
print(selected_columns)

# b. Extract the first two rows and then all columns
first_two_rows_all_columns <- my_data[1:2, ]
cat("\nFirst Two Rows and All Columns:\n")
print(first_two_rows_all_columns)

# c. Extract 3rd & 5th row with 2nd and 4th column
selected_rows_columns <- my_data[c(3, 5), c(2, 4)]
cat("\nSelected Rows and Columns:\n")
print(selected_rows_columns)
```

4.b

```
# Create two sample matrices
matrix1 <- matrix(1:6, nrow = 2, ncol = 3)
matrix2 <- matrix(7:12, nrow = 2, ncol = 3)

# Display the original matrices
cat("Matrix 1:\n")
print(matrix1)

cat("\nMatrix 2:\n")
print(matrix2)

# Concatenate the matrices vertically
concatenated_matrix <- rbind(matrix1, matrix2)

# Display the concatenated matrix
cat("\nConcatenated Matrix:\n")
print(concatenated_matrix)
```

5.a

```
# Load the iris dataset
data(iris)

# Function to perform min-max normalization
min_max_normalize <- function(x) {
  (x - min(x)) / (max(x) - min(x))
}

# Apply min-max normalization to each numeric variable/column of the iris dataset
iris_normalized <- as.data.frame(apply(iris[, sapply(iris, is.numeric)], 2,
min_max_normalize))

# Display the original and normalized datasets
cat("Original Iris Dataset:\n")
print(head(iris))

cat("\nNormalized Iris Dataset (Min-Max Normalization):\n")
print(head(iris_normalized))
```

5.b

```
# Create a 5 x 4 matrix filled by rows
matrix_by_rows <- matrix(1:20, nrow = 5, ncol = 4, byrow = TRUE)

# Create a 3 x 3 matrix with labels
matrix_with_labels <- matrix(21:29, nrow = 3, ncol = 3)
rownames(matrix_with_labels) <- c("Row1", "Row2", "Row3")
colnames(matrix_with_labels) <- c("Col1", "Col2", "Col3")

# Create a 2 x 2 matrix filled by columns
matrix_by_columns <- matrix(30:33, nrow = 2, ncol = 2, byrow = FALSE)

# Display the matrices
cat("Matrix 1 (5 x 4 filled by rows):\n")
print(matrix_by_rows)

cat("\nMatrix 2 (3 x 3 with labels):\n")
print(matrix_with_labels)

cat("\nMatrix 3 (2 x 2 filled by columns):\n")
print(matrix_by_columns)
```

6.a

```
# Create a data frame with 10 observations and 3 variables
original_data <- data.frame(
  Name = c("John", "Jane", "Bob", "Alice", "Charlie", "David", "Emma", "Frank",
"Grace", "Henry"),
  Age = c(25, 30, 22, 28, 35, 40, 24, 32, 27, 29),
  Score = c(90, 85, 78, 92, 88, 76, 95, 82, 89, 93)
)

# Display the original data frame
cat("Original Data Frame:\n")
print(original_data)

# Add a new row using rbind
new_row <- c("Isaac", 26, 87)
data_with_new_row <- rbind(original_data, new_row)

# Display the data frame with the new row
cat("\nData Frame with New Row:\n")
print(data_with_new_row)

# Add a new column using cbind
new_column <- c("Male", "Female", "Male", "Female", "Male", "Male", "Female",
"Male", "Female", "Male")
data_with_new_column <- cbind(data_with_new_row, Gender = new_column)

# Display the data frame with the new column
cat("\nData Frame with New Column:\n")
print(data_with_new_column)
```

6.b

```
# Create two vectors
vector1 <- c("John", "Jane", "Bob", "Alice", "Charlie", "David", "Emma", "Frank",
"Grace", "John")
vector2 <- c(25, 30, 22, 28, 35, 40, 24, 32, 27, 25)

# Create a data frame using the two vectors
my_data_frame <- data.frame(Name = vector1, Age = vector2)

# Display the original data frame
cat("Original Data Frame:\n")
print(my_data_frame)

# Find duplicated elements in the data frame
duplicated_elements <- my_data_frame[duplicated(my_data_frame) |
duplicated(my_data_frame, fromLast = TRUE), ]

cat("\nDuplicated Elements:\n")
print(duplicated_elements)
```

```
# Find unique rows in the data frame
unique_rows <- unique(my_data_frame)

cat("\nUnique Rows:\n")
print(unique_rows)
```

7.a

```
# Generate sample data
set.seed(123)
data <- matrix(rnorm(100 * 2), ncol = 2)

# Perform K-means clustering with K=3
k <- 3
kmeans_result <- kmeans(data, centers = k)

# Display the cluster assignments
cat("Cluster Assignments:\n")
print(kmeans_result$cluster)

# Display the cluster centers
cat("\nCluster Centers:\n")
print(kmeans_result$centers)

# Plot the data points with cluster assignments
plot(data, col = kmeans_result$cluster, pch = 19, main = "K-Means Clustering", xlab = "X-
axis", ylab = "Y-axis")

# Plot the cluster centers
points(kmeans_result$centers, col = 1:k, pch = 8, cex = 2, lwd = 2)
```

7.b

```
# Create a vector of month names
month_names <- c("January", "February", "March", "April", "May", "June", "July",
"August", "September", "October", "November", "December")

# Create an ordered factor from the month names
ordered_months <- factor(month_names, ordered = TRUE, levels = month.name)

# Display the ordered factor
cat("Ordered Factor of Months:\n")
print(ordered_months)
```

8.a

```
# Install and load the 'cluster' package
# install.packages("cluster")
library(cluster)

# Generate sample data
set.seed(123)
data <- matrix(rnorm(100 * 2), ncol = 2)

# Perform K-medoids clustering with K=3
k <- 3
kmedoids_result <- pam(data, k)

# Display the cluster assignments
cat("Cluster Assignments:\n")
print(kmedoids_result$clustering)

# Display the medoids (representative objects in each cluster)
cat("\nMedoids:\n")
print(kmedoids_result$medoids)

# Plot the data points with cluster assignments
plot(data, col = kmedoids_result$clustering, pch = 19, main = "K-Medoids Clustering", xlab =
"X-axis", ylab = "Y-axis")

# Plot the medoids
points(kmedoids_result$medoids, col = 1:k, pch = 8, cex = 2, lwd = 2)
```

8.b

```
# Create a vector with factor levels
sample_vector <- factor(c("High", "Low", "Medium", "High", "Low", "Medium", "Low",
"High"))

# Find the levels of the factor
factor_levels <- levels(sample_vector)

# Display the factor levels
cat("Factor Levels:\n")
print(factor_levels)
```

9.a

```
# Install and load the 'dbscan' package
# install.packages("dbscan")
library(dbscan)

# Load the iris dataset
data(iris)

# Select two features for simplicity (you can use all features if needed)
selected_features <- iris[, c("Sepal.Length", "Sepal.Width")]

# Standardize the selected features
scaled_features <- scale(selected_features)

# Perform DBSCAN clustering
dbscan_result <- dbscan(scaled_features, eps = 0.5, minPts = 5)

# Display the cluster assignments
cat("Cluster Assignments:\n")
print(dbscan_result$cluster)

# Plot the data points with cluster assignments
plot(scaled_features, col = dbscan_result$cluster + 1, pch = 19, main = "DBSCAN
Clustering", xlab = "Sepal.Length", ylab = "Sepal.Width")

# Highlight noise points (cluster 0)
points(scaled_features[dbscan_result$cluster == 0,], col = "red", pch = 3, cex = 2, lwd = 2)
```

9.b

```
# Create two factors
factor1 <- factor(c("A", "B", "C"))
factor2 <- factor(c("X", "Y", "Z"))

# Concatenate the two factors
concatenated_factor <- factor(c(factor1, factor2))

# Display the original factors
cat("Factor 1:\n")
print(factor1)

cat("\nFactor 2:\n")
print(factor2)

# Display the concatenated factor
cat("\nConcatenated Factor:\n")
print(concatenated_factor)
```