Lab Exercises:

Exercise 1: **Basic Arithmetic Operations**

You are working as a cashier at a grocery store. Your task is to create a program that simulates the checkout process for a customer's shopping cart. The program should calculate the total cost of the items, including tax, and provide a detailed receipt.

i. Define a list of products, each represented as a dictionary with keys: "name", "price", and "quantity".

ii. Allow the cashier to input the products in the customer's shopping cart, including the name, price, and quantity of each item.

iii. Calculate the subtotal (price * quantity) for each item and display a detailed receipt with product names, quantities, prices, and subtotals.

iv. Calculate the total cost of the items in the cart before tax.

v. Apply a tax rate (e.g., 8%) to the total cost to calculate the tax amount.

vi. Calculate the final total cost by adding the tax amount to the total cost before tax.

Code in R: # List of products

```
# List of products
products <- list(
  list(name = "Apple", price = 0.5),
  list(name = "Banana", price = 0.3),
  list(name = "Milk", price = 2),
  list(name = "Bread", price = 1.5),
  list(name = "Eggs", price = 2.5)
)


# Initialize shopping cart as an empty list
shopping_cart <- list()


# Define items to be added to the cart
cart_items_to_add <- list(
  list(name = "Apple", quantity = 3),
  list(name = "Milk", quantity = 2)
)
```

```
# Add items to the shopping cart
for (item in cart_items_to_add) {
  product_name <- item$name
  quantity <- item$quantity

  # Find the product in the list
  product <- NULL
  for (p in products) {
    if (p$name == product_name) {
      product <- p
      break
    }
  }

  if (!is.null(product)) {
    cart_item <- list(name = product$name, price = product$price, quantity = quantity)
    shopping_cart <- c(shopping_cart, list(cart_item))
    cat("Item added to cart.\n")
  } else {
    cat("Product not found.\n")
  }
}

# Calculate and display receipt
subtotal <- 0
cat("\nReceipt:\n")
for (item in shopping_cart) {
  item_subtotal <- item$price * item$quantity
  cat(sprintf("%s (%d units) - Price: $%.2f - Subtotal: $%.2f\n", item$name, item$quantity,
item$price, item_subtotal))
  subtotal <- subtotal + item_subtotal
}

tax_rate <- 0.08
```

```
tax_amount <- subtotal * tax_rate

total_cost_before_tax <- subtotal

total_cost <- total_cost_before_tax + tax_amount


cat("\nSubtotal: $%.2f\n", subtotal)

cat("Tax Amount (8%): $%.2f\n", tax_amount)

cat("Total Cost: $%.2f\n", total_cost)
```

Exercise 2: **Loops Operations**

You have been tasked with creating a program that calculates and assigns grades for students enrolled in multiple courses. The program will take input for the marks obtained by 10 students in 5 different courses, compute the total and average marks for each student, and assign corresponding grades based on their average performance.

Declare constants for the number of students (num_students) and the number of courses (num_courses).

Initialize an empty list to store student information.

For each student:

- Input the student's name.
- Input marks for each of the 5 courses.
- Calculate the total marks and average marks.
- Determine the grade based on the average marks using a grading scale.
- Display the student information, including their name, individual course marks, total marks, average marks, and the assigned grade.

Program:

```
# Constants

num_students <- 5

num_courses <- 5


# Predefined student names

student_names <- c("Arun Rahul", "Bheem Kumar","Raj jumar","Jahal  A R","Suresh")


# Predefined course marks for each student

course_marks <- matrix(c(

  85, 92, 78, 88, 95,
```

```
  75, 80, 85, 70, 60,
  100,78,56,34,56,
  78,45,67,89,90,
  89,80,67,78,90
), nrow = num_students, byrow = TRUE)

# Initialize a list to store student information
student_records <- list()

# Loop for each student
for (student_index in 1:num_students) {
  student_name <- student_names[student_index]

  # Initialize variables for calculations
  total_marks <- sum(course_marks[student_index, ])
  average_marks <- total_marks / num_courses

  # Determine grade based on average marks
  grade <- ifelse(average_marks >= 90, "A",
                  ifelse(average_marks >= 80, "B",
                         ifelse(average_marks >= 70, "C",
                                ifelse(average_marks >= 60, "D", "F"))))

  # Store student information in a record
  student_record <- list(name = student_name, marks = course_marks[student_index, ],
                         total = total_marks, average = average_marks, grade = grade)

  student_records <- c(student_records, list(student_record))
}

# Display student information
cat("\nStudent Grade Report:\n")
for (student_record in student_records) {
```

```
  cat("\nName:", student_record$name, "\n")
  cat("Marks:", student_record$marks, "\n")
  cat("Total Marks:", student_record$total, "\n")
  cat("Average Marks:", student_record$average, "\n")
  cat("Grade:", student_record$grade, "\n")
}
```

Output of the program:

Student Grade Report:

Name: Arun Rahul

Marks: 85 92 78 88 95

Total Marks: 438

Average Marks: 87.6

Grade: B

Name: Bheem Kumar

Marks: 75 80 85 70 60

Total Marks: 370

Average Marks: 74

Grade: C

Name: Raj jumar

Marks: 100 78 56 34 56

Total Marks: 324

Average Marks: 64.8

Grade: D

Name: Jahal A R

Marks: 78 45 67 89 90

Total Marks: 369

Average Marks: 73.8

Grade: C

Name: Suresh

Marks: 89 80 67 78 90

Total Marks: 404

Average Marks: 80.8

Grade: B

Exercise 3: Conditional statement, Loops and Functions:

You are developing a library management system that needs a **fine calculation feature.** Write a program that takes the number of days a book is overdue and calculates the fine amount based on the library's policy. The policy states that for the first 7 days, there is no fine. After 7 days, a fixed fine per day is charged. Additionally, there's a cap on the fine amount after 30 days.

Input the number of days the book is overdue.

- Use conditional statements to calculate the fine amount based on the library's policy.
- Display the fine amount along with a message indicating whether the fine is within the cap or exceeded it.

Program:

```
# Function to calculate library fine
calculate_fine <- function(days_overdue) {
  if (days_overdue <= 7) {
    fine <- 0  # No fine for the first 7 days
  } else if (days_overdue <= 30) {
    fine_per_day <- 2  # Fine per day after 7 days
    fine <- (days_overdue - 7) * fine_per_day
  } else {
    fine_cap <- 50  # Maximum fine after 30 days
    fine <- fine_cap
  }
  return(fine)
}

# Input number of days overdue
```

```
days_overdue <- as.integer(readline("Enter the number of days the book is overdue: "))


# Calculate fine
fine_amount <- calculate_fine(days_overdue)


# Display fine information
cat("Fine Amount:", fine_amount, "\n")
if (fine_amount == 0) {
  cat("No fine. Thank you for returning the book on time!\n")
} else {
  if (days_overdue > 30) {
    cat("Fine exceeds the maximum cap. Please contact the library.\n")
  } else {
    cat("Please pay the fine within the specified period.\n")
  }
}
```

Output:

```
Enter the number of days the book is overdue: 20

Fine Amount: 26

Please pay the fine within the specified period.
```

Exercise 4: **arrays and Functions:**

You are developing an inventory management system for a small store. The system needs to handle inventory items and their quantities. Write a program that uses arrays to store inventory items and their quantities, and includes functions to add new items, update quantities, and display the inventory.

- Define an array to store inventory items.
- Define an array to store corresponding quantities.
- Implement functions to:
    - Add a new item along with its quantity.
    - Update the quantity of an existing item.
    - Display the inventory items and quantities.

- Use the functions to manage the inventory and handle user interactions.

Program:

```
# Initialize arrays for inventory items and quantities
inventory_items <- character(0)
inventory_quantities <- numeric(0)

# Function to add a new item with quantity
add_item <- function(item, quantity) {
  inventory_items <<- c(inventory_items, item)
  inventory_quantities <<- c(inventory_quantities, quantity)
  cat("Item added to inventory.\n")
}

# Function to update quantity of an existing item
update_quantity <- function(item, new_quantity) {
  if (item %in% inventory_items) {
    item_index <- which(inventory_items == item)
    inventory_quantities[item_index] <<- new_quantity
    cat("Quantity updated.\n")
  } else {
    cat("Item not found in inventory.\n")
  }
}

# Function to display inventory
display_inventory <- function() {
  cat("Inventory Items and Quantities:\n")
  for (i in 1:length(inventory_items)) {
    cat(sprintf("%s: %d\n", inventory_items[i], inventory_quantities[i]))
  }
}

# Main program
```

```
while (TRUE) {
  cat("\n1. Add Item\n2. Update Quantity\n3. Display Inventory\n4. Exit\n")
  choice <- as.integer(readline("Enter your choice: "))

  if (choice == 1) {
    item <- readline("Enter item name: ")
    quantity <- as.integer(readline("Enter quantity: "))
    add_item(item, quantity)
  } else if (choice == 2) {
    item <- readline("Enter item name: ")
    new_quantity <- as.integer(readline("Enter new quantity: "))
    update_quantity(item, new_quantity)
  } else if (choice == 3) {
    display_inventory()
  } else if (choice == 4) {
    cat("Exiting the program. Goodbye!\n")
    break
  } else {
    cat("Invalid choice. Please try again.\n")
  }
}
```

Output:

```
1. Add Item
2. Update Quantity
3. Display Inventory
4. Exit
Enter your choice: 1
Enter item name: rice
Enter quantity: 100
Item added to inventory.

1. Add Item
2. Update Quantity
3. Display Inventory
4. Exit
Enter your choice: 1
Enter item name: bicuits
Enter quantity: 23
Item added to inventory.
```

Enter your choice:

Lab5: Dataframe

You are working as an educational analyst and need to analyze the performance of students in a school. You have data on student names, their scores in different subjects, and attendance. Write a program that uses data frames to manage and analyze student data, including calculating average scores, identifying students with low attendance, and generating a report.

Create a data frame to store student information with columns: "Name", "Math_Score", "Science_Score", "History_Score", "Attendance".

Implement functions to:

- Calculate the average scores for each student.
- Identify students with attendance below a certain threshold.
- Generate a report with student names, average scores, and attendance status.
- Use the functions to analyse student performance and generate the report.

Program:

```
# Load the 'dplyr' package for data manipulation
library(dplyr)


# Create a data frame to store student information
student_data <- data.frame(
  Name = character(0),
  Math_Score = numeric(0),
  Science_Score = numeric(0),
  History_Score = numeric(0),
  Attendance = numeric(0)
)


# Function to add student information
add_student <- function(name, math_score, science_score, history_score, attendance) {
  new_student <- data.frame(
    Name = name,
    Math_Score = math_score,
```

```
    Science_Score = science_score,
    History_Score = history_score,
    Attendance = attendance
  )
  student_data <<- bind_rows(student_data, new_student)
  cat("Student information added.\n")
}

# Function to calculate average scores
calculate_average_scores <- function() {
  avg_scores <- student_data %>%
    mutate(Average_Score = (Math_Score + Science_Score + History_Score) / 3) %>%
    select(Name, Average_Score)
  return(avg_scores)
}

# Function to identify students with low attendance
identify_low_attendance <- function(threshold) {
  low_attendance <- student_data %>%
    filter(Attendance < threshold) %>%
    select(Name, Attendance)
  return(low_attendance)
}

# Function to generate a performance report
generate_report <- function() {
  avg_scores <- calculate_average_scores()
  low_attendance <- identify_low_attendance(70)

  report <- merge(avg_scores, low_attendance, by = "Name", all = TRUE)
  report$Attendance[is.na(report$Attendance)] <- 100

  cat("Performance Report:\n")
```

```
  print(report)
}

# Main program
while (TRUE) {
  cat("\n1. Add Student\n2. Generate Report\n3. Exit\n")
  choice <- as.integer(readline("Enter your choice: "))

  if (choice == 1) {
    name <- readline("Enter student name: ")
    math_score <- as.numeric(readline("Enter math score: "))
    science_score <- as.numeric(readline("Enter science score: "))
    history_score <- as.numeric(readline("Enter history score: "))
    attendance <- as.numeric(readline("Enter attendance percentage: "))
    add_student(name, math_score, science_score, history_score, attendance)
  } else if (choice == 2) {
    generate_report()
  } else if (choice == 3) {
    cat("Exiting the program. Goodbye!\n")
    break
  } else {
    cat("Invalid choice. Please try again.\n")
  }
}
```

Explanation:

```
# Load the 'dplyr' package for data manipulation
library(dplyr)
```

This line loads the 'dplyr' package, which provides a wide range of functions for data manipulation and transformation. It's used in this program for tasks like filtering, summarizing, and selecting data within data frames.

```
# Create a data frame to store student information
student_data <- data.frame(
```

```
  Name = character(0),
  Math_Score = numeric(0),
  Science_Score = numeric(0),
  History_Score = numeric(0),
  Attendance = numeric(0)
)
```

Here, a data frame named student_data is created. It's initialized with empty columns for student names, math scores, science scores, history scores, and attendance. This data frame will hold all the student information.

```
# Function to add student information
add_student <- function(name, math_score, science_score, history_score, attendance) {
  new_student <- data.frame(
    Name = name,
    Math_Score = math_score,
    Science_Score = science_score,
    History_Score = history_score,
    Attendance = attendance
  )
  student_data <<- bind_rows(student_data, new_student)
  cat("Student information added.\n")
}
```

This function add_student adds a new student's information to the student_data data frame. It creates a new data frame new_student with the provided information and then uses the bind_rows function from the 'dplyr' package to append this new student's data to the existing data frame.

```
# Function to calculate average scores
calculate_average_scores <- function() {
  avg_scores <- student_data %>%
    mutate(Average_Score = (Math_Score + Science_Score + History_Score) / 3) %>%
    select(Name, Average_Score)
  return(avg_scores)
}
```

The calculate_average_scores function calculates the average scores for each student. It uses the %>% (pipe) operator from the 'dplyr' package to perform operations sequentially. The mutate

function adds a new column Average_Score calculated by averaging math, science, and history scores. The select function extracts only the "Name" and "Average_Score" columns.

```
# Function to identify students with low attendance
identify_low_attendance <- function(threshold) {
  low_attendance <- student_data %>%
    filter(Attendance < threshold) %>%
    select(Name, Attendance)
  return(low_attendance)
}
```

The identify_low_attendance function identifies students with attendance below a specified threshold. It uses the filter function to extract rows where attendance is below the threshold, and the select function to extract the "Name" and "Attendance" columns.

```
# Function to generate a performance report
generate_report <- function() {
  avg_scores <- calculate_average_scores()
  low_attendance <- identify_low_attendance(70)

  report <- merge(avg_scores, low_attendance, by = "Name", all = TRUE)
  report$Attendance[is.na(report$Attendance)] <- 100

  cat("Performance Report:\n")
  print(report)
}
```

The generate_report function generates a performance report. It calls the calculate_average_scores and identify_low_attendance functions to get average scores and low attendance data. The merge function combines the data based on the "Name" column. It also handles missing attendance values by replacing them with 100 using is.na and assignment.

Output:

```
1. Add Student
2. Generate Report
3. Exit
Enter your choice: 1
Enter student name: ramesh
Enter math score: 67
Enter science score: 89
Enter history score: 89
```

Enter your choice:

Lab 6 and 7: R functions for statistical analysis

R provides a wide range of functions for statistical analysis. Here's an overview of some commonly used R packages and their functions for statistical analysis:

Base R:

mean(), median(), sd(): Calculate mean, median, and standard deviation.

cor(), cov(): Compute correlation and covariance matrices.

t.test(): Perform t-tests for comparing means.

wilcox.test(), kruskal.test(): Perform non-parametric tests.

lm(): Fit linear regression models.

anova(), summary(): Perform analysis of variance and obtain model summaries.

chisq.test(), fisher.test(): Conduct tests for contingency tables.

dplyr Package:

filter(), select(), mutate(): Manipulate and transform data.

group_by(), summarize(): Group data and calculate summary statistics.

arrange(): Sort data based on variables.

join(): Merge data frames.

ggplot2 Package:

ggplot(), geom_point(), geom_line(), etc.: Create sophisticated visualizations.

facet_wrap(), facet_grid(): Create small multiples plots.

theme(), labs(): Customize plot appearance and labels.

tidyr Package:

gather(), spread(): Convert data between wide and long formats.

separate(), unite(): Split and combine variables.

stats Package:

pnorm(), qnorm(): Compute cumulative and quantile normal distribution probabilities.

pf(), qf(): Compute cumulative and quantile F-distribution probabilities.

pchisq(), qchisq(): Compute cumulative and quantile chi-squared distribution probabilities.

MASS Package:

fitdistr(): Fit distributions to data.

stepAIC(): Perform stepwise model selection.

survival Package:

survfit(): Fit Kaplan-Meier survival curves.

coxph(): Fit Cox proportional hazards models.

caret Package:

train(): Train predictive models.

confusionMatrix(): Compute confusion matrices for classification models.

forecast Package:

auto.arima(): Fit automatic ARIMA time series models.

ets(), tbats(): Fit exponential smoothing models.

These are just a few examples of functions available for statistical analysis in R. Depending on your specific analysis needs, you might need to explore and use additional R packages and functions that cater to your domain and research interests.

Lab 6 :

You are a data analyst at a retail company that sells products online. The company is interested in predicting sales for the upcoming months to better manage inventory and plan marketing strategies. As part of your role, you need to develop a program that utilizes time series analysis to forecast sales based on a historical sales dataset.

Write an R program to forecast sales for the next three months using time series analysis techniques. The program should perform the following steps:

- Load the required libraries, including the forecast package.
- Create a data frame with two columns: Month and Sales. The Month column should contain a sequence of dates from January 2023 to June 2023 (inclusive), and the Sales column should contain the corresponding sales amounts (12000, 15000, 18000, 16000, 20000, 22000).
- Convert the sales data into a time series object with a monthly frequency.
- Fit an ARIMA (AutoRegressive Integrated Moving Average) model to the sales time series using the auto.arima() function.

- Forecast sales for the next three months using the fitted ARIMA model and the forecast() function.
- Display the forecasted sales results, including point forecasts and prediction intervals.

Code:

```
# Load required libraries
library(forecast)

# Create a data frame with Month and Sales columns
sales_data <- data.frame(
  Month = seq(as.Date("2023-01-01"), as.Date("2023-06-01"), by = "months"),
  Sales = c(12000, 15000, 18000, 16000, 20000, 22000)
)

# Convert to time series
sales_ts <- ts(sales_data$Sales, frequency = 12)

# Fit ARIMA model
arima_model <- auto.arima(sales_ts)

# Forecast sales for next 3 months
forecast_result <- forecast(arima_model, h = 3)

# Display forecast results
print(forecast_result)
```

Output:

```
print(forecast_result)
     Point Forecast    Lo 80    Hi 80
Jul 1          22000 18285.70 25714.30
Aug 1          22000 16747.19 27252.81
Sep 1          22000 15566.65 28433.35
        Lo 95    Hi 95
Jul 1 16319.47 27680.53
Aug 1 13966.52 30033.48
Sep 1 12161.04 31838.96
```

Lab 8 : Customer Purchase Analysis for E-commerce Company (Enhanced)

You are a data analyst working for an e-commerce company that specializes in selling a variety of products online. The company aims to analyze customer purchase data comprehensively to gain insights into customer behavior and spending patterns.

Your goal is to develop a R program that performs an in-depth analysis of customer purchase data. You will calculate various statistical measures and generate visualizations to understand the distribution of purchase amounts among customers.

Note: Load the necessary libraries, including the dplyr and ggplot2 packages.

Given the example customer purchase data provided below, create a data frame named purchase_data with two columns: CustomerID and PurchaseAmount.

Calculate and display the following statistical measures:

- Mean (average) purchase amount
- Median purchase amount
- Standard deviation of purchase amounts
- 1st quartile (25th percentile) of purchase amounts
- 3rd quartile (75th percentile) of purchase amounts

Create a histogram to visualize the distribution of purchase amounts using the ggplot2 package. Display the histogram with appropriate labels and titles.

Example Customer Purchase Data:

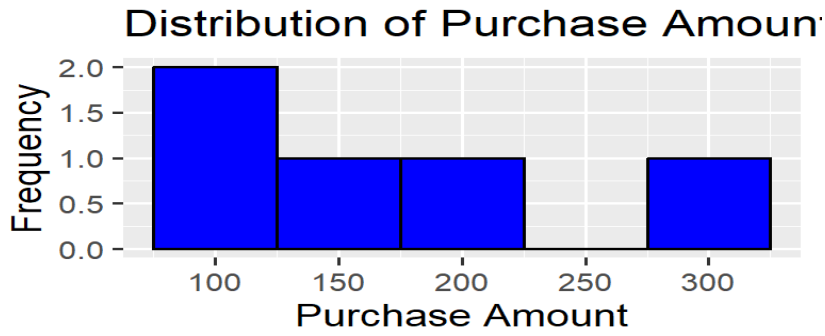
CustomerID	PurchaseAmount
101	150
102	200
103	120
104	300
105	80

Solution:

```
# Load required libraries
library(dplyr)
library(ggplot2)

# Example customer purchase data
purchase_data <- data.frame(
  CustomerID = c(101, 102, 103, 104, 105),
  PurchaseAmount = c(150, 200, 120, 300, 80)
)
```

```
# Calculate statistical measures
mean_purchase <- mean(purchase_data$PurchaseAmount)
median_purchase <- median(purchase_data$PurchaseAmount)
sd_purchase <- sd(purchase_data$PurchaseAmount)
q1_purchase <- quantile(purchase_data$PurchaseAmount, probs = 0.25)
q3_purchase <- quantile(purchase_data$PurchaseAmount, probs = 0.75)

# Display results
cat("Mean Purchase Amount:", mean_purchase, "\n")
cat("Median Purchase Amount:", median_purchase, "\n")
cat("Standard Deviation of Purchase Amounts:", sd_purchase, "\n")
cat("1st Quartile of Purchase Amounts:", q1_purchase, "\n")
cat("3rd Quartile of Purchase Amounts:", q3_purchase, "\n")

# Create a histogram
ggplot(purchase_data, aes(x = PurchaseAmount)) +
  geom_histogram(binwidth = 50, fill = "blue", color = "black") +
  labs(title = "Distribution of Purchase Amounts", x = "Purchase Amount", y = "Frequency")
```

```
> # Calculate statistical measures
> mean_purchase <- mean(purchase_data$PurchaseAmount)
> median_purchase <- median(purchase_data$PurchaseAmount)
> sd_purchase <- sd(purchase_data$PurchaseAmount)
> q1_purchase <- quantile(purchase_data$PurchaseAmount, probs = 0.25)
> q3_purchase <- quantile(purchase_data$PurchaseAmount, probs = 0.75)
>
> # Display results
> cat("Mean Purchase Amount:", mean_purchase, "\n")
Mean Purchase Amount: 170
> cat("Median Purchase Amount:", median_purchase, "\n")
Median Purchase Amount: 150
> cat("Standard Deviation of Purchase Amounts:", sd_purchase, "\n")
Standard Deviation of Purchase Amounts: 84.85281
> cat("1st Quartile of Purchase Amounts:", q1_purchase, "\n")
1st Quartile of Purchase Amounts: 120
> cat("3rd Quartile of Purchase Amounts:", q3_purchase, "\n")
3rd Quartile of Purchase Amounts: 200
>
> # Create a histogram
```



Lab 8: Matrix Manipulation in R

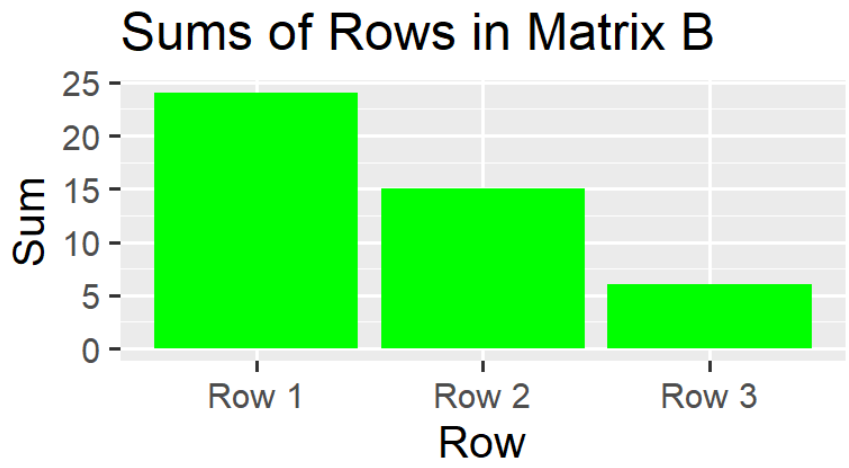
Write an R program that generates two matrices, matrix_A and matrix_B, and conducts operations including element-wise addition, scalar multiplication, matrix transpose, and multiplication.

Code:

```
# Task 1: Matrix Creation

matrix_A <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow = 3, ncol = 3, byrow = TRUE)

matrix_B <- matrix(c(9, 8, 7, 6, 5, 4, 3, 2, 1), nrow = 3, ncol = 3, byrow = TRUE)


# Task 2: Matrix Manipulation

sum_matrix <- matrix_A + matrix_B

scaled_matrix <- matrix_A * 2


# Task 3: Matrix Operations

transposed_A <- t(matrix_A)

product_matrix <- matrix_A %*% matrix_B


# Task 4: Matrix Statistics

sum_matrix_A <- sum(matrix_A)

mean_matrix_B <- mean(matrix_B)

sd_matrix_B <- sd(matrix_B)


# Task 5: Visualization

library(ggplot2)

library(reshape2)


# Create a heatmap of matrix_A

heatmap_data <- melt(matrix_A)
```

```
heatmap_plot <- ggplot(heatmap_data, aes(x = Var1, y = Var2, fill = value)) +
 geom_tile() +
 scale_fill_gradient(low = "white", high = "blue") +
 labs(title = "Heatmap of Matrix A", x = "Column", y = "Row")

# Create a bar plot comparing sums of rows in matrix_B
row_sums <- rowSums(matrix_B)
row_names <- paste("Row", 1:3)
barplot_data <- data.frame(Row = row_names, Sum = row_sums)
barplot_plot <- ggplot(barplot_data, aes(x = Row, y = Sum)) +
 geom_bar(stat = "identity", fill = "green") +
 labs(title = "Sums of Rows in Matrix B", x = "Row", y = "Sum")

# Display the visualizations
print(heatmap_plot)
print(barplot_plot)
```

output:



Lab 9 and 10: Visualization

You are a data analyst tasked with analyzing and visualizing a dataset. The dataset contains information about student performance in a course. You need to create a program that generates various types of plots to help understand and present the data effectively.

Write a program that performs data analysis and generates visualizations for a given dataset. The program should:

a) Load the necessary libraries (ggplot2).

b) Prepare example data with columns for student names, scores, and attendance percentages.
c) Perform the following tasks:
   i. Create a scatter plot to visualize the relationship between scores and attendance percentages.
   ii. Generate a bar plot to show the distribution of scores among different students.
   iii. Create a line plot to display the trend of scores over time (assuming different students' scores were collected at different time intervals).
   iv. Generate a histogram to visualize the distribution of scores.
d) Customize the appearance of each plot, such as color, labels, and titles.
e) Arrange the plots in a way that they are easy to compare and understand.
f) Provide appropriate titles for each plot and the axes.

Scatter Plot:

```
library(ggplot2)


# Example data
data <- data.frame(x = rnorm(50), y = rnorm(50))


# Scatter plot
ggplot(data, aes(x, y)) +
  geom_point() +
  labs(title = "Scatter Plot", x = "X-axis", y = "Y-axis")
```

Bar Plot:

```
library(ggplot2)
# Example data
data <- data.frame(category = c("A", "B", "C"), value = c(10, 20, 15))
# Bar plot
ggplot(data, aes(x = category, y = value)) +
  geom_bar(stat = "identity") +
  labs(title = "Bar Plot", x = "Category", y = "Value")
```

Line Plot:

```
library(ggplot2)
# Example data
data <- data.frame(x = 1:10, y = 2 * (1:10))
# Line plot
ggplot(data, aes(x, y)) +
```

```
  geom_line() +
  labs(title = "Line Plot", x = "X-axis", y = "Y-axis")
```

Histogram:

```
library(ggplot2)
# Example data
data <- data.frame(values = rnorm(100))
# Histogram
ggplot(data, aes(values)) +
  geom_histogram(binwidth = 0.5, fill = "blue", color = "black") +
  labs(title = "Histogram", x = "Values", y = "Frequency")
```

scatter, bar, line, histogram, and box plots

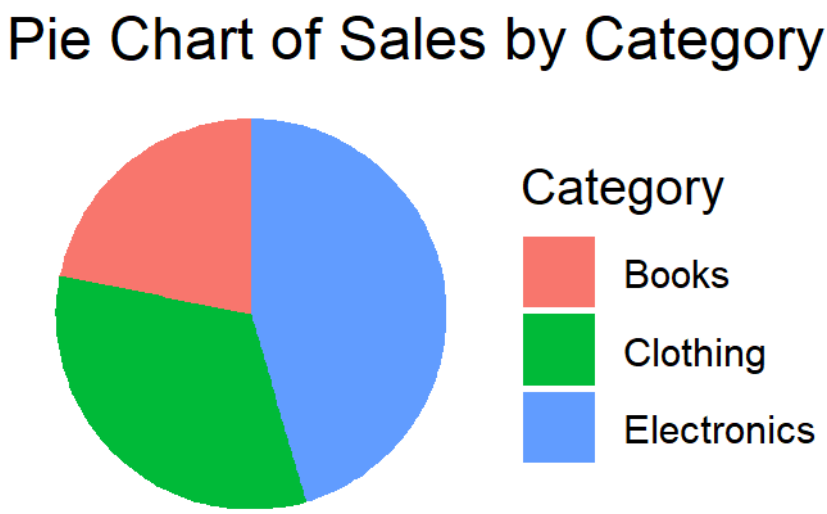
```
# Load required libraries
library(ggplot2)
library(patchwork)
library(plotly)

# Example data
set.seed(123)
data <- data.frame(
  Date = seq(as.Date("2023-01-01"), as.Date("2023-12-31"), by = "months"),
  Sales = cumsum(sample(5000:10000, 12, replace = TRUE)),
  Engagement = sample(1:10, 12, replace = TRUE),
  Category = rep(c("Electronics", "Clothing", "Books"), each = 4)
)

# Scatter plot
scatter_plot <- ggplot(data, aes(x = Sales, y = Engagement, color = Category)) +
  geom_point() +
  labs(title = "Sales vs. Customer Engagement", x = "Sales", y = "Engagement")

# Bar plot for sales by category
bar_plot <- ggplot(data, aes(x = Category, y = Sales, fill = Category)) +
```

```
  geom_bar(stat = "identity") +
  labs(title = "Sales by Category", x = "Category", y = "Sales") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Line plot for sales over time
line_plot <- ggplot(data, aes(x = Date, y = Sales, color = Category)) +
  geom_line() +
  labs(title = "Sales Trend Over Time", x = "Date", y = "Sales")

# Histogram for customer engagement
histogram_plot <- ggplot(data, aes(Engagement)) +
  geom_histogram(binwidth = 1, fill = "green", color = "black") +
  labs(title = "Customer Engagement Distribution", x = "Engagement", y = "Frequency")

# Box plot for sales by category
box_plot <- ggplot(data, aes(Category, Sales, fill = Category)) +
  geom_boxplot() +
  labs(title = "Sales Distribution by Category", x = "Category", y = "Sales")

# Combine plots using patchwork
combined_plots <- scatter_plot +
  bar_plot +
  line_plot +
  histogram_plot +
  box_plot

# Display combined plots
print(combined_plots)
```

PIE chart in R:

```
# Load required libraries
library(ggplot2)
```

```
# Example data
data <- data.frame(
  Category = c("Electronics", "Clothing", "Books"),
  Sales = c(25000, 18000, 12000)
)

# Create a pie chart
pie_chart <- ggplot(data, aes(x = "", y = Sales, fill = Category)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar("y", start = 0) +  # Convert to polar coordinates
  labs(title = "Pie Chart of Sales by Category") +
  theme_void()  # Remove axes and labels

# Display the pie chart
print(pie_chart)
```



LAB11: Data Analysis: Statement: Customer Purchase Analysis

You are a data analyst at an e-commerce company that sells a variety of products online. The company has provided you with a dataset containing information about customer purchases. Your task is to perform a comprehensive data analysis to gain insights into customer behavior and spending patterns.

Dataset Description:

The dataset customer_purchases.csv contains the following columns:

CustomerID: Unique identifier for each customer.

PurchaseAmount: The amount spent by the customer on a purchase.

Problem Tasks:

You are required to perform the following tasks using R:

Task 1: Load the Dataset

Load the necessary libraries, including readr and dplyr.

Read the dataset customer_purchases.csv into a data frame named purchase_data.

Task 2: Data Summary

Calculate and display the total number of records in the dataset.

Calculate and display the total number of unique customers in the dataset.

Task 3: Calculate Statistical Measures

Calculate and display the mean (average) purchase amount.

Calculate and display the median purchase amount.

Calculate and display the standard deviation of purchase amounts.

Task 4: Customer Segmentation

Create a new column named Segment in the purchase_data data frame based on the following criteria:

"Low Spender" if the purchase amount is less than the median.

"High Spender" if the purchase amount is greater than or equal to the median.

Task 5: Visualize Data

Create a histogram to visualize the distribution of purchase amounts using the ggplot2 package. Customize the plot with appropriate labels and titles.

Code:

```
# Load required libraries
library(readr)
library(dplyr)
library(ggplot2)

# Task 1: Load the Dataset
purchase_data <- read_csv("D:/RPrograms2023/LAb/customer_purchases.csv")
# Load required libraries
library(readr)
library(dplyr)
library(ggplot2)
```

```
# Task 1: Load the Dataset
# Task 1: Load the Dataset
purchase_data <- read_csv("D:/RPrograms2023/LAb/customer_purchases.csv")

# Task 2: Data Summary
total_records <- nrow(purchase_data)
total_customers <- n_distinct(purchase_data$CustomerID)

cat("Total Number of Records:", total_records, "\n")
cat("Total Number of Unique Customers:", total_customers, "\n")

# Task 3: Calculate Statistical Measures
mean_purchase <- mean(purchase_data$PurchaseAmount)
median_purchase <- median(purchase_data$PurchaseAmount)
sd_purchase <- sd(purchase_data$PurchaseAmount)

cat("Mean Purchase Amount:", mean_purchase, "\n")
cat("Median Purchase Amount:", median_purchase, "\n")
cat("Standard Deviation of Purchase Amounts:", sd_purchase, "\n")

# Task 4: Customer Segmentation (Based on Quartiles)
q1_purchase <- quantile(purchase_data$PurchaseAmount, probs = 0.25)
q2_purchase <- quantile(purchase_data$PurchaseAmount, probs = 0.5)
q3_purchase <- quantile(purchase_data$PurchaseAmount, probs = 0.75)

purchase_data <- purchase_data %>%
  mutate(Segment = case_when(
    PurchaseAmount < q1_purchase ~ "Low Spender",
    PurchaseAmount >= q1_purchase & PurchaseAmount < q3_purchase ~ "Medium Spender",
    PurchaseAmount >= q3_purchase ~ "High Spender"
  ))
```

```
# Task 5: Visualize Data (Histogram)
ggplot(purchase_data, aes(x = PurchaseAmount)) +
  geom_histogram(binwidth = 50, fill = "blue", color = "black") +
  labs(title = "Distribution of Purchase Amounts", x = "Purchase Amount", y = "Frequency")

# Task 6: Visualize Relationship (Scatter Plot)
ggplot(purchase_data, aes(x = CustomerID, y = PurchaseAmount)) +
  geom_point(color = "green") +
  labs(title = "Customer Purchase Amounts", x = "Customer ID", y = "Purchase Amount")

# Save the visualizations as image files
ggsave("purchase_histogram.png", width = 8, height = 6)
ggsave("customer_purchase_scatter.png", width = 8, height = 6)
```

LAB12: Data Analysis:

You are a data analyst tasked with performing an Exploratory Data Analysis (EDA) on the Indian Premier League (IPL) dataset. The IPL dataset contains information about various IPL matches, including match dates, teams, venues, outcomes, and performance metrics. Your objective is to gain insights into the dataset by conducting an in-depth analysis using R programming.

Dataset Description:

The dataset named "ipl_data.csv" includes the following columns:

Match_ID: Unique identifier for each match.

Date: Date of the match.

Team1: Name of the first team participating in the match.

Team2: Name of the second team participating in the match.

Venue: Stadium where the match was played.

Winner: Name of the winning team.

Total.Runs: Total runs scored in the match.

Total.Wickets: Total wickets taken in the match.

Other relevant columns (if any).

Problem Tasks:

Your task is to perform the following Exploratory Data Analysis (EDA) tasks using R:

Task 1: Data Overview and Structure

Display the structure of the dataset using str() function.

Output summary statistics of numerical columns using summary().

Task 2: Basic Data Insights

Calculate and display the total number of matches in the dataset.

Determine the number of unique teams that have participated in IPL matches.

List the unique teams from both Team1 and Team2.

Task 3: Team Performance Analysis

Calculate the number of matches won by each team and display the results.

Compute the average total runs scored in the matches.

Calculate the average total wickets taken in the matches.

Task 4: Venue Insights

Identify and display the most frequently used venue for matches.

Task 5: Visualization

Create a bar plot to visualize the number of matches won by each team.