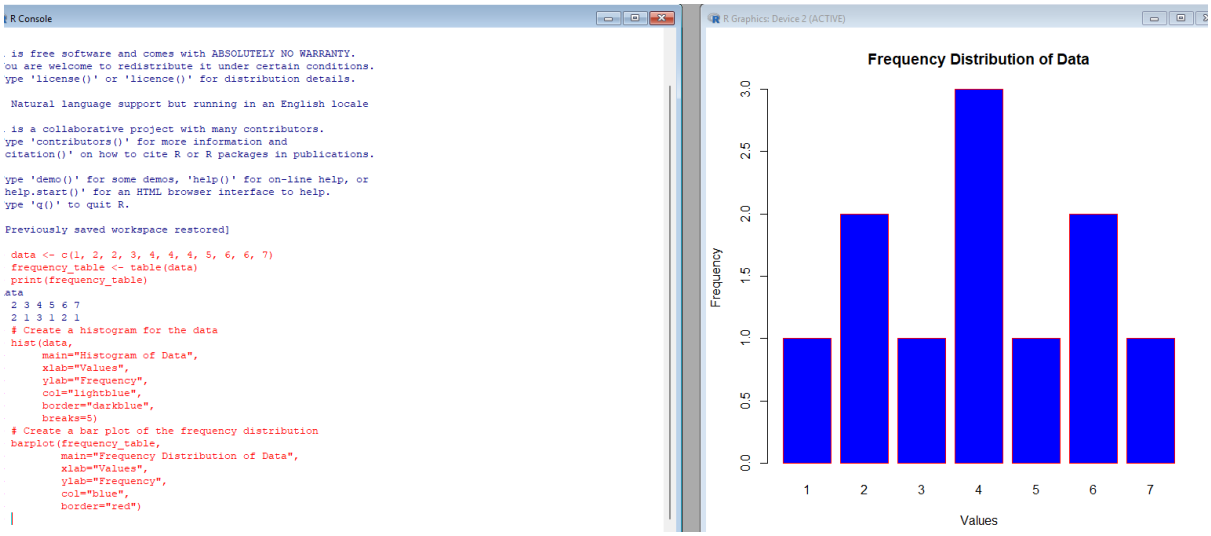
```
# Example data
data <- c(1, 2, 2, 3, 4, 4, 4, 5, 6, 6, 7)

# 1. Create Frequency Distribution
frequency_table <- table(data)
print("Frequency Distribution:")
print(frequency_table)

# 2. Bar Plot for Frequency Distribution
barplot(frequency_table,
        main="Frequency Distribution of Data",
        xlab="Values",
        ylab="Frequency",
        col="blue",
        border="red")

# 3. Histogram for Data (For Continuous Data)
hist(data,
     main="Histogram of Data",
     xlab="Values",
     ylab="Frequency",
     col="lightblue",
     border="darkblue",
     breaks=5)
```



Practical 2

Example data

data <- c(1, 2, 2, 3, 4, 4, 4, 5, 6, 6, 7)

1. Mean (Average) (The mean is calculated by summing all the numbers and dividing by the count of numbers.)

mean_value <- mean(data)

print(paste("Mean:", mean_value))

2. Median (Middle Value) :The median is the middle value when the data is ordered. If the data set has an even number of elements, the median is the average of the two middle values

median_value <- median(data)

print(paste("Median:", median_value))

3. Mode (Most Frequent Value)

The mode is the most frequent value in the data set. If no value repeats, there is no mode.

R Code to Calculate the Mode:

R doesn't have a built-in mode() function for finding the most frequent value. So, we need to write a custom function to find the mode.

- **The custom get_mode() function first finds the unique values in the data.**
- **Then, tabulate() counts the frequency of each unique value, and which.max() identifies the value with the highest frequency**.

```
get_mode <- function(data) {
  uniq_data <- unique(data)
  freq <- tabulate(match(data, uniq_data))
  uniq_data[which.max(freq)]
}
mode_value <- get_mode(data)
print(paste("Mode:", mode_value))
```

Practical4

Frequency Distribution

```
# Sample data
numbers <- c(5, 12, 18, 25, 30, 35, 40, 45, 50, 55)

# Define bins (intervals)
bins <- cut(numbers, breaks = c(0, 10, 20, 30, 40, 50, 60), right = FALSE)

# Create frequency table
frequency_table <- table(bins)

# Print results
cat("Numbers:", numbers, "\n")
cat("Frequency Distribution:\n")
print(frequency_table)
```

(explanation: cut() divides numbers into bins.

breaks define intervals [0-10), [10-20), ..., [50-60).

table() counts how many numbers fall into each bin.

print(frequency_table) displays the frequency distribution.)

cat("Numbers:", numbers, "\n")

- **cat() is used to concatenate and print text and values in a single line.**

- **"Numbers:" is a label to describe the output.**
- **numbers prints the actual vector of numbers.**
- **"\n" inserts a new line to ensure proper formatting.**

Practical 5

Data Presentation

```
# Sample Data: Student Scores
scores <- c(85, 90, 78, 92, 88, 76, 95, 89, 84, 91)

# Creating a Data Frame
student_data <- data.frame(
  ID = 1:10,
  Name = c("Alice", "Bob", "Charlie", "David", "Emma", "Frank", "Grace", "Helen", "Ivy", "Jack"),
  Score = scores
)

# Displaying Data
print("Student Data Frame:")
print(student_data)

# Summary Statistics
print("Summary Statistics:")
print(summary(student_data$Score))  # Min, Max, Mean, Median, 1st & 3rd Quartiles

# Frequency Distribution
score_bins <- cut(scores, breaks = c(70, 80, 90, 100), right = FALSE)
frequency_table <- table(score_bins)
print("Frequency Distribution:")
print(frequency_table)
```

```
# Visualization: Histogram
hist(scores, breaks = c(70, 80, 90, 100), col = "lightblue",
    main = "Score Distribution", xlab = "Scores", ylab = "Frequency")

# Bar Plot for Frequency
barplot(frequency_table, col = "blue",
      main = "Frequency of Score Ranges", xlab = "Score Ranges", ylab = "Count")
```

Explanation:

1. **Data Entry:**
    - **Created a data frame with student names and scores.**
2. **Data Display (print())**
    - **Prints the student data frame in tabular format.**
3. **Summary Statistics (summary())**
    - **Displays min, max, mean, median, and quartiles of scores.**
4. **Frequency Distribution (cut(), table())**
    - **Groups scores into ranges (70-80, 80-90, 90-100) and counts occurrences.**
5. **Data Visualization (hist(), barplot())**
    - **Histogram: Shows score distribution.**
    - **Bar Plot: Displays frequency of score ranges.**

Practical6

Probability

```
# Probability of Coin Toss Outcomes
coin <- c("Heads", "Tails")
coin_toss <- sample(coin, size = 1, replace = TRUE)
cat("Coin Toss Result:", coin_toss, "\n")
```

Explanation:

- **`Coin <- c("Heads", "Tails")`**
    - **This line creates a vector called `coin` containing two elements: `"Heads"` and `"Tails"`. This vector represents the possible outcomes of a coin toss.**

- `coin_toss <- sample(coin, size = 1, replace = TRUE)`

    - **The `sample()` function is used to randomly select an element from the `coin` vector.**
        - **`coin` is the vector from which the sampling will happen.**
        - **`size = 1` means that only one element (either "Heads" or "Tails") will be selected randomly.**
        - **`replace = TRUE` allows for replacement, meaning that after selecting an element, it could be chosen again (though in this case, it doesn't really affect the result since you're selecting only one item).**

- `cat("Coin Toss Result:", coin_toss, "\n")`

    - **`cat()` is used to concatenate and print the output.**
    - **`"Coin Toss Result:"` is a string that will be printed before the result.**
    - **`coin_toss` is the result of the coin toss, which will be either `"Heads"` or `"Tails"`.**
    - **`"\n"` adds a newline character at the end of the output, so the output appears neatly on the console.**

Probability of Rolling a Dice

dice <- 1:6

dice_roll <- sample(dice, size = 1, replace = TRUE)

cat("Dice Roll Result:", dice_roll, "\n")

Explanation:

1. **`dice <- 1:6`**
    - **This line creates a vector called `dice` that contains the numbers 1 through 6. These represent the possible outcomes when rolling a standard six-sided die.**
2. **`dice_roll <- sample(dice, size = 1, replace = TRUE)`**
    - **The `sample()` function is used here to randomly select one value from the `dice` vector.**
        - **`dice` is the vector from which to randomly choose a number (the sides of the die).**
        - **`size = 1` means that we are selecting a single value from the `dice` vector.**
        - **`replace = TRUE` specifies that sampling is with replacement, meaning that after selecting a number, it's as if it is "replaced" back into the pool (though with just one number being selected, it doesn't affect the outcome).**
3. **`cat("Dice Roll Result:", dice_roll, "\n")`**
    - **The `cat()` function is used to print a message to the console.**
    - **`"Dice Roll Result:"` is the string that will be printed before the result, to provide context for what the number represents.**
    - **`dice_roll` is the result of the dice roll, which will be a single number between 1 and 6, inclusive.**

- **`"\n"` is a newline character, which adds a line break after the result for better readability.**

In summary:

This code simulates the rolling of a six-sided die by randomly selecting a number between 1 and 6, and then prints the result of that roll.

```
# Simulating Multiple Coin Tosses

num_tosses <- 100

coin_tosses <- sample(coin, size = num_tosses, replace = TRUE)

coin_table <- table(coin_tosses)

cat("Coin Toss Frequency (100 Tosses):\n")

print(coin_table)
```

Explanation:

1. **`num_tosses <- 100`**
   - **This line defines a variable called `num_tosses` and sets it to 100. It represents the number of coin tosses that will be simulated.**
2. **`coin_tosses <- sample(coin, size = num_tosses, replace = TRUE)`**
   - **The `sample()` function is used here to simulate multiple coin tosses.**
     - **`coin` is the vector containing the two possible outcomes of a coin toss: `"Heads"` and `"Tails"`.**
     - **`size = num_tosses` means we are generating 100 random tosses (because `num_tosses` is 100).**
     - **`replace = TRUE` ensures that after each coin toss, the outcome is "replaced" back into the pool, meaning each toss is independent and has an equal chance of being "Heads" or "Tails".**

   As a result, `coin_tosses` is a vector of 100 elements, each being either `"Heads"` or `"Tails"`, chosen randomly.

3. **`coin_table <- table(coin_tosses)`**
   - **The `table()` function is used here to count the occurrences of each outcome in the `coin_tosses` vector.**
     - **It creates a table (a frequency count) of how many times `"Heads"` and `"Tails"` appear in the `coin_tosses` vector.**

   The result, `coin_table`, will be a table with the frequency count of "Heads" and "Tails" from the 100 tosses.

4. **`cat("Coin Toss Frequency (100 Tosses):\n")`**

- **The `cat()` function is used to print the text `"Coin Toss Frequency (100 Tosses):"` to the console, followed by a newline (`\n`), to introduce the result and make the output clearer.**

5. **`print(coin_table)`**
   - **The `print()` function is used to display the `coin_table`, which shows how many times "Heads" and "Tails" appeared in the 100 tosses.**

In summary:

This code simulates 100 coin tosses, counts how many times each outcome ("Heads" and "Tails") occurs, and then prints the frequency count of each outcome. The output will show how many times each side of the coin came up in the 100 simulated tosses.

Explanation:

1. **Simulating Coin Toss (sample())**
   - **Randomly selects "Heads" or "Tails".**
2. **Simulating Dice Roll (sample())**
   - **Randomly picks a number from 1 to 6.**
3. **Simulating Multiple Coin Tosses (table())**
   - **Performs 100 tosses and counts occurrences of Heads/Tails.**

Practical 7

```
# 1. Creating Numeric Vectors Using c()

num_vector <- c(1, 2, 3, 4, 5) # Numeric vector

print("Numeric Vector:")

print(num_vector)


# 2. Creating Text Vectors Using c()

text_vector <- c("apple", "banana", "cherry")

print("Text Vector:")

print(text_vector)


# 3. Creating a Sequence with seq()
```

```
seq_vector <- seq(1, 10, by=2)  # Sequence from 1 to 10 with a step of 2
print("Sequence Vector:")
print(seq_vector)

# 4. Mathematical Operations on Vectors
sum_result <- num_vector + 2     # Addition: Add 2 to each element
diff_result <- num_vector - 2    # Subtraction: Subtract 2 from each element
prod_result <- num_vector * 2    # Multiplication: Multiply each element by 2
div_result <- num_vector / 2     # Division: Divide each element by 2
exp_result <- exp(num_vector)    # Exponential
log_result <- log(num_vector)    # Natural Logarithm
log10_result <- log10(num_vector) # Log base 10

print("Addition Result (num_vector + 2):")
print(sum_result)
print("Subtraction Result (num_vector - 2):")
print(diff_result)
print("Multiplication Result (num_vector * 2):")
print(prod_result)
print("Division Result (num_vector / 2):")
print(div_result)
print("Exponential Result (exp(num_vector)):")
print(exp_result)
print("Logarithm (Natural Log) Result (log(num_vector)):")
print(log_result)
print("Logarithm (Base 10) Result (log10(num_vector)):")
print(log10_result)
```

Practical 8

```
# 5. Data Entry Using scan()
print("Enter a series of numbers: ")
user_input <- scan()   # Accept user input for a vector
```

```
print("You entered:")
print(user_input)

# 6. Creating a Data Frame
data <- data.frame(
  Numbers = num_vector,
  Square = num_vector^2,
  Log10 = log10(num_vector)
)
print("Data Frame Example:")
print(data)

# 7. Matrix Operations
matrix1 <- matrix(1:9, nrow=3, byrow=TRUE)  # Create a 3x3 matrix
matrix2 <- matrix(9:1, nrow=3, byrow=TRUE)  # Another 3x3 matrix
matrix_sum <- matrix1 + matrix2  # Matrix addition
matrix_prod <- matrix1 %*% matrix2  # Matrix multiplication (dot product)

print("Matrix 1:")
print(matrix1)
print("Matrix 2:")
print(matrix2)
print("Matrix Addition Result (matrix1 + matrix2):")
print(matrix_sum)
print("Matrix Multiplication Result (matrix1 %*% matrix2):")
print(matrix_prod)

# 8. Using split() for Text Data
split_result <- split(text_vector, c(1, 1, 2))  # Split the vector into two groups
print("Split Vector Result:")
print(split_result)
```

explanation: • **Creating Numeric and Text Vectors**:

- `num_vector` holds numeric values, and `text_vector` holds string values.

- **Mathematical Operations**:
  - We apply basic operations like addition, subtraction, multiplication, division on vectors.
  - We also perform exponential (`exp()`), natural logarithm (`log()`), and base-10 logarithm (`log10()`) operations.

- **Data Entry Using `scan()`**:
  - `scan()` is used to enter a series of numbers interactively.

- **Creating a Data Frame**:
  - A data frame is created with the `data.frame()` function that stores the numeric values, their squares, and their base-10 logarithms.

- **Matrix Operations**:
  - We create two matrices and perform matrix addition and multiplication. Note the use of `%*%` for matrix multiplication.

- **Split Function**:
  - The `split()` function splits the vector into groups based on a given factor.