Different Types of R data structure

1. Vectors

Definition:

A **vector** is the most basic data structure in R. It is a sequence of data elements of the same type (numeric, character, or logical). Vectors are atomic, meaning they cannot contain elements of different types.

Use Cases:

- **Storing Homogeneous Data**: When you need to store a series of data points that are all of the same type.
- **Mathematical Operations**: Vectors are ideal for performing operations like addition, subtraction, and multiplication on a series of numbers.
- **Logical Indexing**: Useful for subsetting data or filtering based on conditions.

Example:
Use Case: Storing a list of temperatures recorded over a week.

```
temperatures <- c(25.3, 26.1, 24.8, 23.9, 27.2, 26.4, 25.7)
```

2. Lists

Definition:

A **list** is a data structure in R that can contain elements of different types, including numbers, strings, vectors, and even other lists. Lists are heterogeneous and are used to group together related data of varying types.

Use Cases:

- **Storing Mixed Data Types**: When you need to store related data that may vary in type, such as a combination of strings, numbers, and vectors.
- **Complex Data Structures**: Lists can be used to create more complex data structures like trees, graphs, or objects in R.

Example:
Use Case: Storing information about a book, including its title (string), author (string), publication year (integer), and chapters (vector of strings).

```
book_info <- list(
  title = "The Art of R Programming",
  author = "Norman Matloff",
  year = 2011,
  chapters = c("Introduction", "Vectors", "Lists", "Data Frames")
)
```

3. Matrices

Definition:

A **matrix** is a two-dimensional data structure in R, where each element has the same data type. It is essentially a vector with dimensions (rows and columns).

Use Cases:

- **Mathematical and Statistical Computations**: Matrices are often used in linear algebra operations, such as matrix multiplication, finding eigenvalues, and solving linear systems.
- **Tabular Data with Uniform Type**: Suitable when you have tabular data with homogeneous types, such as a dataset of numerical measurements.

Example:
Use Case: Representing a dataset of exam scores for three students in four subjects.

```
scores_matrix <- matrix(c(85, 90, 78, 92, 88, 76, 94, 82, 89, 85, 80, 91), nrow = 3, byrow = TRUE)
```

4. Data Frames

Definition:

A **data frame** is a table or two-dimensional array-like structure in R, where each column contains values of one variable, and each row contains one set of values from each column. Unlike matrices, data frames can contain different types of data (numeric, character, factor) in different columns.

Use Cases:

- **Handling Tabular Data**: Data frames are the most commonly used data structure for storing datasets in R, especially when the dataset contains variables of different types.

- **Data Analysis and Manipulation**: Useful for performing operations like filtering, subsetting, aggregating, and joining datasets.

Example:
Use Case: Storing a dataset of employees, including their names (character), ages (numeric), and departments (factor).

```
employees <- data.frame(
  Name = c("John", "Jane", "Doe"),
  Age = c(28, 34, 45),
  Department = factor(c("HR", "Finance", "IT"))
)
```

5. Factors

Definition:

A **factor** is a data structure used for fields that take on a limited number of different values; these values are known as levels. Factors are used for categorical data and can be ordered or unordered.

Use Cases:

- **Categorical Data**: Factors are ideal for representing categorical variables in a dataset, such as gender, education level, or customer satisfaction ratings.
- **Statistical Modeling**: Factors are often used in statistical modeling, especially in regression models, where categorical predictors need to be encoded.

Example:
Use Case: Representing the satisfaction level of survey respondents.

```
satisfaction <- factor(c("Satisfied", "Dissatisfied", "Neutral", "Satisfied",
"Dissatisfied"))
```

Summary of When to Use Each Data Structure:

- **Use Vectors** when you need to store a sequence of homogeneous data elements (e.g., numeric, character, logical) for mathematical operations or logical indexing.
- **Use Lists** when you need to group together related data of different types or create more complex data structures.
- **Use Matrices** when working with two-dimensional homogeneous data, particularly for mathematical and statistical computations.

- **Use Data Frames** when dealing with tabular data that includes variables of different types (numeric, character, factor) and for data analysis tasks.
- **Use Factors** when working with categorical data, particularly in datasets that involve statistical analysis or modeling with categorical predictors.

Vectors

1. Creating a Numeric Vector

```
numeric_vector <- c(10, 20, 30, 40, 50)
print(numeric_vector)
```

2. Creating a Character Vector

```
char_vector <- c("apple", "banana", "cherry")
print(char_vector)
```

3. Creating a Logical Vector

```
logical_vector <- c(TRUE, FALSE, TRUE, FALSE)
print(logical_vector)
```

4. Accessing Elements in a Vector

```
vector <- c(5, 10, 15, 20)
element <- vector[3]
print(element)
```

5. Modifying Elements in a Vector

```
vector <- c(1, 2, 3, 4, 5)
vector[2] <- 10
print(vector)
```

6. Vector Length

```
vector <- c(1, 2, 3, 4, 5)
length_vector <- length(vector)
print(length_vector)
```

7. Combining Two Vectors

```
vec1 <- c(1, 2, 3)
vec2 <- c(4, 5, 6)
combined_vector <- c(vec1, vec2)
print(combined_vector)
```

8. Repeating a Vector

```
vector <- c(1, 2, 3)
repeated_vector <- rep(vector, times = 3)
print(repeated_vector)
```

9. Creating a Sequence in a Vector

```
sequence_vector <- seq(1, 10, by = 2)
print(sequence_vector)
```

10. Sorting a Vector

```
unsorted_vector <- c(3, 1, 4, 1, 5)
sorted_vector <- sort(unsorted_vector)
print(sorted_vector)
```

11. Vector Arithmetic (Addition)

```
vec1 <- c(1, 2, 3)
vec2 <- c(4, 5, 6)
```

```
sum_vector <- vec1 + vec2
print(sum_vector)
```

12. Vector Arithmetic (Multiplication)

```
vec1 <- c(2, 4, 6)
vec2 <- c(1, 3, 5)
product_vector <- vec1 * vec2
print(product_vector)
```

13. Logical Operations on Vectors

```
vector <- c(1, 2, 3, 4, 5)
logical_result <- vector > 3
print(logical_result)
```

14. Subsetting a Vector

```
vector <- c(10, 20, 30, 40, 50)
subset_vector <- vector[vector > 25]
print(subset_vector)
```

15. Checking for NA Values

```
vector <- c(1, 2, NA, 4, 5)
na_check <- is.na(vector)
print(na_check)
```

16. Replacing NA Values

```
vector <- c(1, 2, NA, 4, 5)
vector[is.na(vector)] <- 0
print(vector)
```

17. Finding the Maximum Value in a Vector

```
vector <- c(5, 10, 15, 20)
max_value <- max(vector)
print(max_value)
```

18. Finding the Minimum Value in a Vector

```
vector <- c(5, 10, 15, 20)
min_value <- min(vector)
print(min_value)
```

19. Calculating the Sum of Vector Elements

```
vector <- c(1, 2, 3, 4, 5)
sum_vector <- sum(vector)
print(sum_vector)
```

20. Calculating the Mean of Vector Elements

```
vector <- c(10, 20, 30, 40, 50)
mean_vector <- mean(vector)
print(mean_vector)
```

21. Calculating the Standard Deviation of Vector Elements

```
vector <- c(10, 20, 30, 40, 50)
sd_vector <- sd(vector)
print(sd_vector)
```

22. Vector Indexing with a Logical Vector

```
vector <- c(5, 10, 15, 20)
logical_index <- vector > 10
```

```
filtered_vector <- vector[logical_index]
print(filtered_vector)
```

23. Using `any` and `all` Functions with Vectors

```
vector <- c(1, 2, 3, 4, 5)
any_result <- any(vector > 4)
all_result <- all(vector > 4)
print(any_result)
print(all_result)
```

24. Reversing a Vector

```
vector <- c(10, 20, 30, 40, 50)
reversed_vector <- rev(vector)
print(reversed_vector)
```

25. Finding the Index of a Specific Element

```
vector <- c("apple", "banana", "cherry")
index <- which(vector == "banana")
print(index)
```

List

1. Creating a Simple List

```
my_list <- list(1, "apple", TRUE)
print(my_list)
```

2. Creating a Named List

```
named_list <- list(number = 1, fruit = "apple", logic = TRUE)
```

```
print(named_list)
```

3. Accessing an Element in a List by Index

```
my_list <- list(1, "apple", TRUE)
element <- my_list[[2]]
print(element)
```

4. Accessing an Element in a List by Name

```
named_list <- list(number = 1, fruit = "apple", logic = TRUE)
element <- named_list$fruit
print(element)
```

5. Modifying an Element in a List

```
my_list <- list(1, "apple", TRUE)
my_list[[2]] <- "banana"
print(my_list)
```

6. Adding a New Element to a List

```
my_list <- list(1, "apple", TRUE)
my_list[[4]] <- "new element"
print(my_list)
```

7. Removing an Element from a List

```
my_list <- list(1, "apple", TRUE)
my_list[[2]] <- NULL
print(my_list)
```

8. Merging Two Lists

```
list1 <- list(1, 2, 3)
list2 <- list("a", "b", "c")
merged_list <- c(list1, list2)
print(merged_list)
```

9. Applying a Function to Each Element in a List with `lapply`

```
my_list <- list(1, 2, 3)
squared_list <- lapply(my_list, function(x) x^2)
print(squared_list)
```

10. Converting a List to a Vector

```
my_list <- list(1, 2, 3)
vector <- unlist(my_list)
print(vector)
```

11. Checking the Length of a List

```
my_list <- list(1, "apple", TRUE)
length_list <- length(my_list)
print(length_list)
```

12. Creating a Nested List

```
nested_list <- list(a = list(1, 2), b = list(3, 4))
print(nested_list)
```

13. Accessing Elements in a Nested List

```
nested_list <- list(a = list(1, 2), b = list(3, 4))
element <- nested_list$a[[2]]
print(element)
```

14. Modifying Elements in a Nested List

```
nested_list <- list(a = list(1, 2), b = list(3, 4))
nested_list$a[[1]] <- 10
print(nested_list)
```

15. Flattening a Nested List

```
nested_list <- list(a = list(1, 2), b = list(3, 4))
flattened_list <- unlist(nested_list)
print(flattened_list)
```

16. Creating a List with Different Data Types

```
mixed_list <- list(1, "apple", c(2, 4, 6), list(5, "banana"))
print(mixed_list)
```

17. Checking if an Element Exists in a List

```
my_list <- list(1, "apple", TRUE)
element_exists <- "apple" %in% my_list
print(element_exists)
```

18. Extracting a Sublist from a List

```
my_list <- list(1, "apple", TRUE, 4)
sublist <- my_list[2:3]
print(sublist)
```

19. Combining Lists using append

```
list1 <- list(1, 2, 3)
```

```
list2 <- list("a", "b", "c")
combined_list <- append(list1, list2)
print(combined_list)
```

20. Applying a Function to a Named List with `lapply`

```
named_list <- list(a = 1, b = 2, c = 3)
doubled_list <- lapply(named_list, function(x) x * 2)
print(doubled_list)
```

21. Using `sapply` to Simplify List Output

```
my_list <- list(1, 2, 3)
squared_list <- sapply(my_list, function(x) x^2)
print(squared_list)
```

22. Filtering a List Based on a Condition

```
my_list <- list(1, 2, 3, 4, 5)
filtered_list <- Filter(function(x) x > 3, my_list)
print(filtered_list)
```

23. Sorting a List

```
my_list <- list(5, 3, 2, 4, 1)
sorted_list <- sort(unlist(my_list))
print(sorted_list)
```

24. Finding the Length of Each Element in a List

```
my_list <- list("apple", "banana", "cherry")
lengths_list <- lapply(my_list, nchar)
print(lengths_list)
```

25. Creating an Empty List and Populating It

```
empty_list <- list()
empty_list[[1]] <- "first element"
empty_list[[2]] <- "second element"
print(empty_list)
```

Matrix

1. Creating a Matrix

```
my_matrix <- matrix(1:9, nrow = 3, ncol = 3)
print(my_matrix)
```

2. Creating a Matrix by Row Binding

```
row1 <- c(1, 2, 3)
row2 <- c(4, 5, 6)
my_matrix <- rbind(row1, row2)
print(my_matrix)
```

3. Creating a Matrix by Column Binding

```
col1 <- c(1, 2, 3)
col2 <- c(4, 5, 6)
my_matrix <- cbind(col1, col2)
print(my_matrix)
```

4. Accessing an Element in a Matrix

```
my_matrix <- matrix(1:9, nrow = 3, ncol = 3)
element <- my_matrix[2, 3]
print(element)
```

5. Accessing an Entire Row in a Matrix

```
my_matrix <- matrix(1:9, nrow = 3, ncol = 3)
row <- my_matrix[2, ]
print(row)
```

6. Accessing an Entire Column in a Matrix

```
my_matrix <- matrix(1:9, nrow = 3, ncol = 3)
column <- my_matrix[, 2]
print(column)
```

7. Modifying an Element in a Matrix

```
my_matrix <- matrix(1:9, nrow = 3, ncol = 3)
my_matrix[2, 3] <- 10
print(my_matrix)
```

8. Matrix Addition

```
matrix1 <- matrix(1:4, nrow = 2, ncol = 2)
matrix2 <- matrix(5:8, nrow = 2, ncol = 2)
result_matrix <- matrix1 + matrix2
print(result_matrix)
```

9. Matrix Subtraction

```
matrix1 <- matrix(5:8, nrow = 2, ncol = 2)
matrix2 <- matrix(1:4, nrow = 2, ncol = 2)
result_matrix <- matrix1 - matrix2
print(result_matrix)
```

10. Matrix Multiplication

```
matrix1 <- matrix(1:4, nrow = 2, ncol = 2)
matrix2 <- matrix(5:8, nrow = 2, ncol = 2)
result_matrix <- matrix1 * matrix2
print(result_matrix)
```

11. Matrix Division

```
matrix1 <- matrix(10:13, nrow = 2, ncol = 2)
matrix2 <- matrix(2:5, nrow = 2, ncol = 2)
result_matrix <- matrix1 / matrix2
print(result_matrix)
```

12. Matrix Multiplication (Dot Product)

```
matrix1 <- matrix(1:4, nrow = 2, ncol = 2)
matrix2 <- matrix(5:8, nrow = 2, ncol = 2)
result_matrix <- matrix1 %*% matrix2
print(result_matrix)
```

13. Transposing a Matrix

```
my_matrix <- matrix(1:6, nrow = 2, ncol = 3)
transposed_matrix <- t(my_matrix)
print(transposed_matrix)
```

14. Creating an Identity Matrix

```
identity_matrix <- diag(3)
print(identity_matrix)
```

15. Finding the Dimensions of a Matrix

```
my_matrix <- matrix(1:9, nrow = 3, ncol = 3)
dimensions <- dim(my_matrix)
```

```
print(dimensions)
```

16. Calculating the Row Sums of a Matrix

```
my_matrix <- matrix(1:9, nrow = 3, ncol = 3)
row_sums <- rowSums(my_matrix)
print(row_sums)
```

17. Calculating the Column Sums of a Matrix

```
my_matrix <- matrix(1:9, nrow = 3, ncol = 3)
col_sums <- colSums(my_matrix)
print(col_sums)
```

18. Calculating the Row Means of a Matrix

```
my_matrix <- matrix(1:9, nrow = 3, ncol = 3)
row_means <- rowMeans(my_matrix)
print(row_means)
```

19. Calculating the Column Means of a Matrix

```
my_matrix <- matrix(1:9, nrow = 3, ncol = 3)
col_means <- colMeans(my_matrix)
print(col_means)
```

20. Extracting the Diagonal of a Matrix

```
my_matrix <- matrix(1:9, nrow = 3, ncol = 3)
diagonal <- diag(my_matrix)
print(diagonal)
```

21. Setting the Diagonal of a Matrix

```r
my_matrix <- matrix(1:9, nrow = 3, ncol = 3)
diag(my_matrix) <- c(10, 20, 30)
print(my_matrix)
```

22. Checking if a Matrix is Symmetric

```r
symmetric_matrix <- matrix(c(1, 2, 2, 1), nrow = 2)
is_symmetric <- isSymmetric(symmetric_matrix)
print(is_symmetric)
```

23. Flattening a Matrix to a Vector

```r
my_matrix <- matrix(1:9, nrow = 3, ncol = 3)
flattened_vector <- as.vector(my_matrix)
print(flattened_vector)
```

24. Finding the Inverse of a Matrix

```r
my_matrix <- matrix(c(4, 7, 2, 6), nrow = 2, ncol = 2)
inverse_matrix <- solve(my_matrix)
print(inverse_matrix)
```

25. Creating a Matrix with Random Numbers

```r
random_matrix <- matrix(runif(9), nrow = 3, ncol = 3)
print(random_matrix)
```

Data Frame

1. Creating a Simple Data Frame

```r
my_df <- data.frame(Name = c("John", "Jane"), Age = c(30, 25))
```

```
print(my_df)
```

2. Accessing a Column in a Data Frame

```
my_df <- data.frame(Name = c("John", "Jane"), Age = c(30, 25))
age_column <- my_df$Age
print(age_column)
```

3. Accessing a Row in a Data Frame

```
my_df <- data.frame(Name = c("John", "Jane"), Age = c(30, 25))
first_row <- my_df[1, ]
print(first_row)
```

4. Accessing a Specific Element in a Data Frame

```
my_df <- data.frame(Name = c("John", "Jane"), Age = c(30, 25))
element <- my_df[2, "Name"]
print(element)
```

5. Adding a New Column to a Data Frame

```
my_df <- data.frame(Name = c("John", "Jane"), Age = c(30, 25))
my_df$Gender <- c("Male", "Female")
print(my_df)
```

6. Adding a New Row to a Data Frame

```
my_df <- data.frame(Name = c("John", "Jane"), Age = c(30, 25))
new_row <- data.frame(Name = "Doe", Age = 28)
my_df <- rbind(my_df, new_row)
print(my_df)
```

7. Removing a Column from a Data Frame

```
my_df <- data.frame(Name = c("John", "Jane"), Age = c(30, 25), Gender
= c("Male", "Female"))
my_df$Gender <- NULL
print(my_df)
```

8. Renaming Columns in a Data Frame

```
my_df <- data.frame(Name = c("John", "Jane"), Age = c(30, 25))
names(my_df) <- c("FullName", "Years")
print(my_df)
```

9. Filtering Rows in a Data Frame

```
my_df <- data.frame(Name = c("John", "Jane", "Doe"), Age = c(30, 25,
40))
filtered_df <- my_df[my_df$Age > 30, ]
print(filtered_df)
```

10. Sorting a Data Frame by a Column

```
my_df <- data.frame(Name = c("John", "Jane", "Doe"), Age = c(30, 25,
40))
sorted_df <- my_df[order(my_df$Age), ]
print(sorted_df)
```

11. Merging Two Data Frames

```
df1 <- data.frame(ID = c(1, 2), Name = c("John", "Jane"))
df2 <- data.frame(ID = c(1, 2), Age = c(30, 25))
merged_df <- merge(df1, df2, by = "ID")
print(merged_df)
```

12. Selecting Specific Columns in a Data Frame

```
my_df <- data.frame(Name = c("John", "Jane"), Age = c(30, 25), Gender
= c("Male", "Female"))
selected_df <- my_df[, c("Name", "Age")]
print(selected_df)
```

13. Calculating Summary Statistics for a Data Frame

```
my_df <- data.frame(Name = c("John", "Jane", "Doe"), Age = c(30, 25,
40))
summary_stats <- summary(my_df)
print(summary_stats)
```

14. Applying a Function to Each Column in a Data Frame

```
my_df <- data.frame(A = c(1, 2, 3), B = c(4, 5, 6))
sum_columns <- lapply(my_df, sum)
print(sum_columns)
```

15. Converting a Data Frame Column to a Different Data Type

```
my_df <- data.frame(Name = c("John", "Jane"), Age = c("30", "25"))
my_df$Age <- as.numeric(my_df$Age)
print(my_df)
```

16. Creating a Data Frame from a Matrix

```
my_matrix <- matrix(1:9, nrow = 3, ncol = 3)
my_df <- as.data.frame(my_matrix)
print(my_df)
```

17. Combining Data Frames by Rows

```
df1 <- data.frame(Name = c("John", "Jane"), Age = c(30, 25))
df2 <- data.frame(Name = c("Doe", "Smith"), Age = c(28, 32))
combined_df <- rbind(df1, df2)
print(combined_df)
```

18. Combining Data Frames by Columns

```
df1 <- data.frame(Name = c("John", "Jane"))
df2 <- data.frame(Age = c(30, 25))
combined_df <- cbind(df1, df2)
print(combined_df)
```

19. Checking for Missing Values in a Data Frame

```
my_df <- data.frame(Name = c("John", NA, "Doe"), Age = c(30, 25, NA))
missing_values <- is.na(my_df)
print(missing_values)
```

20. Replacing Missing Values in a Data Frame

```
my_df <- data.frame(Name = c("John", NA, "Doe"), Age = c(30, 25, NA))
my_df[is.na(my_df)] <- "Unknown"
print(my_df)
```

21. Calculating the Mean of a Data Frame Column

```
my_df <- data.frame(Name = c("John", "Jane", "Doe"), Age = c(30, 25,
40))
mean_age <- mean(my_df$Age)
print(mean_age)
```

22. Converting a Data Frame to a Matrix

```
my_df <- data.frame(A = c(1, 2, 3), B = c(4, 5, 6))
```

```
my_matrix <- as.matrix(my_df)
print(my_matrix)
```

23. Subsetting a Data Frame by Conditions

```
my_df <- data.frame(Name = c("John", "Jane", "Doe"), Age = c(30, 25,
40))
subset_df <- subset(my_df, Age > 30)
print(subset_df)
```

24. Grouping Data Frame by a Column and Summarizing

```
my_df <- data.frame(Name = c("John", "Jane", "Doe", "Smith"), Gender =
c("Male", "Female", "Male", "Male"), Age = c(30, 25, 40, 22))
grouped_summary <- aggregate(Age ~ Gender, data = my_df, FUN = mean)
print(grouped_summary)
```

25. Removing Duplicate Rows from a Data Frame

```
my_df <- data.frame(Name = c("John", "Jane", "Doe", "John"), Age =
c(30, 25, 40, 30))
unique_df <- unique(my_df)
print(unique_df)
```

Factor

1. Creating a Simple Factor

```
gender <- factor(c("Male", "Female", "Male", "Female"))
print(gender)
```

2. Checking the Levels of a Factor

```
gender <- factor(c("Male", "Female", "Male", "Female"))
levels_gender <- levels(gender)
print(levels_gender)
```

3. Creating an Ordered Factor

```
sizes <- factor(c("Small", "Medium", "Large", "Medium"), ordered =
TRUE)
print(sizes)
```

4. Changing the Order of Factor Levels

```
sizes <- factor(c("Small", "Medium", "Large"), levels = c("Large",
"Medium", "Small"))
print(sizes)
```

5. Converting a Factor to a Numeric Vector

```
numeric_factor <- factor(c("1", "2", "3"))
numeric_vector <- as.numeric(as.character(numeric_factor))
print(numeric_vector)
```

6. Converting a Factor to a Character Vector

```
gender <- factor(c("Male", "Female", "Male"))
character_vector <- as.character(gender)
print(character_vector)
```

7. Changing the Levels of a Factor

```
gender <- factor(c("M", "F", "M"), levels = c("M", "F"))
levels(gender) <- c("Male", "Female")
print(gender)
```

8. Counting the Number of Levels in a Factor

```
gender <- factor(c("Male", "Female", "Male", "Female"))
num_levels <- nlevels(gender)
print(num_levels)
```

9. Creating a Factor with Specific Levels

```
responses <- factor(c("Yes", "No", "No"), levels = c("Yes", "No",
"Maybe"))
print(responses)
```

10. Creating a Factor from a Character Vector

```
character_vector <- c("Low", "Medium", "High", "Medium", "Low")
priority <- factor(character_vector)
print(priority)
```

11. Dropping Unused Levels from a Factor

```
factors <- factor(c("Red", "Blue", "Green"))
factors <- droplevels(factors[factors != "Green"])
print(factors)
```

12. Releveling a Factor to Change the Reference Level

```
sizes <- factor(c("Small", "Medium", "Large"), levels = c("Small",
"Medium", "Large"))
sizes <- relevel(sizes, ref = "Large")
print(sizes)
```

13. Combining Levels of a Factor

```
education <- factor(c("High School", "Bachelor's", "Master's", "PhD"))
```

```
levels(education) <- c("High School", "Undergraduate", "Graduate",
"Graduate")
print(education)
```

14. Summarizing a Factor

```
sizes <- factor(c("Small", "Medium", "Large", "Medium", "Small"))
summary_sizes <- summary(sizes)
print(summary_sizes)
```

15. Creating a Factor with Labels

```
grades <- factor(c("A", "B", "A", "C"), labels = c("Excellent",
"Good", "Average"))
print(grades)
```

16. Checking for Specific Levels in a Factor

```
factors <- factor(c("Apple", "Banana", "Cherry"))
has_banana <- "Banana" %in% levels(factors)
print(has_banana)
```

17. Finding the Mode of a Factor

```
responses <- factor(c("Yes", "No", "Yes", "Yes", "No"))
mode_response <- levels(responses)[which.max(summary(responses))]
print(mode_response)
```

18. Splitting a Factor into a List Based on Levels

```
gender <- factor(c("Male", "Female", "Male", "Female"))
split_list <- split(1:length(gender), gender)
print(split_list)
```

19. Converting a Factor to a Matrix

```
gender <- factor(c("Male", "Female", "Male"))
gender_matrix <- model.matrix(~ gender - 1)
print(gender_matrix)
```

20. Ordering a Data Frame by a Factor

```
df <- data.frame(Name = c("John", "Jane", "Doe"), Education =
factor(c("PhD", "Bachelor's", "Master's")))
ordered_df <- df[order(df$Education), ]
print(ordered_df)
```

21. Creating a Factor from a Numeric Vector

```
numbers <- c(1, 2, 3, 1, 2)
factor_numbers <- factor(numbers)
print(factor_numbers)
```

22. Subsetting a Data Frame by a Factor Level

```
df <- data.frame(Name = c("John", "Jane", "Doe"), Gender =
factor(c("Male", "Female", "Male")))
subset_df <- df[df$Gender == "Male", ]
print(subset_df)
```

23. Using Factors in a Plot

```
education <- factor(c("High School", "Bachelor's", "Master's", "PhD"))
barplot(table(education))
```

24. Creating a Factor from a Continuous Variable

```
age <- c(23, 35, 44, 27, 51)
```

```
age_groups <- cut(age, breaks = c(20, 30, 40, 50, 60), labels =
c("20-29", "30-39", "40-49", "50-59"))
print(age_groups)
```

25. Comparing Two Factors

```
factor1 <- factor(c("A", "B", "C"))
factor2 <- factor(c("A", "B", "D"))
comparison <- factor1 == factor2
print(comparison)
```