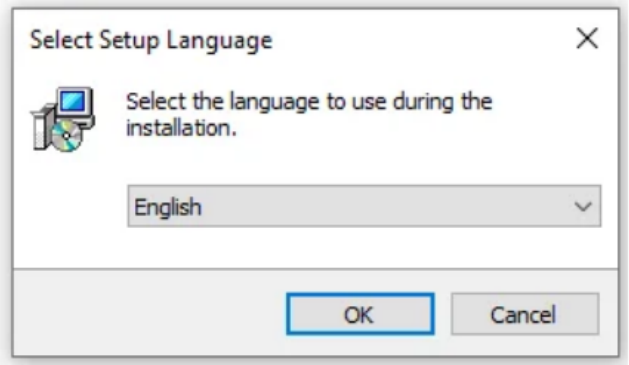
Table of Contents

Learning Objectives

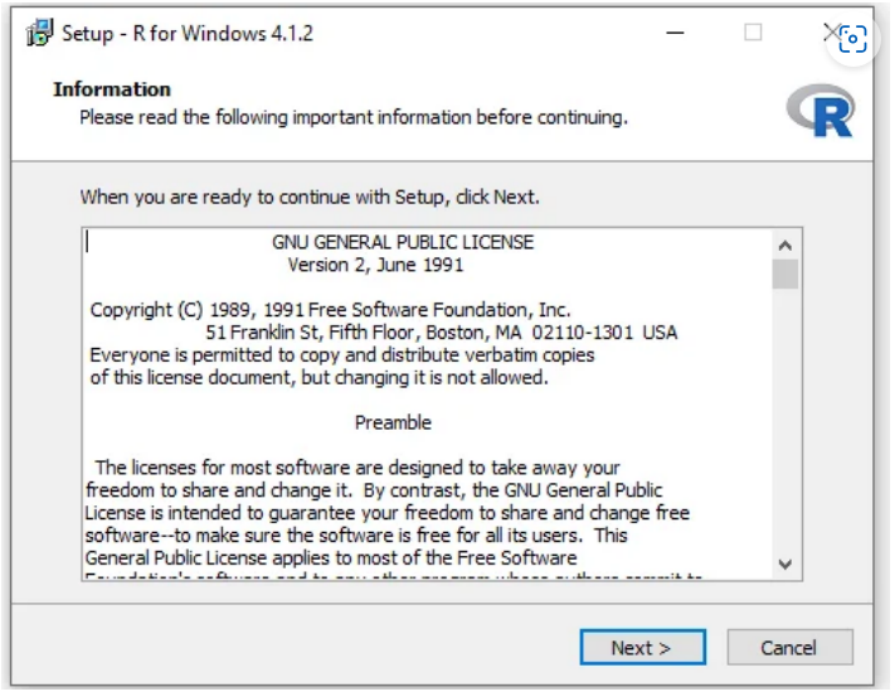
- Understand R
- How to manipulate data with R
- Statistics in R
- Graphing in R
- Concept of Clustering and Data Modelling

1. Installing R

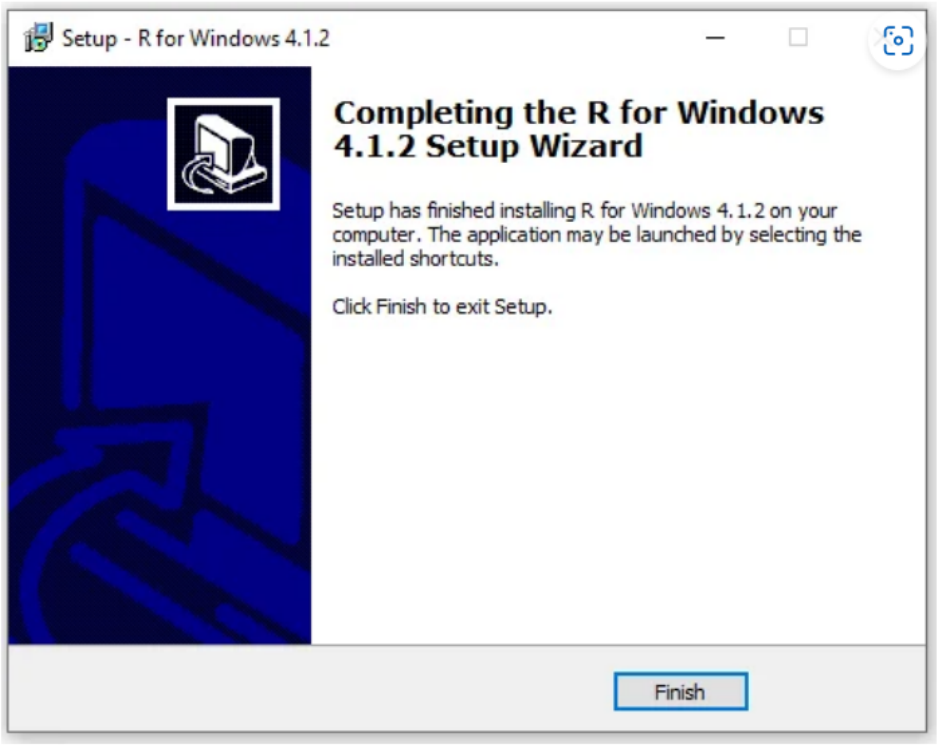
1. To install R on Windows OS:
    1. Go to the CRAN website.
    2. Click on "Download R for Windows".
    3. Click on "install R for the first time" link to download the R executable (.exe) file.
    4. Run the R executable file to start installation, and allow the app to make changes to your device.
    5. Select the installation language.



6. Follow the installation instructions.



7. Click on "Finish" to exit the installation setup.


Setup - R for Windows 4.1.2
Completing the R for Windows 4.1.2 Setup Wizard
Setup has finished installing R for Windows 4.1.2 on your computer. The application may be launched by selecting the installed shortcuts.
Click Finish to exit Setup.
Finish

R Programming

1. Concept of R

R is a programming language and environment specifically designed for statistical computing and graphics. It was developed by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, and first released in 1995. R has since gained immense popularity and is widely used by statisticians, data scientists, researchers, and analysts for data analysis, statistical modeling, visualization, and machine learning tasks.

One of the key strengths of R is its extensive collection of packages and libraries. These packages provide a vast range of functions and tools for various statistical and data analysis tasks. The Comprehensive R Archive Network (CRAN) hosts thousands of packages contributed by the R community, making it a rich resource for users to find and utilize specialized functionality.

R offers a flexible and expressive syntax that allows users to manipulate, analyze, and visualize data efficiently. It provides a wide range of statistical techniques, such as linear and nonlinear modeling, time series analysis, clustering, and hypothesis testing. With R, you can import and export data from different file formats, handle missing values, perform data cleaning and preprocessing, and generate high-quality visualizations using powerful plotting capabilities.

Furthermore, R is an interpreted language, which means that it executes code interactively, making it ideal for exploratory data analysis. It also supports functional programming paradigms, enabling users to write concise and efficient code. R's interactive nature, combined with its vast array of statistical functions and visualization capabilities, makes it a popular choice for data analysis tasks.

R is an open-source language, released under the GNU General Public License (GPL), which means it is freely available for anyone to use, modify, and distribute. This open-source nature has fostered a large and active community around R, resulting in continuous development and improvement of the language and its packages.

In summary, R is a powerful programming language and environment specifically designed for statistical computing and data analysis. Its wide range of statistical functions, extensive package ecosystem, and interactive nature make it a popular choice for professionals working with data in various fields.

2. Data Types in R

R is a programming language widely used for statistical analysis, data manipulation, and visualization. In R, there are several built-in data types that allow you to represent different kinds of information. Here are the commonly used data types in R with examples:

2.1. Numeric:

Numeric data type represents real numbers. It includes both integers and floating-point numbers.

```
x <- 10  # integer
y <- 3.14  # floating-point number
```

2.2. Integer:

Integer data type represents whole numbers without any fractional parts.

```
age <- 25
```

2.3. Character:
Character data type represents text and is enclosed in quotes (either single or double).

```
name <- "John"
```

2.4. Logical:
Logical data type represents Boolean values, either `TRUE` or `FALSE`, which are used for logical operations and conditional statements.

```
isTrue <- TRUE
```

2.5. Complex:
Complex data type represents complex numbers with real and imaginary parts.
Example:

```
z <- 2 + 3i
```

2.6. Raw:
Raw data type represents a sequence of bytes.

```
bytes <- charToRaw("Hello")
```

2.7. Factor:
Factor data type represents categorical variables with a fixed set of values known as levels. It is useful for statistical modeling and analysis.

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

2.8. Date and Time:
R provides specific data types to handle dates and times, such as `Date`, `POSIXct`, and `POSIXlt`.

```
date <- as.Date("2023-06-06")
time <- as.POSIXct("2023-06-06 12:34:56")
```

These are some of the commonly used data types in R. Understanding and utilizing these data types correctly will enable you to manipulate and analyze data effectively in R.

CHECK YOUR PROGRESS

1. Which data type in R represents categorical variables with a fixed set of values?

a) Numeric
b) Integer
c) Character
d) Factor

2. How do you assign a character value to a variable in R?
a) name <- "John"
b) name <- John
c) name = "John"
d) name = John

2. Vectors in R

In R, vectors are one-dimensional arrays that can hold multiple elements of the same data type. They are a fundamental data structure used in R for storing and manipulating data. Vectors can be created using various methods, and their elements can be accessed and modified using indexing.

3.1. Create Numeric Vector

```
#Creating a numeric vector
numeric_vector <- c(1.5, 2.7, 3.2, 4.9)
print(numeric_vector)
```

3.2. Creating Character Vector

```
# Creating a character vector
character_vector <- c("apple", "banana", "orange")
print(character_vector)
```

3.3. Accessing Elements of a Vector

```
# Accessing elements of a vector
numeric_vector <- c(1.5, 2.7, 3.2, 4.9)
print(numeric_vector[2])  # Accessing the second element
```

3.4. Modifying elements of vector

```
# Modifying elements of a vector
numeric_vector <- c(1.5, 2.7, 3.2, 4.9)
numeric_vector[3] <- 5.1  # Modifying the third element
print(numeric_vector)
```

3.5. Vector Arithmetic

```
# Vector arithmetic
vector1 <- c(1, 2, 3)
vector2 <- c(4, 5, 6)
vector_sum <- vector1 + vector2  # Element-wise addition
vector_product <- vector1 * vector2  # Element-wise multiplication
print(vector_sum)
print(vector_product)
```

3.6. Vector Inbuilt Function

```
# Vector functions
numeric_vector <- c(1.5, 2.7, 3.2, 4.9)
vector_length <- length(numeric_vector)  # Length of the vector
vector_sum <- sum(numeric_vector)  # Sum of all elements
vector_mean <- mean(numeric_vector)  # Mean of all elements
vector_max <- max(numeric_vector)  # Maximum value
print(vector_length)
print(vector_sum)
print(vector_mean)
print(vector_max)
```

3.7. Vector Subsetting

```
# Vector subsetting
numeric_vector <- c(1, 2, 3, 4, 5)
subset_vector <- numeric_vector[c(2, 4)]  # Selecting elements at index 2
and 4
print(subset_vector)
```

3.8. Vector Concatenation

```
# Vector concatenation
vector1 <- c(1, 2, 3)
vector2 <- c(4, 5, 6)
concatenated_vector <- c(vector1, vector2)  # Concatenating two vectors
print(concatenated_vector)
```

3.9. Vector Comparison

```
# Vector comparison
vector1 <- c(1, 2, 3)
vector2 <- c(2, 4, 6)
comparison_vector <- vector1 > vector2  # Element-wise comparison
print(comparison_vector)
```

3.10. Vector Sorting

```
# Vector sorting
numeric_vector <- c(5, 2, 8, 1, 6)
sorted_vector <- sort(numeric_vector)  # Sorting in ascending order
print(sorted_vector)
```

3.11. Vector Logical Operations

```
# Vector sorting
numeric_vector <- c(5, 2, 8, 1, 6)
sorted_vector <- sort(numeric_vector)  # Sorting in ascending order
print(sorted_vector)
```

CHECK YOUR PROGRESS

Question 3:
What is the output of the following code?

```
numeric_vector <- c(1.5, 2.7, 3.2, 4.9)
vector_max <- max(numeric_vector)
print(vector_max)

a) 4.9
b) 3.2
c) 2.7
d) 1.5
```

Question 4:
What is the result of concatenating vector1 and vector2 in the following code?

```
vector1 <- c(1, 2, 3)
vector2 <- c(4, 5, 6)
concatenated_vector <- c(vector1, vector2)
print(concatenated_vector)

a) c(1, 2, 3, 4, 5, 6)
b) c(1, 4, 2, 5, 3, 6)
c) c(4, 5, 6, 1, 2, 3)
d) c(1, 2, 3)
```

Activity Question 1:

You are given a numeric vector `numeric_vector <- c(1, 2, 3, 4, 5)`. Perform the following operations on the vector:

1. Modify the third element of the vector to 6.
2. Calculate the sum of all the elements in the vector.
3. Find the maximum value in the vector.
4. Select the elements at index 2 and 4 and create a subset vector.
5. Concatenate the vector with itself.

4. List in R

In R, a list is a data structure that can hold elements of different types. It is a collection of objects or values, where each element is identified by a name or index. Lists allow you to store and organize related data in a flexible and hierarchical manner. Here's an explanation of lists in R with examples:

4.1. Creating a List:

To create a list in R, you can use the list() function. Here's an example:

```
# Creating a list
my_list <- list(name = "John", age = 30, city = "New York")
print(my_list)
```

In this example, we create a list called my_list containing three elements: name, age, and city. Each element is assigned a value.

4.2. Accessing elements of a list

You can access elements in a list using their names or indices. To access an element by name, use the $ operator. To access an element by index, use the square brackets []. Here are a few examples:

```
# Accessing elements
print(my_list$name)
print(my_list[2])
```

4.3. Adding Elements:
You can add new elements to a list using the $ operator or by assigning values to new names. Here's an example:

```
# Adding elements
my_list$occupation <- "Engineer"
my_list["gender"] <- "Male"
print(my_list)
```

In this example, we add two new elements to the list: occupation and gender, along with their respective values.

4.4. Nested Lists:
Lists in R can also contain other lists as elements. This allows you to create nested or hierarchical structures. Here's an example:

```
# Nested lists
nested_list <- list(person = my_list, interests = c("Reading", "Cooking"))
print(nested_list)
```

Check Your Progress

5. To create a list in R, you can use the _____ function.
6. To access an element in a list by name, you can use the _____ operator.

5. Data Frames

R, data frames are widely used for storing and manipulating data. They are two-dimensional structures that can hold different types of data, such as numbers, characters, factors, and logical values. Data frames are similar to tables in a relational database or spreadsheets.

Example 1: Creating a Data Frame

```
# Creating a data frame from vectors
name <- c("John", "Alice", "Bob", "Emily")
age <- c(25, 30, 28, 35)
city <- c("New York", "Paris", "London", "Sydney")

# Combining the vectors into a data frame
df <- data.frame(Name = name, Age = age, City = city)

# Printing the data frame
print(df)
```

Example 2: Accessing Data in a Data Frame

```
# Accessing individual columns
print(df$Name)
print(df$Age)
print(df$City)

# Accessing specific rows and columns
print(df[1,])         # First row
print(df[, "Age"])    # Age column

# Accessing subsets of the data frame
subset_df <- subset(df, Age > 28)
print(subset_df)
```

Example 3: Adding and Removing Columns

```
# Adding a new column
df$Country <- c("USA", "France", "UK", "Australia")
print(df)

# Removing a column
df <- subset(df, select = -City)
print(df)
```

Example 4: Filtering and Sorting Data Frames

```
# Filtering the data frame based on a condition
filtered_df <- df[df$Age > 28, ]
print(filtered_df)

# Sorting the data frame by a specific column
sorted_df <- df[order(df$Age), ]
print(sorted_df)
```

Example 5: Performing Aggregations and Summary Statistics

```
# Calculating the mean age
mean_age <- mean(df$Age)
print(mean_age)

# Computing summary statistics for the data frame
summary_df <- summary(df)
print(summary_df)
```

CHECK YOUR PROGRESS

1. In R, data frames are two-dimensional structures that can hold different types of data. They are similar to tables in a relational database or spreadsheets. To access individual columns in a data frame, you can use the syntax `print(df$_____)`. Fill in the blank to access the "City" column in the data frame `df`.

2. In R, data frames are widely used for storing and manipulating data. To create a new column in a data frame, you can use the syntax `df$NewColumn <- c(_____)`, where "NewColumn" is the name of the new column you want to create. Fill in the blank with the values you would assign to the "Country" column in the data frame `df` to assign "USA", "France", "UK", and "Australia" respectively to the column.

Activity Question 2

Instructions:
Based on the given context about data frames in R, perform the following activities:

1. Create a data frame named `students` with the following columns:

- Column 1: "Name" - containing the names of five students: "Tom", "Emma", "Liam", "Olivia", "Noah".
- Column 2: "Age" - containing the ages of the students: 18, 20, 19, 21, 22.
- Column 3: "City" - containing the cities where the students reside: "New York", "London", "Paris", "Sydney", "Tokyo".

Print the resulting data frame.

2. Access and print the "Age" column of the `students` data frame.

3. Remove the "City" column from the `students` data frame and print the updated data frame.

4. Filter the `students` data frame to only include the students whose age is greater than or equal to 20. Print the filtered data frame.

5. Calculate the average age of the students in the `students` data frame and print the result.

6. Arrays in R

In R, an array is a data structure that can store multiple values of the same data type. It allows you to organize data in a multidimensional format, making it suitable for working with large datasets or matrices. An array in R can have one, two, or more dimensions.

Here's an explanation of arrays in R with examples:

1. One-dimensional array:

A one-dimensional array in R is similar to a vector. It can store multiple elements of the same data type in a single row or column. Here's an example of creating a one-dimensional array:

```
# Creating a one-dimensional array
my_array <- c(10, 20, 30, 40, 50)
print(my_array)
```

2. Two-dimensional array:

A two-dimensional array in R is similar to a matrix. It has rows and columns and can be thought of as a table with values organized in rows and columns. Here's an example of creating a two-dimensional array:

```
# Creating a two-dimensional array
my_array <- array(c(1, 2, 3, 4, 5, 6), dim = c(2, 3))
print(my_array)
```

3. Three-dimensional array:

A three-dimensional array in R extends the concept of a two-dimensional array. It adds an additional dimension, allowing you to store data in a cube-like structure. Here's an example of creating a three-dimensional array:

```
# Creating a three-dimensional array
my_array <- array(1:24, dim = c(2, 3, 4))
print(my_array)
```

4. Creating an array from existing vectors:

You can create an array by combining existing vectors using the array() function. Here's an example:

```
# Creating an array from existing vectors
vector1 <- c(1, 2, 3)
vector2 <- c(4, 5, 6)
vector3 <- c(7, 8, 9)

my_array <- array(c(vector1, vector2, vector3), dim = c(3, 3, 2))
print(my_array)
```

5. Performing operations on arrays:
Arrays in R can be used for performing mathematical operations efficiently across multiple elements. Here's an example of adding two arrays element-wise:

```
# Performing operations on arrays
array1 <- array(1:9, dim = c(3, 3))
array2 <- array(10:18, dim = c(3, 3))

result_array <- array1 + array2
print(result_array)
```

CHECK YOUR PROGRESS

9. What is a one-dimensional array in R?

A) An array that can store multiple values of different data types.
B) An array that can store multiple elements of the same data type in a single row or column.
C) An array that can store multiple elements of different data types in a grid-like structure.
D) An array that can store multiple rows and columns of data in a tabular format.

10. How many dimensions can an array in R have?

A) One dimension
B) Two dimensions
C) Three dimensions
D) Any number of dimensions

7. Statistical Commands in R

Statistical commands in R are essential for analyzing and interpreting data. R provides a comprehensive set of built-in functions and libraries specifically designed for statistical analysis. These commands enable users to perform various statistical operations, ranging from basic descriptive statistics to advanced modeling techniques. Here are a few examples of statistical commands in R:

1. Mean Calculation:

```
# Create a vector
data <- c(10, 20, 30, 40, 50)

# Calculate the mean
mean_value <- mean(data)
print(mean_value)
```

2. Median Calculation:

```
# Create a vector
data <- c(10, 20, 30, 40, 50)

# Calculate the median
median_value <- median(data)
print(median_value)
```

3. Standard Deviation Calculation:

```
# Create a vector
data <- c(10, 20, 30, 40, 50)

# Calculate the standard deviation
sd_value <- sd(data)
print(sd_value)
```

4. Correlation Calculation:

```
# Create two vectors
x <- c(1, 2, 3, 4, 5)
y <- c(10, 20, 30, 40, 50)

# Calculate the correlation
cor_value <- cor(x, y)
print(cor_value)
```

5. Linear Regression:

```
# Create two vectors
x <- c(1, 2, 3, 4, 5)
y <- c(10, 20, 30, 40, 50)

# Perform linear regression
model <- lm(y ~ x)

# Print the coefficients
print(coefficients(model))
```

These are just a few examples of statistical commands in R. R is a powerful statistical programming language with a wide range of functions and packages available for various statistical analyses.

Check Your Progress

1. Which command in R is used to calculate the mean of a given vector?

a) median()
b) sd()
c) cor()

d) mean()

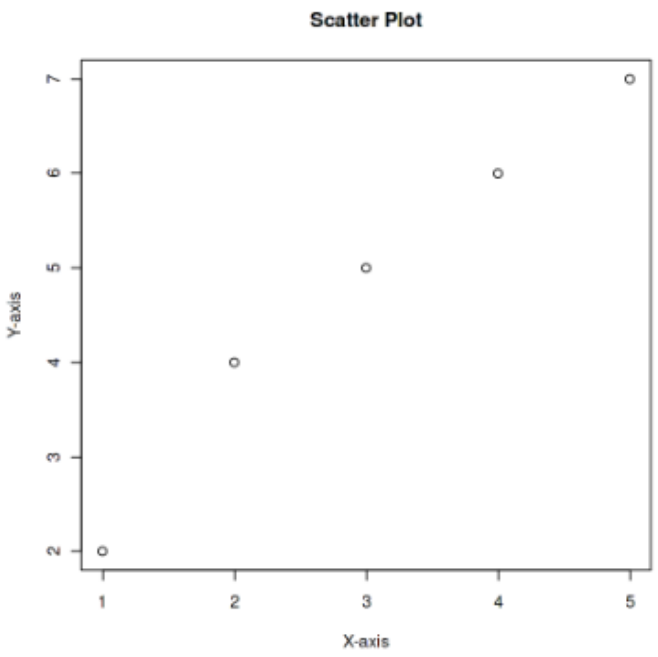
2. In the context of statistical analysis in R, which command is used to perform linear regression?

a) median()
b) lm()
c) sd()
d) cor()

8. Graphs in R

1. Scatter Plot:
A scatter plot is useful for visualizing the relationship between two numeric variables.

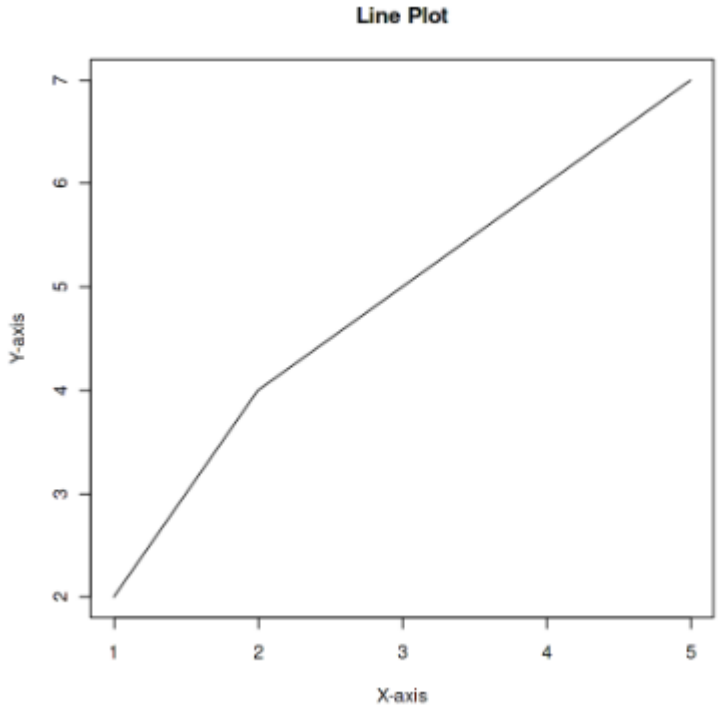
```
# Example data
x <- c(1, 2, 3, 4, 5)
y <- c(2, 4, 5, 6, 7)

# Creating a scatter plot
plot(x, y, main = "Scatter Plot", xlab = "X-axis", ylab = "Y-axis")
```



2. Line Plot:

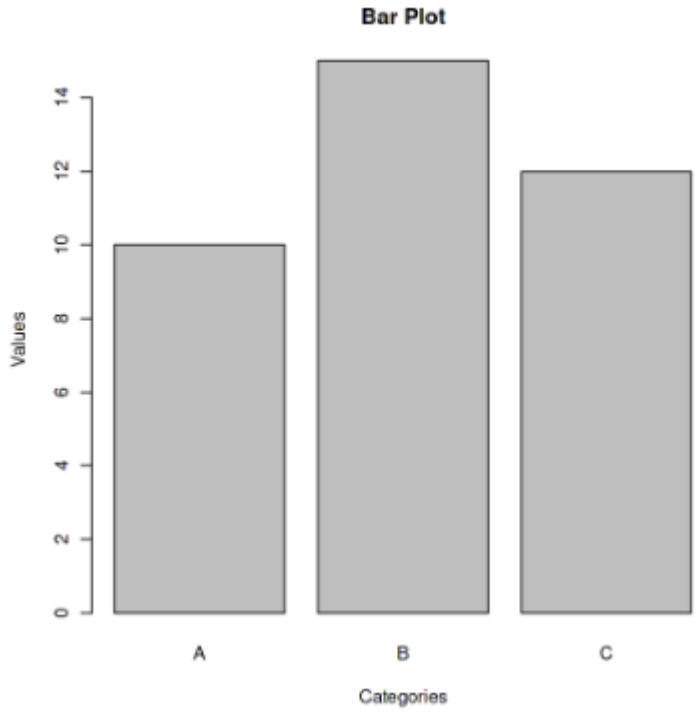
A line plot is commonly used to display trends over time or ordered data points.

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

# Creating a line plot
plot(x, y, type = "l", main = "Line Plot", xlab = "X-axis", ylab = "Y-axis")
```



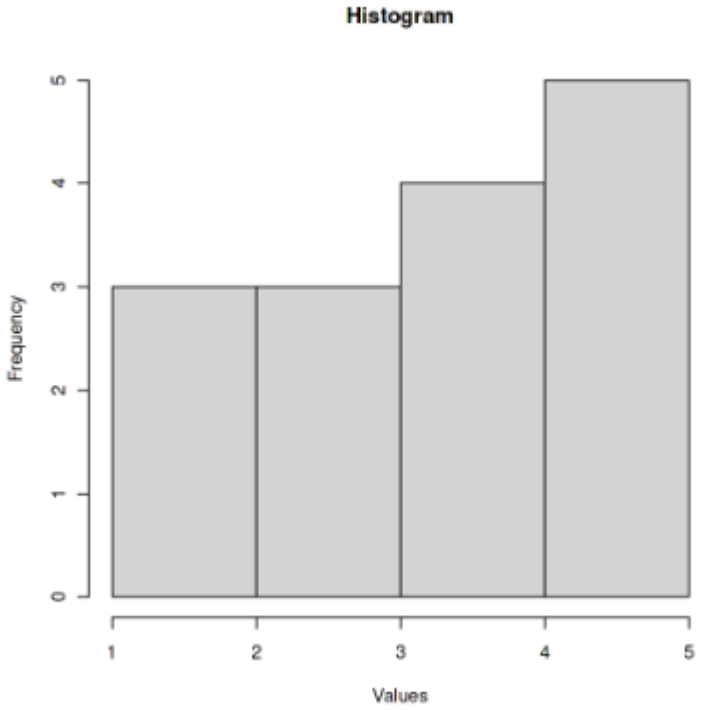
3. Bar Plot:
A bar plot is suitable for comparing categorical variables.

```
# Example data
categories <- c("A", "B", "C")
values <- c(10, 15, 12)

# Creating a bar plot
barplot(values, names.arg = categories, main = "Bar Plot", xlab =
"Categories", ylab = "Values")
```



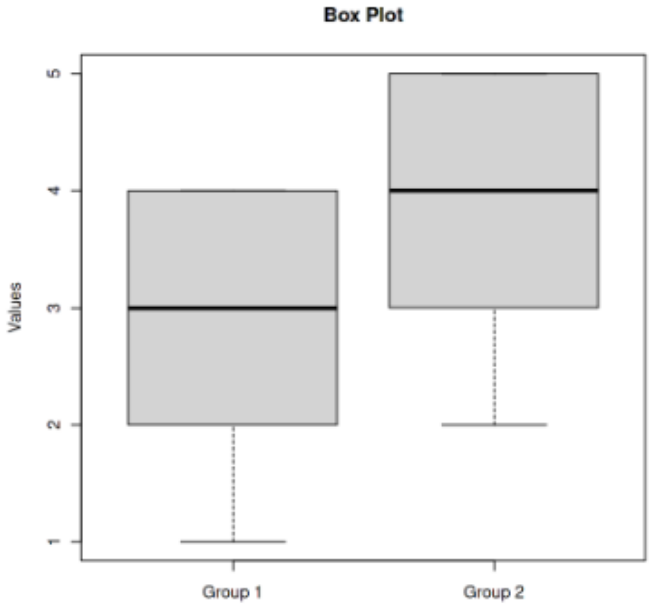
4. Histogram:
A histogram displays the distribution of a numeric variable by dividing it into bins.

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

# Creating a histogram
hist(x, main = "Histogram", xlab = "Values", ylab = "Frequency")
```



5. Box Plot:
A box plot is useful for visualizing the distribution and identifying outliers.

```
# Example data
group1 <- c(1, 2, 2, 3, 3, 3, 4, 4, 4, 4)
group2 <- c(2, 3, 3, 4, 4, 4, 5, 5, 5, 5)

# Creating a box plot
boxplot(group1, group2, names = c("Group 1", "Group 2"), main = "Box Plot",
ylab = "Values")
```



These are just a few examples of the types of graphs you can create in R. You can customize the plots further by exploring additional options and parameters available in R's plotting functions.

Check Your Progress

1. You have a dataset containing information about the sales of different products in three different regions (Region A, Region B, and Region C). You want to visualize the sales data using an appropriate graph in R. Which type of plot would be most suitable for this task, considering the nature of the data?

a) Scatter plot
b) Line plot
c) Bar plot
d) Histogram
e) Box plot

9. Data Tables in R

Data tables in R are a data structure that is designed for efficient data manipulation. They are based on the data.frame data structure, but they offer a number of additional features that make them faster and more flexible.

Some of the key features of data tables include:

Fast indexing: Data tables can be indexed using a variety of criteria, including column names, row numbers, and logical expressions. This makes it possible to quickly subset data tables.

Fast joins: Data tables can be joined together using a variety of join types, including inner joins, outer joins, and semi joins. This makes it possible to combine data from multiple tables.

Fast aggregation: Data tables can be aggregated using a variety of aggregation functions, including mean, sum, and max. This makes it possible to summarize data quickly.

Fast updating: Data tables can be updated using a variety of update operators, including := and [<-. This makes it possible to modify data quickly.

Data tables are a powerful tool for data manipulation in R. They are faster and more flexible than data.frames, and they offer a number of features that make them well-suited for a variety of data analysis tasks.

Here are some of the benefits of using data tables in R:

Speed: Data tables are significantly faster than data.frames for many data manipulation tasks.

Flexibility: Data tables offer a wider range of features than data.frames, making them more versatile for data analysis.

Ease of use: Data tables are easy to learn and use, even for beginners.
If you are working with large datasets in R, or if you need to perform complex data manipulation tasks, then data tables are a valuable tool.

1. Creating a data.table:

```
library(data.table)

# Creating a data.table from a data.frame
df <- data.frame(A = c(1, 2, 3), B = c("apple", "banana", "cherry"))
dt <- as.data.table(df)

# Creating a data.table directly
dt <- data.table(A = c(1, 2, 3), B = c("apple", "banana", "cherry"))
```

2. Selecting columns:

```
# Selecting columns by name
dt[, .(A, B)]

# Selecting columns by index
dt[, c(1, 2)]

# Selecting columns and computing new columns
```

```
dt[, .(A, B, C = A + 1)]

# Excluding columns
dt[, !"A"]
```

3. Filtering rows:

```
# Filtering rows based on a condition
dt[A > 1]

# Filtering rows using multiple conditions
dt[A > 1 & B == "banana"]

# Using %in% to match values
dt[B %in% c("apple", "banana")]

# Negating a condition
dt[!(A > 1)]
```

4. Sorting data:

```
# Sorting by a single column
dt[order(A)]

# Sorting by multiple columns
dt[order(A, B)]

# Sorting in descending order
dt[order(-A)]

# Sorting using the `setorder` function (faster for large data)
setorder(dt, A)
```

5. Aggregating data:

```
# Computing summary statistics by group
dt[, .(mean_A = mean(A), max_B = max(B)), by = B]

# Counting the number of rows by group
dt[, .N, by = B]

# Aggregating with multiple functions
```

```
dt[, .(mean_A = mean(A), max_A = max(A)), by = B]
```

6. Updating values:

```
# Updating values in a column
dt[A == 1, B := "orange"]

# Updating values using computed expressions
dt[, A := A + 1]

# Updating values in multiple columns
dt[A > 1, c("A", "B") := .(A * 2, paste0(B, "_updated"))]
```

7. Joining data.tables:

```
# Inner join
dt1 <- data.table(ID = c(1, 2, 3), Value = c("A", "B", "C"))
dt2 <- data.table(ID = c(2, 3, 4), Price = c(10, 20, 30))
result_inner <- dt1[dt2, on = "ID"]

# Left join
result_left <- dt1[dt2, on = "ID", allow.cartesian = TRUE]

# Right join
result_right <- dt1[dt2, on = "ID", allow.cartesian = TRUE][!is.na(Value)]

# Full join
result_full <- dt1[dt2, on = "ID", allow.cartesian = TRUE][!is.na(Value) |
!is.na(Price)]
```

These examples demonstrate various operations you can perform using the `data.table` package for data manipulation in R. Remember to install and load the `data.table` package before running the code.

Check Your Progress

1. Which of the following is NOT a key feature of data tables?
(A) Fast indexing
(B) Fast joins
(C) Fast aggregation
(D) Fast updating

2. Which of the following is the correct syntax for creating a data.table from a data.frame?
(A) data.table(df)
(B) as.data.table(df)
(C) df <\- data.table()
(D) df <\- as.data.frame()

10. Matrix in R

In R, a matrix is a two-dimensional data structure that contains elements of the same data type arranged in rows and columns. It can be thought of as a rectangular grid of values. Matrices are commonly used for mathematical operations and data analysis.

To create a matrix in R, you can use the matrix() function. The function takes several arguments, including the data elements, the number of rows, and the number of columns. Here's an example:

```
# Creating a matrix using the matrix() function
matrix1 <- matrix(data = 1:9, nrow = 3, ncol = 3)

# Printing the matrix
print(matrix1)
```

In this example, we created a 3x3 matrix with values ranging from 1 to 9. The matrix is stored in the variable matrix1, and we used the print() function to display its contents.

You can also create a matrix by combining vectors using the cbind() or rbind() functions. Here's an example:

```
# Creating a matrix using cbind() and rbind()
vec1 <- 1:3
vec2 <- 4:6
vec3 <- 7:9

# Combining vectors into a matrix using cbind()
matrix2 <- cbind(vec1, vec2, vec3)

# Combining vectors into a matrix using rbind()
matrix3 <- rbind(vec1, vec2, vec3)

# Printing the matrices
```

```
print(matrix2)
print(matrix3)
```

In this example, we created three vectors (vec1, vec2, and vec3) with values ranging from 1 to 9. We then combined these vectors using cbind() and rbind() to create two different matrices (matrix2 and matrix3).

You can access elements of a matrix using square brackets [row, column] notation. For example:

```
# Accessing elements of a matrix
print(matrix1[1, 2])  # Accessing element in the first row, second column
print(matrix2[3, 1])  # Accessing element in the third row, first column
```

In this example, we accessed specific elements of matrix1 and matrix2 using the row and column indices.

These are just some basic examples of working with matrices in R. Matrices in R are versatile and offer various functions and operations for manipulation, calculation, and analysis.

Check Your Progress

Question 16:
Which function is used to create a matrix in R?
A) data.frame()
B) array()
C) matrix()
D) list()

Question 17:
How can you access a specific element in a matrix with row index 3 and column index 2?
A) matrix[2, 3]
B) matrix[3, 2]
C) matrix[1, 3]
D) matrix[3, 1]

3. Activity Question:

Create a matrix named student_grades to store the grades of five students in three subjects: Math, Science, and English. The grades are as follows:

Math: 85, 90, 78, 92, 88
Science: 80, 75, 82, 88, 90
English: 92, 85, 78, 80, 88

Write the code in R to create the matrix student_grades and display its contents. Also, find the average grade of each student across all subjects.

Hint: You can use the matrix() function to create the matrix and then calculate the row-wise average using the rowMeans() function.

11. Clustering

Clustering is a fundamental concept in machine learning and data analysis that involves grouping similar data points together based on their inherent characteristics or patterns. The objective of clustering is to identify meaningful structures within a dataset, where similar data points are clustered together while dissimilar points are separated into different clusters. This process helps in understanding the underlying patterns, relationships, and distributions of the data.

Here's a detailed explanation of the concept of clustering:

1. Objective: The primary goal of clustering is to partition a given dataset into groups or clusters in such a way that data points within the same cluster are more similar to each other compared to those in other clusters. The similarity is typically measured based on certain distance metrics, where smaller distances imply greater similarity.

2. Unsupervised Learning: Clustering is an example of unsupervised learning since it does not rely on labeled data or predefined classes. Instead, it aims to discover patterns and structures within the dataset solely based on the inherent similarities among the data points.

3. Data Representation: Clustering can be performed on various types of data, including numerical, categorical, or mixed data. However, it is commonly applied to datasets represented as feature vectors, where each data point is described by a set of attributes or features. These features could represent any measurable characteristics of the data points.

4. Distance Metrics: The choice of distance metric plays a crucial role in clustering algorithms. Commonly used distance measures include Euclidean distance, Manhattan distance, cosine similarity, and correlation distance. The distance metric depends on the nature of the data and the specific problem at hand.

5. Clustering Algorithms: Numerous clustering algorithms have been developed to perform the task of clustering. Some popular algorithms include K-means, Hierarchical Agglomerative

Clustering (HAC), DBSCAN (Density-Based Spatial Clustering of Applications with Noise), and Gaussian Mixture Models (GMM). Each algorithm has its own assumptions, advantages, and limitations, making them suitable for different types of datasets and clustering objectives.

- K-means: It is an iterative algorithm that aims to partition the data into K clusters. It minimizes the sum of squared distances between data points and their cluster centroids. The number of clusters, K, needs to be predefined.

- HAC: This hierarchical algorithm creates a tree-like structure of clusters. It starts with each data point as a separate cluster and merges the closest pairs of clusters successively until a stopping criterion is met. The stopping criterion can be based on a fixed number of clusters or a threshold distance.

- DBSCAN: This density-based algorithm groups together data points that are closely packed and separates sparse regions. It identifies dense regions based on a specified neighborhood radius and minimum number of points within that radius.

- GMM: This probabilistic model assumes that the data points are generated from a mixture of Gaussian distributions. It estimates the parameters of these distributions and assigns each data point to one of the Gaussian components.

6. Evaluation: Once the clustering algorithm has been applied, it is important to evaluate the quality of the obtained clusters. Evaluation metrics vary depending on whether the ground truth labels are available or not. If ground truth is available, metrics like purity, precision, and recall can be used. In the absence of ground truth, metrics such as silhouette coefficient, Davies-Bouldin index, and Calinski-Harabasz index can be utilized.

7. Applications: Clustering has a wide range of applications in various fields. It is extensively used in customer segmentation, anomaly detection, image segmentation, document clustering, recommender systems, genetic analysis, and social network analysis, to name a few. Clustering helps in understanding the inherent structures and characteristics of the data, enabling better decision-making and insights.

In summary, clustering is a powerful technique for grouping similar data points together based on their intrinsic properties. It is an unsupervised learning approach that helps discover meaningful patterns and structures within datasets, making it a valuable tool in data analysis and machine learning.

Check Your Progress

18. Which of the following is NOT a type of clustering algorithm?
(a) K-means
(b) Hierarchical Agglomerative Clustering (HAC)
(c) DBSCAN
(d) Gaussian Mixture Models (GMM)

19. Which of the following is NOT a distance metric used in clustering algorithms?
(a) Euclidean distance
(b) Manhattan distance
(c) Cosine similarity

(d) Correlation distance

20. What is the objective of clustering?
(a) To partition a given dataset into groups or clusters in such a way that data points within the same cluster are more similar to each other compared to those in other clusters.
(b) To identify meaningful structures within a dataset, where similar data points are clustered together while dissimilar points are separated into different clusters.
(c) Both (a) and (b)

21. What are some applications of clustering?
(a) Customer segmentation
(b) Anomaly detection
(c) Image segmentation
(d) All of the above

12. Concept of Prediction Model

A prediction model, also known as a predictive model, is a mathematical or statistical framework that is used to forecast or estimate future outcomes based on available data. It is a key component of predictive analytics, a field that aims to use historical patterns and data analysis techniques to make informed predictions about future events or behaviors.

The process of developing a prediction model typically involves several steps:

1. Problem Definition: The first step is to clearly define the problem or question you want to address with the prediction model. This could be anything from predicting stock prices to forecasting sales for a particular product.

2. Data Collection: Once the problem is defined, relevant data needs to be collected. This data can come from various sources such as databases, surveys, or online repositories. The data should be representative of the problem domain and include both input variables (also called predictors or features) and the corresponding outcome variable (the target variable).

3. Data Preprocessing: Raw data often requires preprocessing to clean and transform it into a suitable format for analysis. This step involves tasks such as removing outliers, handling missing values, normalizing or standardizing variables, and splitting the data into training and testing sets.

4. Feature Selection/Engineering: In some cases, the available data may contain numerous variables, and not all of them may be relevant or useful for making predictions. Feature selection or engineering involves identifying the most informative features or creating new features from existing ones to enhance the model's predictive power.

5. Model Selection: Choosing an appropriate prediction model is crucial. The selection depends on the nature of the problem, the available data, and the desired outcome. Different types of models can be used, such as linear regression, decision trees, support vector machines, neural networks, or ensemble methods. Each model has its strengths and weaknesses, and the choice depends on factors like interpretability, accuracy, complexity, and scalability.

6. Model Training: After selecting a model, the next step is to train it using the training data. During training, the model learns the relationships between the input variables and the target

variable by optimizing its internal parameters. The optimization process varies depending on the model type and can involve techniques like gradient descent or maximum likelihood estimation.

7. Model Evaluation: Once the model is trained, it needs to be evaluated to assess its performance. This is typically done using the testing data, which the model has not seen during training. Various evaluation metrics are used depending on the problem type, such as accuracy, precision, recall, F1 score, or mean squared error. The goal is to ensure that the model generalizes well to new, unseen data and produces accurate predictions.

8. Model Refinement: If the model's performance is not satisfactory, it may require refinement. This can involve adjusting hyperparameters (parameters that control the model's behavior) or changing the model structure. Techniques like cross-validation or grid search can help identify optimal hyperparameter settings.

9. Prediction and Deployment: Once the model meets the desired performance criteria, it can be used to make predictions on new, unseen data. The model takes the input variables and generates predictions or estimates for the target variable. These predictions can be used for decision-making, forecasting, risk assessment, or other purposes depending on the problem domain.

It's important to note that prediction models are not infallible and are subject to limitations and assumptions. The accuracy and reliability of predictions depend on the quality of the data, the appropriateness of the model, and the assumptions made during the modeling process. Regular monitoring and updating of the model may be necessary to maintain its performance over time.

Examples of prediction models

1. Linear Regression: This is a common prediction model used to estimate a numerical target variable based on a set of input variables. For example, it can be used to predict the sales of a product based on factors like price, advertising budget, and competitors' prices.

2. Decision Trees: Decision trees are used to make predictions by creating a tree-like model of decisions and their possible consequences. They are widely used in fields such as finance, healthcare, and marketing. For example, a decision tree model can predict whether a customer will churn based on factors like their purchase history, demographic information, and customer service interactions.

3. Random Forest: Random Forest is an ensemble learning technique that combines multiple decision trees to make predictions. It is often used for classification and regression tasks. For example, it can be used to predict the likelihood of a loan default based on features such as credit score, income, and loan amount.

4. Support Vector Machines (SVM): SVM is a machine learning algorithm used for both classification and regression tasks. It finds a hyperplane that best separates data points into different classes or predicts a continuous target variable. SVMs have applications in fields like image recognition, text classification, and stock market prediction.

5. Recurrent Neural Networks (RNN): RNNs are a type of neural network designed to process sequential data, such as time series or natural language. They have memory cells that allow them to retain information from previous steps, making them suitable for tasks like speech recognition, language translation, and stock market forecasting.

6. Gradient Boosting: Gradient boosting is an ensemble learning technique that combines multiple weak prediction models (typically decision trees) to create a stronger, more accurate model. It iteratively trains models to correct the mistakes made by the previous models. Gradient boosting has applications in areas like fraud detection, customer churn prediction, and recommendation systems.

7. Time Series Models: Time series models are specifically designed to analyze and predict data that changes over time. Examples include autoregressive integrated moving average (ARIMA) models, exponential smoothing models, and recurrent neural networks. Time series models are widely used for forecasting stock prices, weather patterns, sales trends, and other time-dependent phenomena.

These are just a few examples of prediction models, and there are many more depending on the specific problem and the nature of the data. The choice of model depends on factors like the type of data, the problem complexity, interpretability requirements, and available computational resources.

14. Check Your Progress Answers

1. c) Character
2. a) name <- "John"
3. a) 4.9
4. a) c(1, 2, 3, 4, 5, 6)
5. To create a list in R, you can use the `list()` function.
6. To access an element in a list by name, you can use the `$` operator.
7. `print(df$City)`
8. `df$Country <- c("USA", "France", "UK", "Australia")`
9. B) An array that can store multiple elements of the same data type in a single row or column.
10. D) Any number of dimensions
11. d) mean()
12. b) lm()
13. c) Bar plot
14. The answer is (D). Data tables are fast for indexing, joining, and aggregation, but they are not necessarily faster than data.frames for updating
15. The answer is (B). The as.data.table() function is used to convert a data.frame to a data.table.
16. C) matrix()
17. B) matrix[3, 2]
18. (c) DBSCAN is not a type of clustering algorithm. It is a density-based algorithm that groups together data points that are closely packed and separates sparse regions.

19.(c) Cosine similarity is not a distance metric used in clustering algorithms.

20.(c) Both (a) and (b) are the objective of clustering.

21.(d) All of the above are applications of clustering. Customer segmentation, anomaly detection, and image segmentation are all examples of how clustering can be used in real-world applications.

15. Activity Question Answers

1.

```
# Given numeric vector
numeric_vector <- c(1, 2, 3, 4, 5)

# 1. Modify the third element of the vector to 6
numeric_vector[3] <- 6

# 2. Calculate the sum of all the elements in the vector
vector_sum <- sum(numeric_vector)

# 3. Find the maximum value in the vector
vector_max <- max(numeric_vector)

# 4. Select the elements at index 2 and 4 and create a subset vector
subset_vector <- numeric_vector[c(2, 4)]

# 5. Concatenate the vector with itself
concatenated_vector <- c(numeric_vector, numeric_vector)

# Print the final outputs
print(numeric_vector)
print(vector_sum)
print(vector_max)
print(subset_vector)
print(concatenated_vector)
```

2.

Activity: Create a Data Frame

Instructions:
Based on the given context about data frames in R, perform the following activities:

R Programming

1. Create a data frame named `students` with the following columns:
   - Column 1: "Name" - containing the names of five students: "Tom", "Emma", "Liam", "Olivia", "Noah".
   - Column 2: "Age" - containing the ages of the students: 18, 20, 19, 21, 22.
   - Column 3: "City" - containing the cities where the students reside: "New York", "London", "Paris", "Sydney", "Tokyo".
   Print the resulting data frame.

Answer:

```
students <- data.frame(
  Name = c("Tom", "Emma", "Liam", "Olivia", "Noah"),
  Age = c(18, 20, 19, 21, 22),
  City = c("New York", "London", "Paris", "Sydney", "Tokyo")
)
print(students)
```

2. Access and print the "Age" column of the `students` data frame.

Answer:

```
print(students$Age)
```

3. Remove the "City" column from the `students` data frame and print the updated data frame.

Answer:

```
students <- subset(students, select = -City)
print(students)
```

4. Filter the `students` data frame to only include the students whose age is greater than or equal to 20. Print the filtered data frame.

Answer:

```
filtered_students <- subset(students, Age >= 20)
print(filtered_students)
```

5. Calculate the average age of the students in the `students` data frame and print the result.

Answer:

```
mean_age <- mean(students$Age)
print(mean_age)
```

3.

Answer:

```
# Creating the student_grades matrix
student_grades <- matrix(c(85, 90, 78, 92, 88,
                           80, 75, 82, 88, 90,
                           92, 85, 78, 80, 88),
                         nrow = 5, ncol = 3, byrow = TRUE,
                         dimnames = list(NULL, c("Math", "Science",
"English")))

# Displaying the matrix
print(student_grades)
```

To find the average grade of each student across all subjects, you can use the rowMeans() function:

```
# Calculating the average grade of each student
average_grades <- rowMeans(student_grades)
print(average_grades)
```

The average_grades vector contains the average grade for each student across all subjects.

16. Short Answer Questions

1. How can you create a data.table in R? Provide examples using both a data.frame and direct creation methods.

Answer:

```
#Creating a data.table from a data.frame:

library(data.table)

df <- data.frame(A = c(1, 2, 3), B = c("apple", "banana", "cherry"))
dt <- as.data.table(df)

Creating a data.table directly:

library(data.table)

dt <- data.table(A = c(1, 2, 3), B = c("apple", "banana", "cherry"))
```

2. How can you select specific columns from a data.table? Show examples using column names, indices, and creating new computed columns.
Answer:

```
#Selecting columns by name:
dt[, .(A, B)]

#Selecting columns by index:
dt[, c(1, 2)]

#Selecting columns and computing new columns:
dt[, .(A, B, C = A + 1)]

#Excluding columns:
dt[, !"A"]
```

3. Clustering is a powerful technique for grouping similar data points together based on their intrinsic properties. However, it is not without its challenges. What are some of the challenges of using clustering in R?

4. How do prediction models help businesses in making informed decisions?

5.In your opinion, what are the key challenges and limitations of prediction models in accurately forecasting future outcomes?

17. Long Answer Questions

R Programming

1. You are given a matrix named sales_data that represents the monthly sales of three products (A, B, and C) in four different regions (North, South, East, and West). The matrix sales_data has the following structure:

```
sales_data <- matrix(c(120, 80, 150, 110,
                       90, 100, 120, 80,
                       140, 70, 100, 130,
                       100, 110, 90, 120,
                       80, 130, 110, 90,
                       120, 90, 80, 140),
                     nrow = 6, ncol = 4, byrow = TRUE,
                     dimnames = list(c("Jan", "Feb", "Mar", "Apr", "May",
"Jun"),
                                     c("North", "South", "East", "West")))
```

a) Calculate the total sales for each product across all regions. Which product had the highest total sales?

b) Calculate the average sales for each region across all products. Which region had the highest average sales?

c) Calculate the monthly sales for each product. Which month had the highest sales for Product C?

Hint: You can use functions such as colSums(), rowMeans(), and max() to perform the calculations
.

Answer

a) To calculate the total sales for each product across all regions, we can use the colSums() function and then determine which product had the highest total sales using the max() function.

```
# Calculating total sales for each product
product_totals <- colSums(sales_data)
print(product_totals)

# Finding the product with the highest total sales
highest_sales_product <- names(product_totals)[which.max(product_totals)]
print(highest_sales_product)
```

The product_totals vector contains the total sales for each product across all regions. From the output, we can see that Product B had the highest total sales.

b) To calculate the average sales for each region across all products, we can use the rowMeans() function and then determine which region had the highest average sales using the max() function.

```
# Calculating average sales for each region
region_averages <- rowMeans(sales_data)
print(region_averages)

# Finding the region with the highest average sales
highest_average_region <- names(region_averages)[which.max(region_averages)]
print(highest_average_region)
```

The region_averages vector contains the average sales for each region across all products. From the output, we can see that the North region had the highest average sales.

c) To calculate the monthly sales for each product, we can sum the sales for each month using the rowSums() function and then determine which month had the highest sales for Product C using the max() function.

```
# Calculating monthly sales for each product
monthly_sales <- rowSums(sales_data)
print(monthly_sales)

# Finding the month with the highest sales for Product C
highest_sales_month <- names(monthly_sales[which.max(sales_data[, "East"])])
print(highest_sales_month)
```

The monthly_sales vector contains the total sales for each month across all products. From the output, we can see that January had the highest sales for Product C.

2.Explain the process of clustering in R, including the steps involved, the choice of clustering algorithm, and the evaluation of clustering results. Provide examples of real-world applications where clustering in R can be beneficial.

3.You have been given a dataset that contains information about the sales of different products in various regions. The dataset is stored in a two-dimensional array called "sales_data" with the following structure:

- Each row represents a region, and each column represents a product.
- The first row contains the names of the products.
- The first column contains the names of the regions.

You need to perform several tasks using the given dataset. Complete the following steps:

Step 1:
Create an array named "sales_data" in R that represents the following sales data. Each value represents the number of units sold.

```
     [,1] [,2] [,3] [,4]
[1,] "Region"  "A"   "B"   "C"
[2,] "Product1" 10    5    12
[3,] "Product2"  8    7    15
[4,] "Product3"  3   10     6
```

Step 2:
Calculate the total sales for each region and store them in a vector named "total_sales_region". The vector should contain the sum of units sold in each region.

Step 3:
Find the region with the highest total sales and store its name in a variable named "region_highest_sales".

Step 4:
Calculate the average sales for each product and store them in a vector named "average_sales_product". The vector should contain the average number of units sold for each product.

Step 5:
Find the product with the highest average sales and store its name in a variable named "product_highest_average_sales".

Step 6:
Print the "total_sales_region" vector, "region_highest_sales", "average_sales_product" vector, and "product_highest_average_sales" to the console.

Answer

```r
# Step 1: Create the sales_data array
sales_data <- array(c("Region", "A", "B", "C",
                      "Product1", 10, 5, 12,
```

```
                      "Product2", 8, 7, 15,
                      "Product3", 3, 10, 6),
                    dim = c(4, 4))

# Step 2: Calculate total sales for each region
total_sales_region <- colSums(sales_data[-1, -1])

# Step 3: Find the region with the highest total sales
region_highest_sales <-
names(total_sales_region)[which.max(total_sales_region)]

# Step 4: Calculate average sales for each product
average_sales_product <- colMeans(sales_data[-1, -1])

# Step 5: Find the product with the highest average sales
product_highest_average_sales <-
names(average_sales_product)[which.max(average_sales_product)]

# Step 6: Print the results
cat("Total Sales by Region:\n")
print(total_sales_region)

cat("\nRegion with Highest Sales: ", region_highest_sales)

cat("\n\nAverage Sales by Product:\n")
print(average_sales_product)

cat("\nProduct with Highest Average Sales: ", product_highest_average_sales)
```

4.In your view, discuss the ethical considerations and potential risks associated with the use of prediction models in decision-making processes, particularly in sensitive domains such as criminal justice or healthcare. Provide examples to support your argument.

5. You have been given a dataset named "sales_data.csv" containing sales information for a company's products. The dataset has the following columns: "ProductID", "ProductName", "Category", "Price", "Quantity", and "SalesDate". Using R programming, perform the following data manipulation tasks:

ProductID,ProductName,Category,Price,Quantity,SalesDate
1,Product A,Electronics,100,50,2022-01-01
2,Product B,Electronics,150,80,2022-01-05
3,Product C,Home Appliances,200,120,2022-01-10
4,Product D,Clothing,50,200,2022-01-15
5,Product E,Home Appliances,300,70,2022-02-01
6,Product F,Clothing,80,150,2022-02-05
7,Product G,Electronics,200,110,2022-02-10

8,Product H,Clothing,40,250,2022-03-01
9,Product I,Home Appliances,250,90,2022-03-05
10,Product J,Electronics,120,180,2022-03-10
11,Product K,Clothing,60,300,2022-03-15
12,Product L,Home Appliances,180,120,2022-04-01
13,Product M,Electronics,90,200,2022-04-05
14,Product N,Home Appliances,220,80,2022-04-10
15,Product O,Clothing,70,150,2022-04-15

1. Load the "sales_data.csv" dataset into R and assign it to a variable.
2. Display the first 10 rows of the dataset.
3. Calculate the total sales revenue generated by each product. Create a new column named "TotalRevenue" that stores the calculated values.
4. Filter the dataset to include only the products with a sales quantity greater than or equal to 100.
5. Group the filtered dataset by the "Category" column and calculate the average price and total sales quantity for each category.
6. Sort the dataset based on the "TotalRevenue" column in descending order.
7. Create a new dataset that includes only the top 5 products with the highest total revenue.
8. Calculate the total sales revenue for each month and create a summary table that shows the total revenue for each month in a descending order of revenue.

19. MCQ questions and answers

20 MCQ questions

1. Which command is used to install packages in R?
a) install.packages()
b) load.package()
c) library()
d) require()

2. Which data type is used to store a sequence of values of the same type in R?
a) Vector
b) List
c) Frame
d) Array

3. In R, which data structure is a collection of vectors of the same length?
a) Vector
b) List
c) Frame
d) Matrix

4. How do you access the fourth element of a vector named "data" in R?
a) data[4]

b) data(4)
c) data[[4]]
d) data$4

5. Which R function is used to perform basic statistical calculations like mean, median, and mode?
a) summary()
b) stats()
c) statistics()
d) base()

6. In R, which command is used to create a scatter plot?
a) plot()
b) scatter()
c) scatterplot()
d) scatter_plot()

7. Which package in R is commonly used for data manipulation tasks using data tables?
a) dplyr
b) tidyr
c) data.table
d) plyr

8. In R, what is the command to create a new column named "total" in a data frame called "df"?
a) df$total <- c()
b) df$new <- c()
c) df$total <- list()
d) df$new <- list()

9. Which clustering algorithm is commonly used for partitioning data into groups in R?
a) K-means
b) Hierarchical
c) DBSCAN
d) Spectral

10. What is the concept of a prediction model in R?
a) It is a statistical model used to predict future outcomes based on historical data.
b) It is a visualization technique used to understand patterns in data.
c) It is a method for dimensionality reduction in high-dimensional datasets.
d) It is a technique for outlier detection in data.

11. Which package in R is commonly used for building and evaluating prediction models?
a) caret
b) ggplot2
c) randomForest
d) gbm

12. What is the first step in analyzing a real-world problem using R?
a) Collecting and cleaning the data
b) Exploratory data analysis

c) Building a prediction model
d) Evaluating the model's performance

13. Which R function is used to calculate the correlation coefficient between two variables?
a) cor()
b) cor.test()
c) correlation()
d) pearson()

14. How do you convert a character vector named "numbers" into numeric values in R?
a) as.character(numbers)
b) as.numeric(numbers)
c) convert(numbers, numeric)
d) num(numbers)

15. In R, how do you remove missing values from a data frame named "df"?
a) df[complete.cases(df),]
b) df[is.na(df),]
c) df[!is.na(df),]
d) df[na.rm(df),]

16. Which R function is used to calculate the mean of a numeric vector excluding missing values?
a) mean()
b) sum()
c) na.rm()
d) average()

17. What is the output of the following R code: "length(c(1, 2, 3))"?
a) 1
b) 2
c) 3
d) 4

18. In R, what does the "dim" function return for a matrix?
a) The number of rows
b) The number of columns
c) The dimensions of the matrix
d) The sum of all elements in the matrix

19. Which R function is used to perform a t-test between two groups?
a) t.test()
b) test()
c) ttest()
d) compare()

20. How do you access the second element of a list named "my_list" in R?
a) my_list(2)
b) my_list[[2]]
c) my_list[2]
d) my_list$2

Answer:

1. a) install.packages()
2. a) Vector
3. d) Matrix
4. a) data[4]
5. a) summary()
6. a) plot()
7. c) data.table
8. a) df$total <- c()
9. a) K-means
10. a) It is a statistical model used to predict future outcomes based on historical data.
11. a) caret
12. a) Collecting and cleaning the data
13. a) cor()
14. b) as.numeric(numbers)
15. a) df[complete.cases(df),]
16. a) mean()
17. c) 3
18. c) The dimensions of the matrix
19. a) t.test()
20. b) my_list[[2]]