R Programming: A Hands-On Guide

Introduction to R

R is a powerful open-source statistical programming language and environment. It's widely used in various fields, including data analysis, statistics, machine learning, and data visualization.

Basic R Concepts

1. Variables and Data Types:

- **Variables:** Containers for storing data.
- **Data Types:**
    - Numeric (integer, double)
    - Character (string)
    - Logical (TRUE, FALSE)
    - Factor
    - Complex

Code snippet

```
# Assigning values to variables
x <- 10
y <- "Hello, World!"
z <- TRUE

# Printing the values
print(x)
print(y)
print(z)
```

2. Operators:

- Arithmetic (+, -, *, /, ^)
- Relational (<, >, <=, >=, ==, !=)
- Logical (&&, ||, !)

Code snippet

```
# Example of arithmetic operations
result <- 5 + 3 * 2
print(result)

# Example of logical operations
is_greater <- !10 > 5
print(is_greater)
```

3. Data Structures:

- **Vectors:** Ordered collection of elements of the same data type.
- **Matrices:** Two-dimensional array of elements of the same data type.
- **Data Frames:** Rectangular data structure with columns of different data types.
- **Lists:** Ordered collection of elements of different data types.

Code snippet

```
# Creating a vector
numbers <- c(1, 2, 3, 4, 5)

# Creating a matrix
my_matrix <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)

# Creating a data frame
my_data <- data.frame(
  name = c("Alice", "Bob", "Charlie"),
  age = c(25, 30, 35)
)

# Creating a list
my_list <- list(x = 10, y = "Hello", z = TRUE)
```

R Packages

R's strength lies in its vast ecosystem of packages. Some essential packages for data analysis include:

- **dplyr:** For data manipulation and transformation
- **ggplot2:** For creating elegant visualizations
- **tidyr:** For data tidying
- **caret:** For machine learning models
- **stats:** For statistical functions
- Code snippet

```
# Installing a package
install.packages("dplyr")

# Loading a package
library(dplyr)
```

Data Manipulation and Analysis

1. Filtering and Subsetting:

Code snippet

```
# Filtering rows
filtered_data <- my_data %>% filter(age > 30)

# Subsetting columns
selected_cols <- my_data[, c("name", "age")]
```

2. Summarizing Data:

Code snippet

```
# Calculating summary statistics
summary(my_data)

# Grouping and aggregating data
grouped_data <- my_data %>% group_by(age) %>% summarize(mean_age =
mean(age))
```

Data Visualization

1. Basic Plots:

Code snippet

```
# Creating a scatter plot
plot(x, y)

# Creating a histogram
hist(numbers)
```

2. ggplot2:

Code snippet

```
# Creating a scatter plot using ggplot2
library(ggplot2)
ggplot(my_data, aes(x = age, y = name)) + geom_point()
```

Further Exploration

- **Functions:** Creating your own functions for code reusability.
- **Control Flow:** Using if-else statements, loops, and conditional expressions.
- **Data Import and Export:** Reading and writing data from various formats.
- **Machine Learning:** Building predictive models using R's machine learning libraries.
- **Advanced Visualization:** Creating interactive plots and dashboards.

—-------------------------------

Here’s a hands-on guide to teaching R programming, covering essential topics with both theory and practical examples along with their outputs. This material is structured to help students understand key concepts while providing ample coding practice.

1. Introduction to R

Theory:

- R is a language used primarily for statistical computing and data analysis.

- It is widely used in fields like data science, statistics, and bioinformatics.

Basic Syntax

- R expressions consist of variables, operators, and functions.

- Commands are executed line by line in the R console.

Code Example:

```r
# Basic arithmetic
a <- 5
b <- 10
sum <- a + b
sum
```

Output:

```
[1] 15
```

---

2. Variables and Data Types

Theory:

- R supports various data types: numeric, integer, character, logical, etc.
- Variables store data and can be reused later.

Types of Data in R

Code Example:

```r
# Assigning values to different data types
num_var <- 12.5       # Numeric
int_var <- 5L         # Integer
char_var <- "R Programming"  # Character
bool_var <- TRUE      # Logical

# Printing data types
class(num_var)
class(int_var)
class(char_var)
class(bool_var)
```

Output:

```
[1] "numeric"
[1] "integer"
[1] "character"
[1] "logical"
```

```

---

## 3. Vectors and Basic Operations

Theory:

- Vectors are the most basic data structures in R.

- They are used to store sequences of data of the same type.

### Creating and Manipulating Vectors

Code Example:

```r
# Creating a vector
v <- c(1, 2, 3, 4, 5)

# Accessing elements
v[1]   # First element
v[2:4] # Elements 2 to 4

# Basic operations
v * 2  # Multiply each element by 2
sum(v) # Sum of all elements

```

Output:

```
[1] 1

[1] 2 3 4

[1] 2 4 6 8 10

[1] 15
```

---

4. Matrices

Theory:

- A matrix is a two-dimensional data structure in R, where all elements are of the same type.

Creating and Manipulating Matrices

Code Example:

```r
# Creating a matrix

matrix_data <- matrix(1:9, nrow=3, byrow=TRUE)

```
# Accessing matrix elements
matrix_data[1, ]   # First row
matrix_data[, 2]   # Second column

# Transpose of a matrix
t(matrix_data)
```

Output:

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9

[1] 1 2 3
[1] 2 5 8

     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

---

5. Lists

Theory:

- Lists are collections of objects, which can contain different data types like numbers, strings, vectors, or even other lists.

Creating and Accessing Lists

Code Example:

```r
# Creating a list
my_list <- list(name = "Alice", age = 25, scores = c(90, 85, 88))

# Accessing list elements
my_list$name
my_list$scores

# Adding new elements to the list
my_list$city <- "New York"
my_list
```

Output:

```
[1] "Alice"
[1] 90 85 88
$name
[1] "Alice"

$age
[1] 25

$scores
[1] 90 85 88

$city
[1] "New York"
```

---

6. Data Frames

Theory:

- Data frames are table-like structures where each column can contain different types of data.

- They are widely used for datasets in R.

Creating and Manipulating Data Frames

Code Example:

```r
# Creating a data frame
df <- data.frame(
  Name = c("Alice", "Bob", "Charlie"),
  Age = c(25, 30, 22),
  Score = c(90, 80, 85)
)

# Accessing data frame columns
df$Name
df[2, ]    # Second row

# Summary of the data frame
summary(df)
```

Output:

```
[1] "Alice" "Bob"   "Charlie"
```

     Name   Age   Score
 Alice   :1   Min.   :22.00   Min.   :80.00
 Bob   :1   1st Qu.:23.50   1st Qu.:82.50
 Charlie :1   Median :25.00   Median :85.00
              Mean   :25.67   Mean   :85.00
              3rd Qu.:27.50   3rd Qu.:87.50
              Max.   :30.00   Max.   :90.00
```

---

## 7. Control Structures (if-else, for, while)

Theory:

- Control structures allow decision-making and iteration in R programs.

### Using `if-else`

Code Example:

```r
# if-else example
x <- 10
if (x > 5) {
  result <- "x is greater than 5"

} else {

  result <- "x is less than or equal to 5"

}

result

```

Output:

```

[1] "x is greater than 5"

```

Using `for` Loop

Code Example:

```r

# for loop example

for (i in 1:5) {

  print(i)

}

```

Output:
```

```
[1] 1

[1] 2

[1] 3

[1] 4

[1] 5
```

---

8. Functions in R

Theory:

- Functions are blocks of reusable code that perform specific tasks.

- You can define custom functions using the `function` keyword.

Creating a Function

Code Example:

```r
# Define a function to calculate the square of a number

square_function <- function(x) {

  return(x^2)

}
```

Using the function

square_function(4)

```

Output:

```

[1] 16

```

---

## 9. Reading and Writing Data

Theory:

- R can read and write various data formats, including CSV, Excel, and databases.

### Reading and Writing CSV Files

Code Example:

```r

# Reading a CSV file

data <- read.csv("data.csv")

# Writing a data frame to a CSV file

write.csv(df, "output.csv")

```

Here’s an extended guide focusing on Data Visualization, Statistical Functions, and Working with Real Datasets in R. These topics will help your students perform data analysis and communicate their findings effectively.

---

1. Data Visualization

Theory:

- Data visualization is essential to analyze trends, patterns, and relationships in data.

- R has popular libraries like ggplot2 and base R functions for visualizations.

Base R Plots

Code Example:

```r

# Simple scatter plot

x <- c(1, 2, 3, 4, 5)

y <- c(2, 4, 3, 5, 6)

plot(x, y, main="Scatter Plot", xlab="X-axis", ylab="Y-axis")

```

Output:

A basic scatter plot showing the relationship between x and y.

---

ggplot2 - Advanced Visualization

Theory:

- `ggplot2` is a powerful and flexible library for creating advanced plots.
- Plots are built using layers, with the `ggplot()` function as the base layer.

Code Example:

```r
# Installing and loading ggplot2 (if not already installed)
# install.packages("ggplot2")
library(ggplot2)

# Creating a data frame for plotting
data <- data.frame(
  x = c(1, 2, 3, 4, 5),
  y = c(2, 4, 3, 5, 6)

```
)

# Creating a scatter plot using ggplot2
ggplot(data, aes(x=x, y=y)) +
  geom_point(color="blue", size=3) +
  ggtitle("Scatter Plot with ggplot2") +
  xlab("X-axis") + ylab("Y-axis")
```

Output:

A more polished scatter plot with customizable elements (colors, size, labels).

Bar Plot

Code Example:

```r
# Bar plot with base R
counts <- table(mtcars$cyl)  # Frequency of cylinder values
barplot(counts, main="Bar Plot of Cylinders", xlab="Number of Cylinders", ylab="Frequency", col="lightblue")
```

Output:

A bar plot showing the distribution of car cylinders from the `mtcars` dataset.

---

2. Statistical Functions in R

Theory:

- R offers numerous built-in functions for statistical analysis, ranging from descriptive statistics to hypothesis testing and regression.

Descriptive Statistics

Code Example:

```r
# Basic statistical functions
data <- c(2, 4, 6, 8, 10)
mean(data)       # Mean
median(data)     # Median
sd(data)         # Standard deviation
var(data)        # Variance
range(data)      # Range
```

Output:

```
[1] 6   # Mean
[1] 6   # Median
[1] 3.162278   # Standard deviation
[1] 10   # Variance
[1] 2 10   # Range
```

Correlation and Covariance

Theory:

- Correlation measures the strength of the relationship between two variables.
- Covariance is a measure of how two variables change together.

Code Example:

```r
# Correlation between two variables
x <- c(1, 2, 3, 4, 5)
y <- c(2, 4, 3, 5, 6)
cor(x, y)  # Pearson correlation
```

Covariance between two variables

cov(x, y)

```

Output:

```
[1] 0.9   # Correlation
[1] 1.9   # Covariance
```

---

### Hypothesis Testing - t-test

Theory:

- t-tests are used to compare the means of two groups.
- The `t.test()` function in R performs one-sample, two-sample, or paired t-tests.

Code Example:

```r
# Two-sample t-test
group1 <- c(2, 3, 5, 8, 10)

group2 <- c(3, 5, 7, 9, 12)

t.test(group1, group2)

```

Output:

```

Two Sample t-test result with p-value and confidence intervals.

```

Linear Regression

Theory:

- Linear regression models the relationship between a dependent variable and one or more independent variables.

Code Example:

```r

# Linear regression on mtcars dataset

model <- lm(mpg ~ wt, data = mtcars)

# Summary of the regression model

summary(model)

# Plot the regression line

plot(mtcars$wt, mtcars$mpg)

abline(model, col="red")

```

Output:

- Summary includes coefficients, R-squared value, and p-values.

- The plot shows the regression line fitting the data.

---

3. Working with Real Datasets

Theory:

- R supports importing and manipulating various data formats like CSV, Excel, and SQL databases.

- Data wrangling is done using packages like dplyr and tidyr.

Reading CSV Files

Code Example:

```r

# Reading a CSV file

data <- read.csv("your_dataset.csv")

# Viewing the first few rows

head(data)

```

Data Wrangling with dplyr

Theory:

- `dplyr` simplifies data manipulation with functions like `filter()`, `select()`, `mutate()`, `arrange()`, and `summarise()`.

Code Example:

```r

# Installing and loading dplyr

# install.packages("dplyr")

library(dplyr)

# Using dplyr functions on mtcars dataset

filtered

Certainly! Let's expand your R programming teaching materials to include Data Visualization, Statistical Functions, and Working with Real Datasets. Each section will include theoretical explanations, practical code examples, and their expected outputs to provide a comprehensive learning experience for your students.

---

## 10. Data Visualization

Theory:

- Data visualization is the graphical representation of information and data.

- It helps in understanding trends, patterns, and outliers in data.

- ggplot2 is a popular R package for creating advanced and customizable visualizations based on the Grammar of Graphics.

### Installing and Loading ggplot2

Code Example:

```r
# Install ggplot2 if not already installed
install.packages("ggplot2")

# Load the ggplot2 library
library(ggplot2)
```

Output:

```
# Installation messages (if running for the first time)
```

```

### Basic Plot with ggplot2

Code Example:

```r
# Using the built-in mtcars dataset
data(mtcars)

# Create a scatter plot of mpg vs. hp
ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  labs(title = "Scatter Plot of MPG vs. Horsepower",
       x = "Horsepower (hp)",
       y = "Miles Per Gallon (mpg)")
```

Output:

A scatter plot displaying the relationship between Horsepower and Miles Per Gallon.

Creating a Histogram

Code Example:
```

```r
# Histogram of the 'mpg' (miles per gallon) variable
ggplot(mtcars, aes(x = mpg)) +
  geom_histogram(binwidth = 2, fill = "blue", color = "black") +
  labs(title = "Histogram of Miles Per Gallon",
       x = "Miles Per Gallon (mpg)",
       y = "Frequency")
```

Output:

A histogram showing the distribution of the mpg values in the mtcars dataset.

Boxplot Example

Code Example:

```r
# Boxplot of mpg grouped by the number of cylinders
ggplot(mtcars, aes(x = factor(cyl), y = mpg)) +
  geom_boxplot(fill = "lightgreen") +
  labs(title = "Boxplot of MPG by Number of Cylinders",
       x = "Number of Cylinders",
       y = "Miles Per Gallon (mpg)")
```

Output:

A boxplot comparing mpg across different cylinder counts in the mtcars dataset.

Bar Chart Example

Code Example:

```r
# Bar chart of the number of cars with each number of gears
ggplot(mtcars, aes(x = factor(gear))) +
  geom_bar(fill = "orange") +
  labs(title = "Number of Cars by Gears",
       x = "Number of Gears",
       y = "Count of Cars")
```

Output:

A bar chart showing the count of cars for each gear category in the mtcars dataset.

---

11. Statistical Functions

Theory:

- R provides a wide range of statistical functions to perform descriptive and inferential statistics.

- These functions help in summarizing data, testing hypotheses, and making predictions.

Descriptive Statistics

Code Example:

```r
# Summary statistics for the mtcars dataset
summary(mtcars)

# Calculate mean, median, and standard deviation for 'mpg'
mean_mpg <- mean(mtcars$mpg)
median_mpg <- median(mtcars$mpg)
sd_mpg <- sd(mtcars$mpg)

# Print the results
mean_mpg
median_mpg
sd_mpg
```

Output:

```
   mpg             cyl             disp             hp
 Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0
 1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5
 Median :19.20   Median :6.000   Median :196.3   Median :123.0
 Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7
 3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0
 Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0

[1] 20.09062
[1] 19.2
[1] 6.026948
```

Correlation Analysis

Code Example:

```r
# Calculate correlation between 'mpg' and 'hp'
correlation <- cor(mtcars$mpg, mtcars$hp)
correlation
```

Output:

```
[1] -0.7761684
```

Interpretation:

- There is a strong negative correlation between miles per gallon (mpg) and horsepower (hp). As horsepower increases, mpg tends to decrease.

Linear Regression

Code Example:

```r
# Perform linear regression with mpg as the response and hp as the predictor
lm_model <- lm(mpg ~ hp, data = mtcars)

# Summary of the regression model
summary(lm_model)
```

Output:

```
Call:
lm(formula = mpg ~ hp, data = mtcars)
```

```
Residuals:
    Min      1Q  Median      3Q     Max
-4.5432 -2.3645 -0.1257  1.4102  6.8727

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 30.09886    1.63392   18.40  < 2e-16 *
hp          -0.06823    0.01012   -6.75 1.79e-07 *
---
Signif. codes:
0 ‘*’ 0.001 ‘’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.863 on 30 degrees of freedom
Multiple R-squared:  0.6052,      Adjusted R-squared:  0.589
F-statistic: 45.87 on 1 and 30 DF,  p-value: 1.789e-07
```

Interpretation:

- The model indicates that for each additional horsepower, the miles per gallon decrease by approximately 0.068, holding other factors constant.

- The R-squared value of 0.6052 suggests that about 60.52% of the variability in mpg is explained by horsepower.

t-Test Example

Code Example:

```r
# Perform a t-test to compare mpg between automatic and manual transmission
# In mtcars, 'am' = 0 (automatic), 1 (manual)
automatic <- mtcars$mpg[mtcars$am == 0]
manual <- mtcars$mpg[mtcars$am == 1]

t_test_result <- t.test(manual, automatic)
t_test_result
```

Output:

```
	Welch Two Sample t-test


data:  manual and automatic
t = 3.7671, df = 18.286, p-value = 0.001506
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 3.047441 9.257986
sample estimates:
mean of x mean of y
  24.392   17.147
```

Interpretation:

- There is a statistically significant difference in mpg between manual and automatic transmissions (p-value = 0.001506 < 0.05).

- On average, manual transmission cars have higher mpg than automatic ones.

---

12. Working with Real Datasets

Theory:

- Working with real datasets involves importing data, cleaning and preprocessing, exploratory data analysis (EDA), and applying statistical or machine learning models.

- R provides various functions and packages to handle different data formats and complexities.

Using Built-in Datasets

Code Example:

```r
# Explore the built-in 'iris' dataset

data(iris)

head(iris)

summary(iris)
```

Output:

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa

  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
 Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

Importing External Data (CSV File)

Theory:

- Real-world data often comes from external sources like CSV files, Excel spreadsheets, databases, or APIs.

- R provides functions like `read.csv()` and packages like `readr` and `readxl` for importing data.

Code Example:

```r
# Assuming you have a CSV file named 'students_scores.csv' in your working directory

# Read the CSV file
students_data <- read.csv("students_scores.csv", header = TRUE, stringsAsFactors = FALSE)

# View the first few rows
head(students_data)

# Summary statistics
summary(students_data)
```

Output:

```
# Example output assuming 'students_scores.csv' has columns: Name, Age, Math_Score, English_Score

     Name               Age         Math_Score    English_Score
 Length:30          Min.   :12.00   Min.   : 50.0   Min.   : 45.0
```

```
 Class :character   1st Qu.:14.75   1st Qu.: 65.0   1st Qu.: 60.0

 Mode  :character   Median :16.00   Median : 75.0   Median : 70.0

                    Mean   :16.23   Mean   : 73.5   Mean   : 68.2

                    3rd Qu.:17.00   3rd Qu.: 85.0   3rd Qu.: 80.0

                    Max.   :18.00   Max.   : 95.0   Max.   : 90.0
```

Note:

- Ensure the CSV file (`students_scores.csv`) exists in your working directory. You can set the working directory using `setwd("path/to/your/directory")`.

Data Cleaning and Preprocessing

Theory:

- Real datasets often require cleaning, such as handling missing values, removing duplicates, and correcting data types.

- Proper preprocessing ensures accurate analysis and modeling.

Code Example:

```r
# Check for missing values

sum(is.na(students_data))


# View rows with missing values

students_data[!complete.cases(students_data), ]
```

```
# Remove rows with missing values
cleaned_data <- na.omit(students_data)

# Convert 'Age' to integer if it's not
cleaned_data$Age <- as.integer(cleaned_data$Age)

# Remove duplicate entries based on 'Name'
cleaned_data <- cleaned_data[!duplicated(cleaned_data$Name), ]

# View the cleaned data
head(cleaned_data)
```

Output:

```
# Example output assuming some missing values and duplicates were present

# Number of missing values
[1] 2

# Rows with missing values
# (Displays rows with NA in any column)

# Cleaned data after removing missing values and duplicates
```

```
      Name Age Math_Score English_Score
1    Alice  16         85            78
2      Bob  17         90            88
3  Charlie  15         70            65
4    Diana  18         95            92
5     Evan  14         60            58
6    Fiona  16         75            70
```

Exploratory Data Analysis (EDA)

Theory:

- EDA involves summarizing the main characteristics of the dataset, often using visual methods.

- It helps in understanding the data distribution, relationships between variables, and identifying any anomalies.

Code Example:

```r
# Summary statistics
summary(cleaned_data)

# Plotting Math_Score vs. English_Score
ggplot(cleaned_data, aes(x = Math_Score, y = English_Score)) +
  geom_point(color = "darkblue") +
```

```
  geom_smooth(method = "lm", se = FALSE, color = "red") +
  labs(title = "Math Scores vs. English Scores",
       x = "Math Score",
       y = "English Score")

# Boxplot of Math_Score by Age
ggplot(cleaned_data, aes(x = factor(Age), y = Math_Score)) +
  geom_boxplot(fill = "lightblue") +
  labs(title = "Boxplot of Math Scores by Age",
       x = "Age",
       y = "Math Score")
```

Output:

```
# Summary statistics of cleaned_data
     Name                Age         Math_Score     English_Score
 Length:6           Min.   :14.00   Min.   :60.0   Min.   :58.00
 Class :character   1st Qu.:15.25   1st Qu.:70.0   1st Qu.:70.00
 Mode  :character   Median :16.00   Median :75.0   Median :78.00
                    Mean   :16.00   Mean   :78.333   Mean   :80.00
                    3rd Qu.:17.00   3rd Qu.:90.0   3rd Qu.:88.00
                    Max.   :18.00   Max.   :95.0   Max.   :92.00

# Scatter plot with regression line and boxplot as described
```

```

### Saving and Exporting Data

Code Example:

```r
# Save the cleaned data to a new CSV file
write.csv(cleaned_data, "cleaned_students_scores.csv", row.names = FALSE)

# Verify by reading the saved file
new_data <- read.csv("cleaned_students_scores.csv")
head(new_data)
```

Output:
```
     Name Age Math_Score English_Score
1   Alice  16        85            78
2     Bob  17        90            88
3 Charlie  15        70            65
4   Diana  18        95            92
5    Evan  14        60            58
6   Fiona  16        75            70
```
```

---

13. Additional Advanced Topics (Optional Extensions)

To further enhance your students' R programming skills, consider introducing the following advanced topics:

a. Data Manipulation with dplyr

Theory:

- dplyr is a powerful R package for data manipulation, providing functions for filtering, selecting, mutating, summarizing, and arranging data.

Code Example:

```r
# Install and load dplyr
install.packages("dplyr")
library(dplyr)

# Using mtcars dataset to demonstrate dplyr functions

# Filter cars with mpg greater than 20
filtered_data <- filter(mtcars, mpg > 20)
```

```
# Select specific columns
selected_data <- select(filtered_data, mpg, cyl, hp)

# Create a new column with hp per cylinder
mutated_data <- mutate(selected_data, hp_per_cyl = hp / cyl)

# Summarize average hp_per_cyl by number of cylinders
summary_data <- mutated_data %>%
  group_by(cyl) %>%
  summarise(avg_hp_per_cyl = mean(hp_per_cyl))

# View the summary
summary_data
```

Output:

```
# A tibble: 3 × 2
    cyl avg_hp_per_cyl
  <dbl>          <dbl>
1     4           35
2     6           35.5
3     8           35.4
```

b. Data Visualization with Shiny

Theory:

- Shiny is an R package that makes it easy to build interactive web applications directly from R.

- Useful for creating dashboards and interactive data exploration tools.

Code Example:

```r
# Install and load Shiny

install.packages("shiny")

library(shiny)


# Define UI for the application

ui <- fluidPage(

  titlePanel("Interactive mtcars Dataset"),

  sidebarLayout(

    sidebarPanel(

      selectInput("xvar", "X-axis:", choices = names(mtcars)),

      selectInput("yvar", "Y-axis:", choices = names(mtcars), selected = names(mtcars)[[2]])

    ),

    mainPanel(

      plotOutput("scatterPlot")

    )
```

```
  )
)

# Define server logic
server <- function(input, output) {
  output$scatterPlot <- renderPlot({
    ggplot(mtcars, aes_string(x = input$xvar, y = input$yvar)) +
      geom_point(color = "darkgreen") +
      labs(title = paste("Scatter Plot of", input$yvar, "vs.", input$xvar))
  })
}

# Run the application
shinyApp(ui = ui, server = server)
```

Output:

- Launches a Shiny app where users can select variables for the X and Y axes to generate dynamic scatter plots.

c. Introduction to Machine Learning with caret

Theory:

- caret is a comprehensive R package for building machine learning models, offering tools for data splitting, pre-processing, feature selection, model tuning, and evaluation.

Code Example:

```r
# Install and load caret
install.packages("caret")
library(caret)

# Using the iris dataset for classification

# Set seed for reproducibility
set.seed(123)

# Split the data into training and testing sets (70-30 split)
trainIndex <- createDataPartition(iris$Species, p = 0.7,
                                  list = FALSE,
                                  times = 1)
irisTrain <- iris[ trainIndex,]
irisTest  <- iris[-trainIndex,]

# Train a decision tree model
model <- train(Species ~ ., data = irisTrain, method = "rpart")

# Print the model
print(model)
```

```
# Make predictions on the test set
predictions <- predict(model, irisTest)

# Confusion Matrix
confusionMatrix(predictions, irisTest$Species)
```

Output:

```
# Model details
Decision Tree

150 samples
  4 predictor
  3 classes: 'setosa', 'versicolor', 'virginica'

# Confusion Matrix
            Reference
Prediction   setosa versicolor virginica
  setosa        10         0         0
  versicolor     0        10         1
  virginica      0         2        12
```

Overall Statistics

...

```

Interpretation:

- The confusion matrix shows how well the model predicted the species of iris flowers in the test set.

- Metrics such as accuracy, sensitivity, and specificity can be derived from the confusion matrix.

---

## 14. Resources and Further Learning

To support your teaching and provide students with additional learning materials, consider the following resources:

- R Documentation: Comprehensive documentation for all R functions and packages. Access it using `?function_name` in R or visit [CRAN](https://cran.r-project.org/manuals.html).

- Online Tutorials:

  - [R for Data Science](https://r4ds.had.co.nz/) by Hadley Wickham & Garrett Grolemund

  - [Swirl](https://swirlstats.com/) – Learn R programming and data science interactively within R.

- Books:

  - *"Hands-On Programming with R"* by Garrett Grolemund

  - *"R Graphics Cookbook"* by Winston Chang
```

- Communities:

  - [Stack Overflow](https://stackoverflow.com/questions/tagged/r) – Ask questions and find answers related to R.

  - [RStudio Community](https://community.rstudio.com/) – Engage with other R users and developers.

---

15. Practical Project Ideas

Encourage your students to apply their R programming skills by working on practical projects. Here are some ideas:

1. Student Attention Span Analysis:

   - Utilize data collected from smartwatches to analyze attention spans.

   - Visualize trends and identify factors affecting attention.

2. Ganeshotsav Event Data:

   - Collect data from the cultural night event (e.g., participant feedback, activity engagement).

   - Perform EDA and visualize the results to improve future events.

3. Sports Performance Tracker:

   - Analyze sports performance data, such as running times or scores.

   - Use statistical tests to compare different training methods.

4. Environmental Data Dashboard:

- Create a Shiny app to monitor and visualize environmental metrics like temperature, humidity, and air quality.

5. Survey Data Analysis:

- Design a survey on a topic of interest, collect responses, and perform data analysis to derive insights.
