Section 4 - R Programs

Mohammad Saqib Ansari

2023-11-30

Creating Programs with R

Loops (for or while)

```
# Example 1: Basic for loop iterating from 1 to 5
for(i in 1:5) {
  print(i)
}
```

For Loops

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

This for loop iterates over the sequence from 1 to 5 and prints each value of `i`.

```
# Example 2: For loop with sequence increments of 2
for(i in seq(1, 10, by = 2))
  print(i)
```

```
## [1] 1
## [1] 3
## [1] 5
## [1] 7
## [1] 9
```

In this loop, the sequence runs from 1 to 10 with an increment of 2 for each iteration, printing each value of `i`.

```
# Example 3: For loop iterating through a character vector
days <- c("Mon", "Tue", "Wed")
for(i in days)
  print(i)
```

```
## [1] "Mon"
## [1] "Tue"
## [1] "Wed"
```

Here, the loop iterates through the elements in the `days` vector and prints each day.

```
# Example 4: Basic while loop with a conditional statement
i <- 1
while(i < 3) {
  print(i)
  i <- i + 1
}
```

While Loop

```
## [1] 1
## [1] 2
```

This while loop runs as long as the condition `i < 3` is true, printing the value of `i` and incrementing it until it reaches 3.

Conditional Statements (if-else)

```
# Example 5: Basic if-else conditional statement
come <- "late"
if(come == "early") {
  print("Don't cook food.")
} else {
  print("Cook food.")
}
```

```
## [1] "Cook food."
```

This checks the value of `come`. If it's "early," it prints a message advising not to cook food; otherwise, it prints a message to cook food.

```
# Example 6: Repeat loop with a break statement
i <- 1
repeat {
  print(i)
  i <- i + 1
  if(i > 3)
    break
}
```

Repeat Loop

```
## [1] 1
## [1] 2
## [1] 3
```

The repeat loop continues indefinitely printing `i` and incrementing it, but it breaks when `i` exceeds 3.

Explanation: - The R Markdown document showcases different types of loops (for and while), conditional statements (if-else), and the repeat loop in R programming. - Each code example is explained, demonstrating the functionality and purpose of the code blocks in R. - The explanations aim to provide a clear understanding of how each loop or conditional statement operates within the context of the R programming language.

Predefined Functions

```
# Generating a matrix X and displaying it
set.seed(1234)
X <- matrix(sample(1:20, 20), ncol = 4)
X
```

Calculate mean of each column

```
##      [,1] [,2] [,3] [,4]
## [1,]   16   19   14    8
## [2,]    5    6   10   17
## [3,]   12    4   11    1
## [4,]   15    2   20   18
## [5,]    9    7   13    3
```

A matrix `X` of random numbers between 1 and 20 is created with 4 columns.

```
# Applying the 'apply' function to calculate column-wise mean
apply(X, MARGIN = 2, FUN = mean)  # MARGIN 1 for rows, 2 for columns
```

```
## [1] 11.4  7.6 13.6  9.4
```

The `apply` function calculates the mean of each column (`MARGIN = 2`) in matrix `X`.

```
# Applying 'apply' function with additional argument to remove NA values
X[1, 1] <- NA  # Introducing NA value
apply(X, MARGIN = 2, FUN = mean, na.rm = TRUE)
```

```
## [1] 10.25  7.60 13.60  9.40
```

This applies `mean` function to each column of `X` while ignoring NA values.

```
# Using colMeans and colSums as shortcuts for column-wise operations
colMeans(X, na.rm = TRUE)  # Calculate column means while removing NA values
```

```
## [1] 10.25  7.60 13.60  9.40
```

```r
colSums(X)  # Calculate column sums
```

```
## [1] NA 38 68 47
```

```r
rowSums(X, na.rm = TRUE)  # Calculate row sums while ignoring NA values
```

```
## [1] 41 38 28 55 32
```

```r
# Applying 'apply' function to an array Y with margin 3 for depth
set.seed(1234)
Y <- array(sample(24), dim = c(4, 3, 2))
Y
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]   16   15   24
## [2,]   22    9    4
## [3,]    5   23    2
## [4,]   12    6    7
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]   17   11   18
## [2,]   10    8    1
## [3,]   21   13   14
## [4,]   20   19    3
```

```r
apply(Y, MARGIN = c(1, 2), FUN = sum, na.rm = TRUE)
```

```
##      [,1] [,2] [,3]
## [1,]   33   26   42
## [2,]   32   17    5
## [3,]   26   36   16
## [4,]   32   25   10
```

Here, `apply` calculates the sum across dimensions 1 and 2 of array `Y`, effectively summing across rows and columns.

```r
# Defining a custom function MyFunc and applying it using 'apply'
MyFunc <- function(x, y) {
  z = x + sqrt(y)
  return(1 / z)
}

set.seed(1234)
M <- matrix(sample(20), ncol = 4)
apply(M, MARGIN = c(1, 2), FUN = MyFunc, y = 2)
```

```
##            [,1]       [,2]       [,3]       [,4]
## [1,] 0.05742436 0.04898548 0.06487519 0.10622236
## [2,] 0.15590376 0.13487607 0.08761007 0.05430588
## [3,] 0.07454779 0.18469903 0.08055283 0.41421356
## [4,] 0.06092281 0.29289322 0.04669796 0.05150865
## [5,] 0.09602261 0.11884652 0.06937597 0.22654092
```

The function `MyFunc` performs a custom calculation based on inputs `x` and `y`, and then `apply` is used to apply this function across rows and columns of matrix `M`.

```
## tapply function applies a function on factors or combinations of factors

# Creating vectors and displaying them
z <- 1:5
vec1 <- c(rep("A1", 2), rep("A2", 2), rep("A3", 1))
vec1
```

```
## [1] "A1" "A1" "A2" "A2" "A3"
```

```
vec2 <- c(rep("B1", 3), rep("B2", 2))
vec2
```

```
## [1] "B1" "B1" "B1" "B2" "B2"
```

```
tapply(z, vec1, sum)  # Applying 'sum' to 'z' based on 'vec1'
```

```
## A1 A2 A3
##  3  7  5
```

```
tapply(z, list(vec1, vec2), sum)  # Applying 'sum' to 'z' based on combinations of 'vec1' and 'vec2'
```

```
##    B1 B2
## A1  3 NA
## A2  3  4
## A3 NA  5
```

The `tapply` function applies the `sum` function to elements of vector `z` based on factors defined by `vec1` and combinations of `vec1` and `vec2`.

```
# sapply function is equivalent to lapply, yielding a matrix or vector
set.seed(545)

mat1 <- matrix(sample(12), ncol = 4)
mat1
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   11    2    7    4
## [2,]    6    3   10    8
## [3,]    9   12    5    1
```

```r
mat2 <- matrix(sample(4), ncol = 2)
mat2
```

```
##      [,1] [,2]
## [1,]    4    1
## [2,]    3    2
```

```r
mylist <- list(matrix1 = mat1, matrix2 = mat2)

lapply(mylist, mean)
```

```
## $matrix1
## [1] 6.5
##
## $matrix2
## [1] 2.5
```

```r
# Using apply function as FUNC function
lapply(mylist, apply, 2, sum, na.rm = TRUE)
```

```
## $matrix1
## [1] 26 17 22 13
##
## $matrix2
## [1] 7 3
```

```r
# The aggregate function works on data frames
Z <- 1:5
T <- 5:1
vec1 <- c(rep("A1", 2), rep("A2", 2), rep("A3", 1))
vec2 <- c(rep("B1", 3), rep("B2", 2))
df <- data.frame(Z, T, vec1, vec2)
df
```

```
##   Z T vec1 vec2
## 1 1 5   A1   B1
## 2 2 4   A1   B1
## 3 3 3   A2   B1
## 4 4 2   A2   B2
## 5 5 1   A3   B2
```

```r
# Using aggregate to perform operations on data frame columns based on factors
aggregate(df[, 1:2], list(FactorA = vec1), sum)
```

```
##   FactorA Z T
## 1      A1 3 9
## 2      A2 7 5
## 3      A3 5 1
```

```r
# Defining subgroups using vectors generated by two factors for aggregate
aggregate(df[, 1:2], list(factorA = vec1, factorB = vec2), sum)
```

```
##   factorA factorB Z T
## 1      A1      B1 3 9
## 2      A2      B1 3 3
## 3      A2      B2 4 2
## 4      A3      B2 5 1
```

```r
# The sweep function applies a single procedure to all margins
set.seed(1234)
X <- matrix(sample(12), ncol = 3)

mean_X <- apply(X, 2, mean)
mean_X
```

```
## [1] 8.25 5.25 6.00
```

```r
sd_X <- apply(X, 2, sd)
sd_X
```

```
## [1] 3.304038 3.500000 4.242641
```

```r
Xc <- sweep(X, 2, mean_X, FUN = "-")
Xc
```

```
##       [,1]  [,2] [,3]
## [1,]  3.75 -1.25   -4
## [2,]  1.75  1.75    2
## [3,] -2.25 -4.25    5
## [4,] -3.25  3.75   -3
```

```r
Xcr <- sweep(Xc, 2, sd_X, FUN = "/")
Xcr
```

```
##            [,1]       [,2]       [,3]
## [1,]  1.1349749 -0.3571429 -0.9428090
## [2,]  0.5296549  0.5000000  0.4714045
## [3,] -0.6809849 -1.2142857  1.1785113
## [4,] -0.9836449  1.0714286 -0.7071068
```

Explanation and execution:

- The code demonstrates the usage of various functions in R including `lapply`, `apply`, `aggregate`, and `sweep`.
- `lapply` and `sapply` are used to apply functions across lists or vectors.
- `aggregate` showcases aggregating data frame columns based on factors or combinations of factors.
- Lastly, `sweep` is illustrated for applying a single procedure across all margins of a matrix.

```r
# Using scale() function - Xcr is equivalent to scale
scale(X)  # Xcr is equivalent to using scale function
```

```
##            [,1]       [,2]       [,3]
## [1,]  1.1349749 -0.3571429 -0.9428090
## [2,]  0.5296549  0.5000000  0.4714045
## [3,] -0.6809849 -1.2142857  1.1785113
## [4,] -0.9836449  1.0714286 -0.7071068
## attr(,"scaled:center")
## [1] 8.25 5.25 6.00
## attr(,"scaled:scale")
## [1] 3.304038 3.500000 4.242641
```

```r
# Generating data
set.seed(1234)
T <- rnorm(100)
Z <- rnorm(100) + 3 * T + 5

vec1 <- c(rep("A1", 25), rep("A2", 25), rep("A3", 50))
don <- data.frame(Z, T)

# Using 'by' function to perform operations by groups
by(don, list(FactorA = vec1), summary)  # Summary statistics by group
```

```
## FactorA: A1
##        Z                T          
##  Min.   :-2.540   Min.   :-2.3457  
##  1st Qu.: 2.380   1st Qu.:-0.7763  
##  Median : 3.662   Median :-0.4907  
##  Mean   : 4.331   Mean   :-0.2418  
##  3rd Qu.: 5.737   3rd Qu.: 0.2774  
##  Max.   :12.221   Max.   : 2.4158  
## ------------------------------------------------------------ 
## FactorA: A2
##        Z                T          
##  Min.   :-1.856   Min.   :-2.1800  
##  1st Qu.: 1.520   1st Qu.:-1.1073  
##  Median : 3.071   Median :-0.8554  
##  Mean   : 2.991   Mean   :-0.6643  
##  3rd Qu.: 4.050   3rd Qu.:-0.4659  
##  Max.   : 9.321   Max.   : 1.4495  
## ------------------------------------------------------------ 
## FactorA: A3
##        Z                 T           
##  Min.   :-0.7953   Min.   :-1.80603  
##  1st Qu.: 3.2903   1st Qu.:-0.56045  
##  Median : 5.1260   Median :-0.04396  
##  Mean   : 5.4811   Mean   : 0.13953  
##  3rd Qu.: 8.0739   3rd Qu.: 0.81208  
##  Max.   :12.6221   Max.   : 2.54899
```

```r
by(don, list(FactorA = vec1), sum)  # Sum of variables by group
```

```
## FactorA: A1
## [1] 102.2201
## ------------------------------------------------------------
## FactorA: A2
## [1] 58.16823
## ------------------------------------------------------------
## FactorA: A3
## [1] 281.0313
```

```r
# Calculating regression coefficients for each level of the variable vec1
myfunction <- function(x) {
  summary(lm(Z ~ T), data = x)$coef
}
by(don, vec1, myfunction)
```

```
## vec1: A1
##             Estimate Std. Error  t value     Pr(>|t|)
## (Intercept) 5.037154  0.1049788 47.98260 7.093891e-70
## T           2.973915  0.1037759 28.65709 1.956630e-49
## ------------------------------------------------------------
## vec1: A2
##             Estimate Std. Error  t value     Pr(>|t|)
## (Intercept) 5.037154  0.1049788 47.98260 7.093891e-70
## T           2.973915  0.1037759 28.65709 1.956630e-49
## ------------------------------------------------------------
## vec1: A3
##             Estimate Std. Error  t value     Pr(>|t|)
## (Intercept) 5.037154  0.1049788 47.98260 7.093891e-70
## T           2.973915  0.1037759 28.65709 1.956630e-49
```

```r
# Using 'replicate' function to repeat a process n times
set.seed(1234)
replicate(n = 8, mean(rnorm(100)))
```

```
## [1] -0.1567617424  0.0412431799  0.1546036721 -0.0081051362 -0.0217858703
## [6] -0.1368770057 -0.0878617963 -0.0008371926
```

```r
# Using 'outer' function to perform operations for combinations of vectors
Month <- c("Jan", "Feb", "Mar")
Year <- 2008:2010
outer(Month, Year, FUN = "paste")  # Generating combinations of Month and Year
```

```
##      [,1]       [,2]       [,3]
## [1,] "Jan 2008" "Jan 2009" "Jan 2010"
## [2,] "Feb 2008" "Feb 2009" "Feb 2010"
## [3,] "Mar 2008" "Mar 2009" "Mar 2010"
```

```r
outer(Month, Year, FUN = paste, sep = "-")  # Combinations with a separator
```

```
##      [,1]       [,2]       [,3]
## [1,] "Jan-2008" "Jan-2009" "Jan-2010"
## [2,] "Feb-2008" "Feb-2009" "Feb-2010"
## [3,] "Mar-2008" "Mar-2009" "Mar-2010"
```

```r
# Function to sum 1:n
mysum <- function(n) {
  result <- sum(1:n)
  return(result)
}
mysum(3)  # Example usage of mysum function
```

```
## [1] 6
```

```r
# Improved version of mysum function with checks for positive integers
mysum <- function(n) {
  if (n < 0) stop("n must be a positive integer")
  if (floor(n) != n) warning(paste("rounding", n, "to", floor(n)))
  result <- sum(1:n)
  return(result)
}
mysum(4.325)  # Example usage with a non-integer value
```

```
## Warning in mysum(4.325): rounding 4.325 to 4
```

```
## [1] 10
```

```r
# Function to find solutions of quadratic equations
Quad_soln <- function(a, b, c) {
  D <- b^2 - 4 * a * c
  D <- as.complex(D)
  x <- (-b + sqrt(D)) / 2
  y <- (-b - sqrt(D)) / 2
  return(paste("The solutions are", x, "and", y))
}

# Calculating factorial using prod function
n <- 8
x <- 1:n
prod(x)  # Factorial using prod function
```

```
## [1] 40320
```

```r
# Calculating factorial using for loops
n <- 7
num <- 1
for (i in 1:n) {
  num <- num * i
}
print(num)  # Factorial using for loops
```

```
## [1] 5040
```

Explanation:

- The provided R code includes various functionalities and functions in R programming.
- It demonstrates the usage of functions like `scale`, `by`, `replicate`, `outer`, and custom functions like calculating factorial, quadratic equation solutions, and summary statistics for grouped data.
- Functions such as `scale`, `by`, and `outer` are used for scaling data, performing operations by groups, and creating combinations, respectively.
- Additionally, functions for calculating factorial, solving quadratic equations, and summaries for grouped data using custom functions are presented.