Lesson 7-Key!

YOUR NAME

2023-10-24

Learning goals

- Using R's built-in loop functions: `apply`, `lapply` and contrasting them with `for` loops
- Investigate sampling error
- Visualize confidence intervals.
- Adding error bars to plots
- Conditional statements
- Functions

Apply and lapply

Last week you learned about loops, which are standard in all programming languages. However, R has some built in functions that perform similar operations and are a bit more simple to use. They belong to the apply family. Here are some differences between apply and lapply Differences between lapply and apply: Data Types:

- lapply works on lists and vectors.
- apply works on matrices and arrays.

Return Type:

- lapply always returns a list.
- apply can return a vector, matrix, or array, depending on the function being applied and the input's dimensions.

Use Case:

- Use lapply when you have a list or vector and you want to apply a function to each of its elements.
- Use apply when you have a matrix or array and you want to apply a function across its rows or columns.

There are many other functions in the apply family but we are just focusing on these two. I recommend typing ? apply and ?lapply into the console before starting. The second argument, margin, can be tricky.

```
mat <- matrix(1:12, nrow=3)
column_sums <- apply(mat, 2, sum)
print(column_sums)
```

1. Let's start super basic. Say you have a matrix mat <- matrix(1:12, nrow=3). You want to find the sum of each column. paste the matrix first then use apply to find the sum. Name it something then print the result

```
## [1]  6 15 24 33
```

2. Now you have a list of names. Use lapply to find how many characters are in each name. Save as an object+print Here are your names, get counting names_list <- list("Alessandra", "Penelope", "Meigui", "Tatiana", "Keisuke", "Nadia", "Amy")

```
names_list <- list("Alessandra", "Penelope", "Meigui", "Tatiana", "Keisuke", "Nadia",
↪  "Amy")
char_count <- lapply(names_list, nchar)
print(char_count)
```

```
## [[1]]
## [1] 10
##
## [[2]]
## [1] 8
##
## [[3]]
## [1] 6
##
## [[4]]
## [1] 7
##
## [[5]]
## [1] 7
##
## [[6]]
## [1] 5
##
## [[7]]
## [1] 3
```

What is different about this output? Correct, it's a list

3. Comparing Apply and For loops You're a teacher and you've just given your students a test. The test scores are stored in a matrix where each row represents a student and each column represents a test question. Calculate the average score for each student using both apply and a for loop. Your matrix is called scores and has all the information you need.

```r
scores <- matrix(c(85, 90, 78, 88,
                   92, 80, 95, 90,
                   60, 55, 78, 84),
                 nrow=3, byrow=TRUE)
colnames(scores) <- c("Math", "English", "Biology", "History")
rownames(scores) <- c("Alice", "Bob", "Charlie")
```

First use the new functions you learned to find the average test score for each student.

“‘{r. ex3.1 } averages_apply <- apply(scores, 1, mean)

~~~
Now use a for loop. remember to create blank variables first to store your information




```r
num_students <- nrow(scores)
averages <- numeric(num_students)

for (i in 1:num_students) {
  averages[i] <- mean(scores[i,])
}

names(averages) <- rownames(scores)
~~~

Which of these was easier for you? Which student is doing the best?

Which class do you think is hardest? Lets do the process again to find the average for each class First use apply “‘{r. ex3.3 }

class_averages_apply <- apply(scores, 2, mean)

~~~
Then use a loop


```r
num_subjects <- ncol(scores)
class_averages <- numeric(num_subjects)

for (j in 1:num_subjects) {
  class_averages[j] <- mean(scores[,j])
}

names(class_averages) <- colnames(scores)
~~~

And now you understand the joy of the apply family.

Conditional Statements

Conditional Statements are a key part of any programming language. 1. if Statement: The if statement evaluates a condition, and if that condition is TRUE, it executes the code inside the statement.

```r
x <- 10

if (x > 5) {
  print("x is greater than 5")
}
```

```
## [1] "x is greater than 5"
```

In the above code, since x is indeed greater than 5, the message "x is greater than 5" will be printed.

2. if-else Statement: There might be cases where you want to execute one set of instructions if the condition is TRUE and another set if it's FALSE. This is where the if-else statement comes in.

```r
x <- 3

if (x > 5) {
  print("x is greater than 5")
} else {
  print("x is not greater than 5")
}
```

```
## [1] "x is not greater than 5"
```

You can keep adding else statements!

3. if-else if-else Ladder: For multiple conditions, we can use an if-else if-else ladder.

```r
x <- 5

if (x > 10) {
  print("x is greater than 10")
} else if (x > 5) {
  print("x is greater than 5 but less than or equal to 10")
} else {
  print("x is less than or equal to 5")
}
```

```
## [1] "x is less than or equal to 5"
```