Class06: R Functions

Julia Di Silvestri (PID: A16950824)

2024-01-25

R Functions

Functions are how we get stuff done. We call functions to do everything useful in R.

One cool thing about R is that it makes writing your own functions comparatively easy.

All functions in R have at least three things:

- A **name** (we get to pick this)
- One or more **input arguments** (the input to our function)
- The **body** (lines of code that do the work)

`#| eval: false` will allow invalid code to render by echoing it

```
funname <- function(input1, input2) {
  #The body with  R code
}
```

Let's write a silly first function to add two numbers:

```
x <- 5
y <- 1
x + y
```

```
[1] 6
```

```
addme <- function(x, y) { x + y}
```

```
addme(279, 5678)
```

```
[1] 5957
```

To assign one element with a default:

```
addme <- function(x, y=1) {x + y}
```

```
addme(10)
```

```
[1] 11
```

Lab for Today

Question 1

Writing a "grade" function

First, we assign vectors to each student with each of their grades:

```
student1 <- c(100, 100, 100, 100, 100, 100, 100, 90)
student2 <- c(100, NA, 90, 90, 90, 90, 97, 80)
student3 <- c(90, NA, NA, NA, NA, NA, NA, NA)
```

Next, we will start to form the function. Step one is to get the function to identify a student's lowest score:

```
which.min(student1)
```

```
[1] 8
```

```
which.min(student2)
```

```
[1] 8
```

```
which.min(student3)
```

```
[1] 1
```

Next, we will find the average of each student's scores:

```r
mean(student1)
```

```
[1] 98.75
```

```r
mean(student2, na.rm = T)
```

```
[1] 91
```

```r
mean(student3, na.rm = T)
```

```
[1] 90
```

This is currently not fair – student 3 should not have a mean of 90.

We will move on for now. Things worked for student 1. Now we want to drop the lowest score before getting the `mean()`. Using vector[-x] will spit out every value in that vector except for the xth value.

```r
# Find lowest score
which.min(student1)
```

```
[1] 8
```

```r
# Remove lowest score
student1[-8]
```

```
[1] 100 100 100 100 100 100 100
```

Now to put it together:

```r
# Find mean with lowest score removed
mean(student1[-which.min(student1)])
```

```
[1] 100
```

Nice it worked!

A common shortcut and use `x` as my input so that I don't have to keep typing `student1`

```r
x <- student1
mean( x[ -which.min(x)])
```

```
[1] 100
```

We still have the problem of missing values.

One idea is to replace NA values with zero.

```r
y <- 1:5
y[y == 3] <- 10000
y
```

```
[1]     1     2 10000     4     5
```

^ This method will not work for NAs because there is not data to change in the case of an NA

To find location if NA in a vector:

```r
is.na(student2)
```

```
[1] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
```

How to set NA to 0:

```r
b <- student2
b[is.na(b)] <- 0
mean( b[ -which.min(b)])
```

```
[1] 91
```

Now writing a function with this:

```r
grade <- function(x) {
  #change NA to 0
  x[is.na(x)] <- 0
  #drop lowest score and average
mean( x[ -which.min(x)])}
```

```
grade(student1)
```

```
[1] 100
```

```
grade(student2)
```

```
[1] 91
```

```
grade(student3)
```

```
[1] 12.85714
```

To import CSV file:

```
url <- "https://tinyurl.com/gradeinput"
gradebook <- read.csv(url, row.names = 1)
gradebook
```

```
           hw1 hw2 hw3 hw4 hw5
student-1  100  73 100  88  79
student-2   85  64  78  89  78
student-3   83  69  77 100  77
student-4   88  NA  73 100  76
student-5   88 100  75  86  79
student-6   89  78 100  89  77
student-7   89 100  74  87 100
student-8   89 100  76  86 100
student-9   86 100  77  88  77
student-10  89  72  79  NA  76
student-11  82  66  78  84 100
student-12 100  70  75  92 100
student-13  89 100  76 100  80
student-14  85 100  77  89  76
student-15  85  65  76  89  NA
student-16  92 100  74  89  77
student-17  88  63 100  86  78
student-18  91  NA 100  87 100
student-19  91  68  75  86  79
student-20  91  68  76  88  76
```

To apply our function to “gradebook”, we use `apply()`:

```
# apply function works like: apply(dataframe/matrix, 1 (row) or 2 (col) or c(1,2) (both),
results <- apply(gradebook, 1, grade)
results
```

```
 student-1  student-2  student-3  student-4  student-5  student-6  student-7
     91.75      82.50      84.25      84.25      88.25      89.00      94.00
 student-8  student-9 student-10 student-11 student-12 student-13 student-14
     93.75      87.75      79.00      86.00      91.75      92.25      87.75
student-15 student-16 student-17 student-18 student-19 student-20
     78.75      89.50      88.00      94.50      82.75      82.75
```

Question 2

To pull out the top scoring student, use `which.max()`

```
#who scored the highest
which.max(results)
```

```
student-18
        18
```

```
#what did they score
max(results)
```

```
[1] 94.5
```

Question 3

To determine the homework averages across the whole class, we will apply a function over the columns instead of the rows

```
apply(gradebook, 2, mean)
```

```
 hw1  hw2  hw3  hw4  hw5
89.0   NA 80.8   NA   NA
```

This did not work because we need to write a new function that drops the NAs again

```
hwavgs <- apply(gradebook, 2, mean, na.rm = T)
hwavgs
```

```
     hw1      hw2      hw3      hw4      hw5 
89.00000 80.88889 80.80000 89.63158 83.42105 
```

Now we can pull out the minimum from these answers:

```
# which homework has the lowest average
which.min(hwavgs)
```

```
hw3 
  3 
```

```
# what was the average
min(hwavgs)
```

```
[1] 80.8
```

Now we will use another method to try to weed out biases in the average

```
hwsums <- apply(gradebook, 2, sum, na.rm = T)
which.min(hwsums)
```

```
hw2 
  2 
```

This answer indicates that there was an outlier in HW3 that was skewing the average. Homework two was more consistently the low-scoring.

Question 4

```
# make all (or mask) NAs to zero
mask <- gradebook
mask[is.na(mask)] <- 0
```

We can use the `cor()` function for correlation analysis

```
cor(mask$hw5, results)
```

```
[1] 0.6325982
```

```
cor(mask$hw3, results)
```

```
[1] 0.3042561
```

Homework 5 is much more correlated. How do we apply this across the whole gradebook?

```
cors <- apply(mask, 2, cor, results)
cors
```

```
      hw1       hw2       hw3       hw4       hw5 
0.4250204 0.1767780 0.3042561 0.3810884 0.6325982 
```

We can see that hw5 is the most correlated, and that maybe hw2 should be reconsidered.