Class06:Functions in R

Lilith Sadil, A16470107

Background: Functions

Functions are how we get work done in R; we call functions to do everything from reading data to doing analysis & outputting plots/results

All functions in R have at least 3 components:

- A **name** (you get to pick this)
- Input **arguments** (there can be 1 or more)
- **Body** (the code between `{}` which determines what the function does)

Example basic function

This function will do something very basic: it adds numbers We'll call it `add()`

First, we'll write a simple snippet of code which the body of our function will be based on:

```r
x = 10
y = 10
x + y
```

```
[1] 20
```

Next, we'll define the function with `x` as our input parameter. Now, if we define the value of `x`, we can add it to `y`

```r
add = function(x) {
  y=10
  x+y
}
```

When we try to call this function, we get an error - we must run it first so that the R “brain” is aware that the function exists

```r
add(1)
```

```
[1] 11
```

Next, we’ll modify the function so that both x and y can be input by the user

```r
add = function(x,y){
  x+y
}
```

When we input x=10 and y=5, the function will add them to get 15.

```r
add(x=10,y=5)
```

```
[1] 15
```

```r
#or
add(10, 5)
```

```
[1] 15
```

If we just input 1 parameter when we need 2 (both x and y), then we’ll get an error message

```r
#add(x=4)
```

Even if we had a function where one variable is defined within the function, we can override it through user input (ex. y=1 in the function but we tell the computer that y=10 after so y will be 10)

```r
add = function(x,y=1) {
  x+y
}
add(x=5,y=10)
```

```
[1] 15
```

Example 2: Grade Book Function

Here, we'll write a function to determine a student's grade from their assignment scores. The first three students' scores are shown as vectors below:

```
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)
```

Here is how to calculate the average grade of student 1:

```
sum(student1)/length(student1)
```

```
[1] 98.75
```

```
#or
mean(student1)
```

```
[1] 98.75
```

We can't use these same operations on students 2&3 since they have *"NA"* in their list of grades.

We have to use `na.rm` which is a logical value (TRUE/FALSE) which indicates whether or not NA values are removed before performing calculations;by default, na.rm = FALSE (i.e. NA values aren't removed) - we want to change this to "TRUE"

```
mean(student2, na.rm=TRUE)
```

```
[1] 91
```

For student 3, if we remove all the NA values, then they get a score of 90% for the class even though they only did one assignment.

```
mean(student3, na.rm=TRUE)
```

```
[1] 90
```

Instead, we want to calculate an average grade just by dropping the single lowest value (`min()` function)

```r
(sum(student1) - min (student1))/(length(student1)-1)
```

```
[1] 100
```

We can re-write the `min()` operation using a `range` operation where the output is [lowest value, highest value]. (grade_range[1]=min, grade_range[2]=max)

```r
grade_range=range(student1)
(sum(student1) - grade_range[1])/(length(student1)-1)
```

```
[1] 100
```

The assignment that student 1 did the poorest on was the 8th assignment in the list:

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

```
[1] 8
```

This gives us the lowest score of student1:

```r
student1[which.min(student1)]
```

```
[1] 90
```

To get rid of the 8th element (i.e. drop the lowest score), we can subtract the 8th element using `student1[-8]`

```r
student1[-8]
```

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

Now, to take the adjusted average of student1's grades:

```r
mean(student1[-8])
```

```
[1] 100
```

```
#or
mean(student1[-which.min(student1)])
```

```
[1] 100
```

To make things visually easier, we can replace "student1" with "x"; this will also allow us to perform the calculation with different students by changing the value of x

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

```
[1] 100
```

Still, this won't work with students 2&3 since they have "NA" scores. To fix this, we can replace all "NA" values with "0".

- First, we have to determine **where** the "NA" values are in each vector; using `is.na()` returns a "TRUE" value for the positions in a vector where "NA" is present:

```
x=student2
is.na(x)
```

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

Now, to make the "TRUE" values (for "NA") equal to "0",

```
x=student2
x[is.na(x)]=0
#This is the vector, x, where the NA values are replaced with 0
x
```

```
[1] 100   0  90  90  90  90  97  80
```

Now that NA is converted into "0", we can proceed to remove the lowest value and take the average

```
x=student3
x[is.na(x)]=0
```

```
#this part turns any NA into 0
mean(x[-which.min(x)])
```

```
[1] 12.85714
```

```
#this part removes the lowest score - in this case, 0
```

Now that we have the body written, we can define our function - let's call it "average_grade":

```
average_grade = function(x) {
  x[is.na(x)]=0
  mean(x[-which.min(x)])
}
average_grade(student2)
```

```
[1] 91
```

```
#we can change the argument to student1, 2, 3...
```

Questions

Question 1:

Write a function grade() to determine an overall grade from a vector of student homework assignment scores dropping the lowest single score. If a student misses a homework (i.e. has an NA value) this can be used as a score to be potentially dropped. Your final function should be adequately explained with code comments and be able to work on an example class gradebook such as this one in CSV format: "https://tinyurl.com/gradeinput"

First, we have to read the URL & we'll view the first few columns (`head`)

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

The function for determining a student's average grade (considering that the lowest score is dropped and NA counts as a "0") is:

```r
average_grade = function(x) {
  x[is.na(x)]=0
  mean(x[-which.min(x)])
}
average_grade(student2)
```

```
[1] 91
```

```r
#we can change the argument to student1, 2, 3...
```

~~~
The `apply()` function in R is useful but a bit confusing... Among other things, it can apply

::: {.cell}

```{.r .cell-code}
apply(gradebook, 1, average_grade)

 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:

Using your grade() function and the supplied gradebook, Who is the top scoring student overall in the gradebook?

Using the code above, we can use `which.max` to find which student has the maximum grade:

```
student_grades = apply(gradebook, 1, average_grade)
```

```
which.max(student_grades)
```

```
student-18
        18
```

Student 18 had the highest scores, overall, with an average of 94.50%.

Question 3:

From your analysis of the gradebook, which homework was toughest on students (i.e. obtained the lowest scores overall?

The average score for each assignment can be determined by changing the value in the `apply()` function from 1 to 2. Additionally, we use the mean (excluding NA) to calculate the average score of each column instead of the average_grade function:

```
apply(gradebook, 2, mean, na.rm=TRUE)
```

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

Now, we can look for which assignment had the `minimum` score

```
assignment_grades = apply(gradebook, 2, mean, na.rm=TRUE)
which.min(assignment_grades)
```

```
hw3
  3
```

```
min(assignment_grades)
```

```
[1] 80.8
```

Homework 3 had the worst performance with an average score of 80.8%.

Question 4:

From your analysis of the gradebook, which homework was most predictive of overall score (i.e. highest correlation with average grade score)?

```
#gives you the scores for homework 1 in the gradebook
gradebook$hw1
```

```
 [1] 100  85  83  88  88  89  89  89  86  89  82 100  89  85  85  92  88  91  91
[20]  91
```

```
cor(gradebook$hw1, student_grades)
```

```
[1] 0.4250204
```

```
#to apply this to all the homeworks, use apply
```

Let's make a copy of the gradebook where all NA values are 0:

```
gradebook_copy = gradebook
gradebook_copy[is.na(gradebook_copy)]=0
gradebook_copy
```

```
           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   0  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   0  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   0
student-16  92 100  74  89  77
student-17  88  63 100  86  78
student-18  91   0 100  87 100
student-19  91  68  75  86  79
student-20  91  68  76  88  76
```

```
cor(gradebook_copy$hw1, student_grades)
```

```
[1] 0.4250204
```

```
cor(gradebook_copy$hw2, student_grades)
```

```
[1] 0.176778
```

```
cor(gradebook_copy$hw3, student_grades)
```

```
[1] 0.3042561
```

```
cor(gradebook_copy$hw4, student_grades)
```

```
[1] 0.3810884
```

```
cor(gradebook_copy$hw5, student_grades)
```

```
[1] 0.6325982
```

Alternatively, we can use “apply” to run all these correlations at once

```
apply(gradebook_copy,2,cor,y=student_grades)
```

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

```
# the y=student_grades tells us that we're finding the correlation between the gradebook_c
# the function we're using is cor & everything after "cor" is additional information about
```

Using the `which.max` operation allows us to determine which homework has the highest correlation to student overall grade performance (homework 5, in this case):

```
which.max(apply(gradebook_copy,2,cor,y=student_grades))
```

```
hw5
  5
```

Question 5:

Make sure you save your Quarto document and can click the "Render" (or Rmarkdown"Knit") button to generate a PDF foramt report without errors. Finally, submit your PDF to gradescope