R Functions Lab (Class06)

Joshua Cheung

Work and notes to create the grade() function

We start with example input vectors:

```r
# Example input vectors to start with
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)
```

We first consider how to calculate the mean of the elements in a given vector:

```r
# We use the mean() function
# We use the student1 vector to demonstrate this
mean(student1)
```

```
## [1] 98.75
```

Since we need to identify the lowest score, it follows that we now need to determine which element of a given vector has the lowest value:

```r
# We use the which.min() function
# We use the student1 vector to demonstrate this
which.min(student1)
```

```
## [1] 8
```

We recall that our goal is to drop or exclude the lowest score of a given grade vector, before computing the mean of the elements of the vector:

```r
# we write a code that yields everything, except the minimum element of a given vector
# For instance if we wished to generate a list of everything except the second element of the student1 
student1[-2]
```

```
## [1] 100 100 100 100 100 100  90
```

```r
# We apply this rule to the minimum value of a particular vector
# For the student1 vector we apply this to the value: which.min(student1)
student1[-which.min(student1)]
```

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

Now we see that we can now proceed to computing the mean of a grade vector after excluding the lowest score:

```
# We apply the mean() function to student1[-which.min(student1)]
mean(student1[-which.min(student1)])
```

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

We now check to see if this work on the other example vectors we defined earlier:

```
# We see that we get NA when we run the following:
mean(student2[-which.min(student2)])
```

```
## [1] NA
```

```
mean(student3[-which.min(student3)])
```

```
## [1] NA
```

We see that this not what we wanted as we need numerical grade values.

We note that both vectors student2 and student3 having missing value or values denoted by NA, which is likely the source of our troubles.

Examination of the mean() function through the help function shows that the argument na.rm, which determined whether missing values should be removed before computation, is listed as FALSE by default.

We could change this value to TRUE to remove the missing values:

```
# We test this on the student3 vector
mean(student3, na.rm=TRUE)
```

```
## [1] 90
```

We once again recall that since our goal is to compute a the mean grade after dropping the lowest score. Setting na.rm to TRUE would likely not be equitable, especially in cases like the student3 vector, where the student has only completed one assignment but would still get a decent grade by this system.

Thus, it follows we should replace the missing values with a score of 0.

We first need to identify if a given vector has any NA elements:

```
# We use the is is.na() function to identify if a given vector has any NA elements
# We test this on the student3 vector
is.na(student3)
```

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

```
# Now is we wanted to the find the position in the vector that the NA element is in we use the followin
which(is.na(student3))
```

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

Thus we have the means to identify the NA elements of a given vector and now turn to how to consider how to replace these elements with 0:

```
# We perform a substitution as follows
# We test on the student3 vector
student3[which(is.na(student3))] <- 0
student3
```

```
## [1] 90  0  0  0  0  0  0  0
```

So we have successfully replaced all NA elements of the student3 vector with 0.

We now check that performing student3[-which.min(student3)]) removes only one of the zeros and not all of them:

```
student3[-which.min(student3)]
```

```
## [1] 90  0  0  0  0  0  0
```

We see only one 0 is removed so for the purposes of our project, we don't need to worry about all duplicates of the minimum score being dropped.

We now have all the information we need to compute the mean grade of a set of scores after dropping the lowest score:

```
# Given a vector x we would perform the following two line of code:
# Step 1: x[which(is.na(x))] <- 0
# Step 2: mean(x[-which.min(x)])
# We test this on the student3 vector
student3[which(is.na(student3))] <- 0
mean(student3[-which.min(student3)])
```

```
## [1] 12.85714
```

Questions

> **Q1. 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 adquately 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"**

We can combine everything we found in the first section to create a function capable to determining an overall grade from a vector of student homework assignment scores after dropping the lowest score.

The function grade() with code comments is as follows:

```
#' Calculate average score for a vector of student scores after dropping the lowest score
#' Missing values are replaced with a score of 0
#'
#' @param x A numeric vector of homework scores
#'
#' @return A average score
#' @export
```

```r
#'
#' @examples
#' student <- c(100, NA, 90, 97)
#' grade(student)
#'
grade <- function(x) {
  # Replace missing values with 0
  x[which(is.na(x))] <- 0
  # Exclude the lowest score from the mean
  mean(x[-which.min(x)])
}
```

we wish to use our function on an example class gradebook. Specifically, we use data from the following CSV format file: "https://tinyurl.com/gradeinput"

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

We can apply grade() over the gradebook to get average grades for all students in the example CSV file:

```r
apply(gradebook, 1, 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
```

> **Q2. Using your grade() function and the supplied gradebook, Who is the top scoring student overall in the gradebook?**

We run the apply() function again, and this time, save the results:

```r
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
```

To find the top scoring student we use the which.max() function:

```r
which.max(results)
```

```
## student-18
##         18
```

Thus, student 18 is the top scoring student overall in the gradebook

Q3. From your analysis of the gradebook, which homework was toughest on students (i.e. obtained the lowest scores overall?

We examine the gradebook:

```
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
```

We use compute median of each column, as the median is less likely to be affected by outliers:

```
median.scores <- apply(gradebook, 2, median, na.rm = TRUE)
median.scores
```

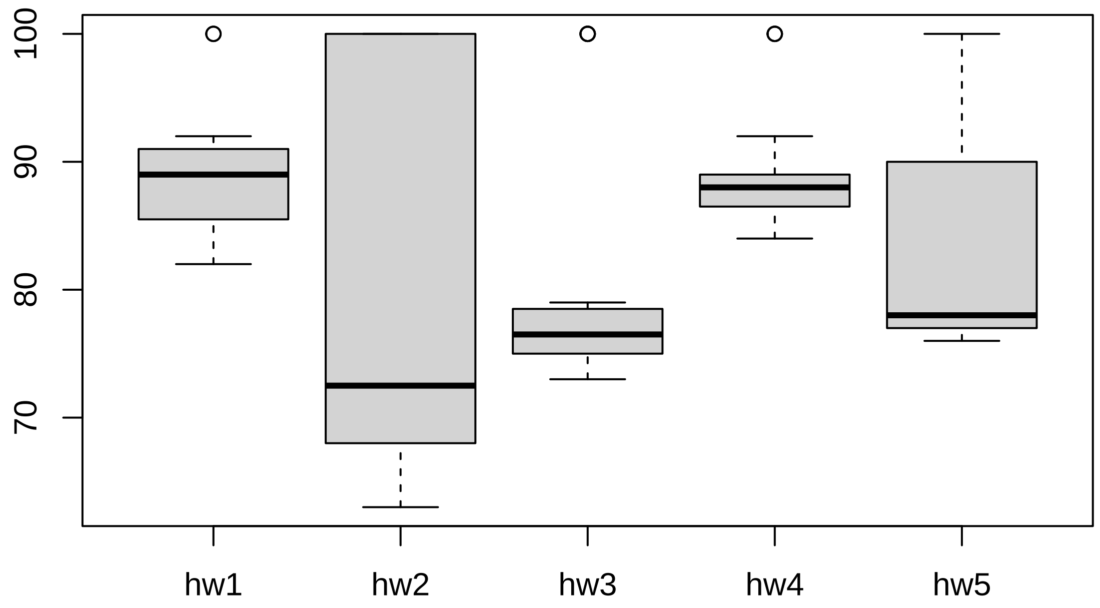
```
##  hw1  hw2  hw3  hw4  hw5
## 89.0 72.5 76.5 88.0 78.0
```

```
which.min(median.scores)
```

```
## hw2
##   2
```

To confirm our results we examine a basic box blot of the data from gradebook:

```
boxplot(gradebook)
```



We see that homework 2 has a very wide spread of scores with the median being pulled significantly down relative to the other assignments. Additionally he other homework assignment are more consistent in scores (homework 5 had some spread, but it is not as wide in range as homework 2).

Thus, homework 2 was the toughest on students as it has the lowest median score.

Q4. Optional Extension: From your analysis of the gradebook, which homework was most predictive of overall score (i.e. highest correlation with average grade score)?

We use the correlation function to see if the average score for each student is correlated with the scores for individual homeworks (which are the gradebook columns).

However first we replace all instance of NA in gradebook with 0:

```
masked.gradebook <- gradebook
masked.gradebook[is.na(masked.gradebook)] <- 0
masked.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   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
```

Now we can use the apply function to apply the correlation function across the entire gradebook:

```
apply(masked.gradebook, 2, cor, x=results)
```

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

Thus we see that homework 5 has the highest correlation value and has the highest correlation with average grade score.

> **Q5. Make sure you save your Rmarkdown document and can click the "Knit" button to generate a PDF format report without errors. Finally, submit your PDF to gradescope.**

We have indeed saved the Rmarkdown document. Hopefully, if the grader is reading this document, that indicates that we have successfully created a PDF document to upload to gradescope.