Lecture 9

Paul Holaway, Abhi Thanvi

July 6th, 2022

Lecture 8 Review

Before we move on to the new material, we will do a quick review of Lecture 8 content. Last time we learned about random number generation, conditionals/loops, and how to use them when creating a custom function. Random number generation is pretty straight forward once you have the syntax, so we will just do a quick review on conditionals/loops and using them in custom functions. Recall last time the grade assignment problem. Normally I do not redo an example, but since we are going to be continuing it today I will to refresh your memory.

Example 1; Conditionals/Loops & Custom Functions Review

Let's say you have graded your students' final exams. You have the scores in the computer, but do not want to manually type in all their grades. For now, just assume the course is Pass/Fail, so you just need two options. Say anyone with at least a 70% is a Pass. Let's write an if-else statement to do this. Recall the syntax for this is ...

```
if (condition){
expression
} else {
expression
}
```

```
score = 80
if(score >= 70){
  Result = "Pass"
} else {
  Result = "Fail"
}
Result
```

```
## [1] "Pass"
```

```
score = 55
if(score >= 70){
  Result = "Pass"
} else {
  Result = "Fail"
}
Result
```

```
## [1] "Fail"
```

Now what if the course uses letter grades? recall the `else if` syntax.

```
if (condition){
expression
} else if (condition) {
expression
} else {
expression
}
```

You may use as many `else if` as you need. It must though come *after* the initial `if` and *before* the final `else`. For clarification, we will use the following cutoffs for letter grades.

	A	B	C	D
Plus	99	87	77	67
Neutral	93	83	73	63
Minus	90	80	70	60

```
score = 92
if (score >= 99){
  Result = "A+"
} else if (score >= 93){
  Result = "A"
} else if (score >= 90){
  Result = "A-"
} else if (score >= 87){
  Result = "B+"
} else if (score >= 83){
  Result = "B"
} else if (score >= 80){
  Result = "B-"
} else if (score >= 77){
  Result = "C+"
} else if (score >= 73){
  Result = "C"
} else if (score >= 70){
  Result = "C-"
} else if (score >= 67){
  Result = "D+"
} else if (score >= 63){
  Result = "D"
} else if (score >= 60){
  Result = "D-"
} else {
  Result = "F"
}
Result
```

```
## [1] "A-"
```

Okay, now recall loops. Remember that a loop is used ...

1. When we want to do repetitive actions on observations.
2. To show how values change over iterations.
3. Because we want efficient coding.
4. For more complicated calculations.

Recall the syntax for a `for` loop is ...

```
for (index in expression){
expressions
}
```

Okay, let's look at using a loop to assign grades. Don't forget to include the `[i]` part after each of the objects you are saving the grades to.

```
#Setup Code; DO NOT DELETE
set.seed(143572)
score = sample(55:100, 10, replace = TRUE)
Result1 = rep(0,10)
Result2 = rep(0,10)
#Loop/Conditional
for(i in 1:10){
  if(score[i] >= 70){
    Result1[i] = "Pass"
  } else {
    Result1[i] = "Fail"
  }

  if (score[i] >= 99){
  Result2[i] = "A+"
} else if (score[i] >= 93){
  Result2[i] = "A"
} else if (score[i] >= 90){
  Result2[i] = "A-"
} else if (score[i] >= 87){
  Result2[i] = "B+"
} else if (score[i] >= 83){
  Result2[i] = "B"
} else if (score[i] >= 80){
  Result2[i] = "B-"
} else if (score[i] >= 77){
  Result2[i] = "C+"
} else if (score[i] >= 73){
  Result2[i] = "C"
} else if (score[i] >= 70){
  Result2[i] = "C-"
} else if (score[i] >= 67){
  Result2[i] = "D+"
} else if (score[i] >= 63){
  Result2[i] = "D"
} else if (score[i] >= 60){
  Result2[i] = "D-"
} else {
  Result2[i] = "F"
}
```

```
}
Result1
```

```
##  [1] "Pass" "Pass" "Pass" "Pass" "Pass" "Pass" "Pass" "Fail" "Pass" "Pass"
```

```
Result2
```

```
##  [1] "B+" "C+" "C-" "A"  "B+" "C"  "B+" "D"  "C-" "A"
```

Don't feel bad if you have to reread through this a few times to understand it. Loops and conditionals are one of the more difficult topics in this course. You are probably wondering where the functions review is. Well, since we are going to build off of that here, then we will just jump right into the next part of lecture material and mix in some review with it. Okay, now we can move onto the next portion of lecture content.

Custom Functions in R Part II and More Sampling

Today we are going to make some more custom functions, specifically continuing the example from last time. Now making a custom function is useful, knowing how to use it to help you out is equally as useful. In other words, a custom function does not really help you out unless you know how to use it properly. We will explore that today and also one more intricacy of functions. Then to wrap things up, we will look at more sampling features using what we learned last time and with the help of custom functions, make the sampling process much smoother.

Custom Functions in R Part II

Recall that to make a custom function, the general syntax is ...

```
Name = function(parm1 = ..., parm2 = ..., ...){
expressions here
return(What You Want To Calculate)
}
```

The expressions will be things like all the loops and conditionals that we were going over in the review example. Now the nice thing about custom functions is that they are super flexible. You can manipulate them anyway you wish and with ease. As I mentioned above, there is one more intricacy of functions. Those are the default arguments. Before we get into that, what is an argument? It is simply what you put into the function. For example, when you use the `sample()` function, you type in three things, the range of numbers, the sample size, and the replacement. These three things are each arguments. A default argument is one that is pre-programmed into a function to be used if you do not type it in with a specific value. If you go to the `sample()` functions help page, you will notice that there is a fourth argument, `prob = ...`. This is for doing sampling with unequal probabilities. By you not using it in the functions, the computer uses the default `prob = ...` for the function of all the probabilities being equal. To set a default argument, simply do `parm = x`, where `x` is the default argument value. Okay, now let's jump into an example to illustrate this.

Example 2; Final Grades

You are a professor teaching two different courses. Both of your finals have been graded and you now have your students' final percentages calculated in your spreadsheet. You are almost done for the semester, all you have to do is input the final grades. However, your first course has 135 students, and your second has 279 students. Also, some of the students decided to take the course as Pass/Fail. This is further complicated with your spreadsheet being organized by student ID number. You decide to create a custom function to

speed up the process of assigning letter grades. You use the following cutoffs for letter grades as your default . . .

	A	B	C	D
Plus	99	87	77	67
Neutral	93	83	73	63
Minus	90	80	70	60

However, your second course you had to implement the following curve . . .

	A	B	C	D
Plus	95	82	71	60
Neutral	89	78	67	56
Minus	86	75	64	53

You do not normally have to curve your courses, but this time you had no other choice. You still use your default 70% Pass/Fail cutoff for the first course, but lower it to 64% for the second. Using the data below, create a custom function (with defaults) that will assign grades for you.

```
#DO NOT DELETE THIS CODE!!
set.seed(143572)
grades1 = round(runif(135, min = 65, max = 100), 2)
pf = c("Letter", "Pass/Fail")
result1 = sample(pf, 135, replace = TRUE, prob = c(0.95,0.05))
Course1 = data.frame(grades1,result1)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.6     v purrr   0.3.4
## v tibble  3.1.7     v dplyr   1.0.9
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
Course1 = Course1 %>% rename(Percentage = "grades1") %>% rename(Grade_Type = "result1")
set.seed(314439)
grades2 = round(runif(279, min = 58, max = 95), 2)
result2 = sample(pf, 279, replace = TRUE, prob = c(0.95,0.05))
Course2 = data.frame(grades2,result2)
Course2 = Course2 %>% rename(Percentage = "grades2") %>% rename(Grade_Type = "result2")
```

Okay, now let's create a custom function that fits all the criteria above. You do **NOT** have to do something this complicated for your project, this is just to show you what all you can do.

```
GradeAssign = function(score, type, A_p = 99, A = 93, A_m = 90, B_p = 87, B = 83, B_m = 80,
                       C_p = 77, C = 73, C_m = 70, D_p = 67, D = 63, D_m = 60){
  x = length(score)
  result = rep(0,x)
  for(i in 1:x){
    if(type[i] == "Pass/Fail"){
      if(score[i] >= C_m){
        result[i] = "Pass"
      } else {
        result[i] = "Fail"
      }
    } else {
      if (score[i] >= A_p){
        result[i] = "A+"
      } else if (score[i] >= A){
        result[i] = "A"
      } else if (score[i] >= A_m){
        result[i] = "A-"
      } else if (score[i] >= B_p){
        result[i] = "B+"
      } else if (score[i] >= B){
        result[i] = "B"
      } else if (score[i] >= B_m){
        result[i] = "B-"
      } else if (score[i] >= C_p){
        result[i] = "C+"
      } else if (score[i] >= C){
        result[i] = "C"
      } else if (score[i] >= C_m){
        result[i] = "C-"
      } else if (score[i] >= D_p){
        result[i] = "D+"
      } else if (score[i] >= D){
        result[i] = "D"
      } else if (score[i] >= D_m){
        result[i] = "D-"
      } else {
        result[i] = "F"
      }
    }
  }
  return(result)
}
```

That was a lot of work and the function looks really complicated. However, let's see what happens once we use it. Let's start with `Course1`.

```
Course1 = Course1 %>% mutate(Final_Grade = GradeAssign(Percentage,Grade_Type))
head(Course1,20)
```

```
##    Percentage Grade_Type Final_Grade
## 1       68.99     Letter          D+
## 2       99.40     Letter          A+
## 3       80.99     Letter          B-
## 4       75.06     Letter           C
## 5       65.52     Letter           D
## 6       85.64     Letter           B
## 7       75.21     Letter           C
## 8       95.38     Letter           A
## 9       93.39     Letter           A
## 10      69.02     Letter          D+
## 11      92.36     Letter          A-
## 12      81.77     Letter          B-
## 13      70.75     Letter          C-
## 14      82.27     Letter          B-
## 15      68.30  Pass/Fail        Fail
## 16      98.78     Letter           A
## 17      70.04     Letter          C-
## 18      91.83     Letter          A-
## 19      90.30     Letter          A-
## 20      73.82     Letter           C
```

Look at that, you just assigned your grades in no time. Make sure you do a cursory glance at the first twenty or so observations to check that your code is running correctly. Do not worry if you do not catch a mistake in your code because your students will. Now let's do `Course2`. Remember that this course had a curve, so you will have to manually type in your new grade cutoffs. Yes you have to do a bit of typing, but it is easier than changing the cutoffs manually in your code.

```
Course2 = Course2 %>% mutate(Final_Grade = GradeAssign(Percentage,Grade_Type,A_p = 95,
                                                       A = 89, A_m = 86, B_p = 82,
                                                       B = 78, B_m = 75, C_p = 71,
                                                       C = 67, C_m = 64, D_p = 60,
                                                       D = 56, D_m = 53))
head(Course2,20)
```

```
##    Percentage Grade_Type Final_Grade
## 1       77.25     Letter          B-
## 2       90.71     Letter           A
## 3       81.07     Letter           B
## 4       62.69     Letter          D+
## 5       82.00     Letter          B+
## 6       72.26     Letter          C+
## 7       72.00     Letter          C+
## 8       76.46     Letter          B-
## 9       61.03     Letter          D+
## 10      71.16     Letter          C+
## 11      90.89     Letter           A
## 12      75.99     Letter          B-
## 13      85.69     Letter          B+
```

```
## 14      91.94     Letter           A
## 15      87.56     Letter          A-
## 16      91.38     Letter           A
## 17      73.02     Letter          C+
## 18      71.25     Letter          C+
## 19      88.95     Letter          A-
## 20      60.67     Letter          D+
```

Ta da!! A bit more work, but grading is still done *much* faster. That is the whole idea behind custom functions.

More Sampling

Recall last time how we learned different ways of randomly generating numbers. What will be most useful for us is the random integer generation. We can use this to help us generate random samples from our data set. What we do is ...

1. Generate random numbers
2. Filter observations such that the row indexes match the randomly generated numbers

This is pretty simple to do in `RStudio`. Remember, you can use this method as an alternative to the one you learned in our notes on experimental design. You may use the method you prefer.

Example 3; Sampling Using Indexes

Let's say your department head wants you to randomly sample about ten students from each of your courses for department interviews. There is no selection criteria, it can be any student in your course. We will now select ten students from both courses using this new method. First, generate **TWO** different lists of ten random numbers.

```
set.seed(143572)
sample1 = sample(1:135, 10, replace = FALSE)
set.seed(314439)
sample2 = sample(1:279, 10, replace = FALSE)
sample1
```

```
##  [1]  52 135  98  22   9  81  44  20 133  59
```

```
sample2
```

```
##  [1]  83 111  10 173 229 245 116 205 136 252
```

Now let's look at the students you are sending to your department for interviewing. We need to create the subset of just the students selected. You can do that using the following code `data[sample,]`. Do **NOT** forget the `,` after the sample name.

```
S1 = Course1[sample1,]
S2 = Course2[sample2,]
S1
```

```
##     Percentage Grade_Type Final_Grade
## 52       66.53     Letter           D
## 135      82.27     Letter          B-
## 98       71.84     Letter          C-
## 22       80.02     Letter          B-
## 9        93.39     Letter           A
## 81       91.74     Letter          A-
## 44       71.95     Letter          C-
## 20       73.82     Letter           C
## 133      97.64     Letter           A
## 59       75.99     Letter           C
```

```
S2
```

```
##     Percentage Grade_Type Final_Grade
## 83       58.83     Letter           D
## 111      60.39     Letter          D+
## 10       71.16     Letter          C+
## 173      59.67     Letter           D
## 229      77.45     Letter          B-
## 245      77.67     Letter          B-
## 116      85.16     Letter          B+
## 205      81.76     Letter           B
## 136      67.06     Letter           C
## 252      63.18     Letter          D+
```

Uh oh. Looks like by pure chance you are not sending many students with high grades to be interviewed. You are worried that you will get a lot of negative reviews from students, so you "conveniently ensure" the selection process only selects those who pass. What your department head does not know will not hurt them. There are multiple ways to do it, so here is just one.

```
Course1a = Course1 %>% filter(Percentage >= 70)
Course2a = Course2 %>% filter(Percentage >= 64)
sample1a = sample(1:length(Course1a$Percentage), 10, replace = FALSE)
sample2a = sample(1:length(Course2a$Percentage), 10, replace = FALSE)
S1a = Course1a[sample1a,]
S2a = Course2a[sample2a,]
S1a
```

```
##     Percentage Grade_Type Final_Grade
## 117      82.27     Letter          B-
## 38       96.39     Letter           A
## 43       72.06     Letter          C-
## 60       86.95     Letter           B
## 41       87.02  Pass/Fail        Pass
## 30       81.52     Letter          B-
## 19       89.84     Letter          B+
## 105      85.15     Letter           B
## 83       73.61     Letter           C
## 4        85.64     Letter           B
```

```
S2a
```

```
##     Percentage Grade_Type Final_Grade
## 124      89.54     Letter           A
## 96       84.82     Letter          B+
## 141      76.15     Letter          B-
## 215      90.76     Letter           A
## 47       64.68     Letter          C-
## 240      75.82     Letter          B-
## 86       73.33     Letter          C+
## 43       87.75     Letter          A-
## 147      79.13  Pass/Fail        Pass
## 126      77.92     Letter          B-
```

Hopefully this helps you get an understanding of how to use custom functions and uses for random number generation. We covered a lot today, so do not feel bad if you have to reread through the notes a few times. Plus we have lab too. This is the second to last coding heavy topic we have in this course. The last will be Simple Linear Regression (SLR), which is covered in the last lecture. The rest will be more statistical so the coding (on your part) will not be as intense. Next time we will introduce the statistical topic of distributions (types, properties, etc.).

End of Lecture 9 Notes