Lecture 8

Paul Holaway, Abhi Thanvi

July 5th, 2022

Lecture 7 Review

Before we move on to the new material, we will do a quick review of Lecture 7 content. Last time we learned about independence, conditional probability, and the multiplication rule. Recall that when two events are independent if they are not affected by the occurrence of another event or events. The most classic example of this is flipping a coin. Whatever you flipped the first n times, the probability of the next flip being heads or tails is not going to change.
Then there was conditional probability. Conditional probability is the probability that an event A occurs given that an event B has already occurred. The formula to calculate this is $P(A|B) = \frac{P(A \cap B)}{P(B)}$. Conditional probability is important to know when your events are *not* independent.
Then there was the multiplication rule. There were two cases, one when the events are independent and another when they are not. The formula for the two cases are…

1. $P(A \cap B) = P(A) * P(B)$ (Independent)
2. $P(A \cap B) = P(A|B)P(B)$ (Dependent)

Okay, this was a bit dense, so let's go through a quick example.

Example 1; Probability Review

You have been hired by the Norfolk Southern railroad to do some probability calculations for them. Pretend that this data set is just for the Norfolk Southern Railroad dispatchers (it's really all railroads in the US).

```
library(readxl)
Dispatch <- read_excel("~/Desktop/DPISu22/Data Sets/Dispatchers_Background_Data.xls")
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()
```

Norfolk Southern has to have their employees interviewed by an independent third party, who will then record their responses and send them to the National Railroad Transportation Safety Board. Norfolk Southern is in hot water. If the NRTSB interviews a dispatcher who is severely mentally drained, they will get a full audit done on them. Obviously Norfolk Southern does not want this. However, some of the employees suffer from sleeping disorders. If one of them goes before the NRTSB and is severely mentally drained, then the NRTSB will help them get treatment instead of auditing Norfolk Southern.

Question 1: First, Norfolk Southern wants to know what the probability is that a dispatcher is severely mentally drained given they have a sleep disorder.
Hint: 1 = Has Disorder and 4 = Severely Mentally Drained

```
PSD = Dispatch %>% filter(Diagnosed_Sleep_disorder == 1)
PSM = Dispatch %>% filter(Mentally_Drained == 4 & Diagnosed_Sleep_disorder == 1)
length(PSM$ID)/length(PSD$ID)
```

```
## [1] 0.2195122
```

Question 2: Second, Norfolk Southern wants to know what the probability is that a dispatcher is severely mentally drained and does **NOT** have a sleep disorder. We know how to calculate $P(MD \cap SD)$, but here we want $P(MD \cap SD^C)$. How can we do this? Easy, it is just $1 - P(MD \cap SD)$.

```
DND = Dispatch %>% filter(Mentally_Drained == 4)
length(DND$ID)/length(Dispatch$ID)
```

```
## [1] 0.1595506
```

Question 3: What is the probability that a dispatcher selected is severely mentally drained?

```
PMD = Dispatch %>% filter(Mentally_Drained == 4)
length(PMD$ID)/length(Dispatch$ID)
```

```
## [1] 0.1595506
```

Okay, now we can move onto the next portion of lecture content.

Random Numbers, Loops/Conditionals, and Custom Functions in `R` Part I

Today we are going to learn three more concepts from the data science side of statistics. We will be just going into the basics of each because they can get complicated quickly. These concepts can also get a bit dense so today's lecture will be shorter than usual to ensure that students understand the concepts well enough to use them.

Random Numbers

A random number is just a number picked randomly from a predefined range of numbers. Generating random numbers has two main uses in statistics. Running computer simulations in theoretical statistics and sampling selection. The former is beyond the scope of this course so we will be looking at the latter next time. For now, let's just learn how to generate random numbers using `RStudio`.

Example 2; Generating Random Numbers

Generating random numbers in `R` is possible using one of two functions. The first is the `runif()` function. The second you have already used before, it is the `sample()` function. Each one has its own use in random number generation. `runif()` is used when you want to generate random numbers that are not whole numbers (integers). `sample()` is used when you want to generate random numbers that are whole numbers (integers). Which function you use depends on what you are trying to accomplish. The syntax for the two functions are below… - `runif(n,min=x,max=y)` - `sample(x:y, n, replace = ...)` - `n` = The number of random numbers you want generated. - `x` = Minimum value of the random number range. - `y` = Maximum value of the random number range.

```
set.seed(143572) #Do NOT Delete this line.
runif(5,min = 0, max = 100)
```

```
## [1] 11.407271 98.292774 45.684444 28.744634  1.490916
```

```
sample(1:100, 5, replace = TRUE)
```

```
## [1] 41 52 48 98 22
```

```
sample(1:100, 5, replace = FALSE)
```

```
## [1] 33  9 81 44 20
```

In general `runif()` is best for theoretical simulations, so we will mostly be using `sample()`. You may be wondering right now how this relates to random sampling. Well, let's go to the `Dispatch` data set. Let's say that is the population and we want to take a random sample from it. You can use the previous code we used in experimental design (#Lecture4), however, here is an alternative method.
Notice how we have 445 observations in the data and we want to take a random sample of 100 of them. Also notice how when you view the data, there are white numbers in a light blue box. Those numbers are called index numbers. They are used if you need to identify a specific observation (row) in the data set that is not at the very beginning or end (we can use `head()` and `tail()` then). If you want to look at a specific row in a data set, you can do so using the index number. We will look into this next time. For today, we will just get the idea of how we can sample using the index numbers.
Let's say you want to take a random sample without replacement, but you think the sampling code for experimental design is too complicated. We can randomly select rows by first randomly selecting index numbers, first without replacement.

```
set.seed(143572)
sample(1:445, 100, replace = FALSE)
```

```
##   [1] 308 418 244 407 242 169  52 432 354  22 161   9  81  44 276 226  59 275
##  [19] 124  73 423 133 431  57 152 370  12 361 342 377 253 217 227 224  99  15
##  [37] 394 430 214 405 144 146 266 164 236 310 278 268  38 230 104 395 141  21
##  [55] 176 147 381 388 368 390 136 241 166 119 185 284   4   2 416 296 199 365
##  [73] 422 429 139 386 360 184 249   1  66 231 205 114  69 294 198 322 306 126
##  [91] 304  72 192 154  58 402 162 180 209 346
```

Or we can do it with replacement.

```r
set.seed(143572)
sample(1:445, 100, replace = TRUE)
```

```
##   [1] 308 418 244 407 242 169  52 432 354  22 161   9  81  44 276 226  59 275
##  [19] 124  73 423 133 276  57 152 370  12 361 342 377 253 217 227 224  99  15
##  [37] 394 226 214 405 144 146 266 164 420 236 310 278 268  38 230 104 104 141
##  [55]  21 430 176 408 430 147 381 388 368 147 136 241 166 119 185 284   4 404
##  [73]   2 377 296 199 365  57  59 139 147 406 360 184 428 249   1  66 231 205
##  [91] 114  69 404 294 198 322 429 306 126 304
```

We will learn next time how to implement this into a sampling procedure. For the record, both the method covered in experimental design and today are valid and used by numerous researchers. You may use whichever one you prefer.

Loops/Conditionals

We will first go over what a conditional is. A conditional is an logical expression for the computer. It normally consists of, "If this happens do this, but if that happens do that." The most useful conditional is the if else statement. This can contain two or more conditions that tell the computer what to do. Let's do a quick example.

Example 3; If-Else

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. The syntax is…

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

```r
score = 85
if(score >= 70){
  Result = "Pass"
} else {
  Result = "Fail"
}
Result
```

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

```r
score = 65
if(score >= 70){
  Result = "Pass"
} else {
  Result = "Fail"
}
Result
```

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

Now what if the course uses letter grades? That would be much more than just two options. How do we go from there? We can add in a part to our code that allows for more than one condition. The `else if` command.

```
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`. Let's see how this works in terms of making final grades. 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

```r
score = 85
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] "B"
```

```r
score = 65
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] "D"
```

Okay, now things are working, so let's go into loops. Notice how we had to retype the code multiple times to accomplish one grade result. We do not want to do that because it will take a while and is tedious work (that can be done by underpaid graduate student). This is where loops come in handy. A loop is used for the following reasons…

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.

There are multiple types of loops, but we will just look at the `for` loop. The `for` loop is used for repeated actions for a predetermined number of times using either an index or counter. We will mostly be using if for indexing purposes. The syntax for a `for` loop is very simple…

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

Okay, now let's try this out. Let's say that these are the students grades in percentage, let's convert them to Pass/Fail and a letter grade.

```r
#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"
```

Pretty neat right? Notice how we have to have that `[i]` after each of the `score` and `Result`. This is because since we are running a loop using indexing, we need to make sure the end results are indexed too because we are doing multiple iterations.

Custom Functions in `R` Part I

So far you have just been working with pre-built functions that others have made. However, what happens if you need to do something that does not already have a function made. Well, you can repeat the code for the process multiple times, or you can create a custom function in `RStudio` to do it for you. Creating a custom function has many benefits to it. The main one is you can greatly reduce the length of your code, especially if you need to do something repetitive. If you find yourself copying and pasting the same code often for your project, it may be a good idea to write a custom function. The other benefit is it greatly expands your coding ability. Being able to write a custom function in a language shows you understand it well, which is something companies love. Today we will just be introducing custom functions and tomorrow we will be doing a deep dive into the creative world of custom functions.
To make a custom function in `RStudio`, you need the function `function()`, and the syntax to make a custom function is ...

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

We will come back to the letter grade example next time. Let's do a few basic examples before we end.

Example 4; Calculating a Value

Let's suppose you are working on your math homework and you have to calculate a bunch of future values of money. You have 25 problems to do and do not want to type them all into the calculator. To simplify this, let's make a custom function to do it for us. The formula for calculating the future value of money is $FV = PV * (1 + r)^n$. $r =$ Interest Rate, $n =$ Number of Periods. Let's calculate these values when the interest rate is 2.5% over a period of 12 months.

```
#Creation of Function
FV = function(PV, r, n){
  FV = PV * (1+r)^n #Formula
  FV = round(FV,2) #Rounding to Nearest Penny
  return(FV) #Returning the Future Value
}
#Setup Code; DO NOT DELETE
set.seed(143572)
Money = runif(25,min = 0, max = 100)
Money
```

```
##  [1] 11.407271 98.292774 45.684444 28.744634  1.490916 58.962655 29.163203
##  [8] 86.797011 81.126966 11.476803 78.157324 47.900530 16.419170 49.341592
## [15]  9.441875 96.514862 14.407118 76.651802 72.294422 25.188880 35.267272
## [22] 42.920179 70.957747 35.359180 56.181633
```

```
#Using the Custom Function
FV(Money,0.025,12)
```

```
##  [1]  15.34 132.19  61.44  38.66   2.01  79.30  39.22 116.73 109.11  15.44
## [11] 105.11  64.42  22.08  66.36  12.70 129.80  19.38 103.09  97.23  33.88
## [21]  47.43  57.72  95.43  47.55  75.56
```

See how easy it is to do your homework? Let's try another example.

Example 5; Quadratic Formula

Your friend is also working on their math homework, but their homework is on the quadratic formula. Your friend has 10 problems to do, but keeps screwing up typing it into their really expensive TI-84 calculator. You decide to be nice (since your homework is already done) and help them out by making a custom function in `RStudio`. Recall that the quadratic formula is . . .

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

```
#Creating the Function
QF = function(a, b, c){
  x = (-b + c(-1,1) * sqrt(b^2-4*a*c))/2*a #c(-1,1) is a way to code the +/- into R.
  return(x)
}
#Using the Function
QF(a=1,b=5,c=2)
```

```
## [1] -4.5615528 -0.4384472
```

```
QF(a=1,b=-6,c=-10)
```

```
## [1] -1.358899  7.358899
```

```
QF(a=-2,b=4,c=1)
```

```
## [1]  8.8989795 -0.8989795
```

Okay, now we get the basic idea, but we can definitely do more. Next time we will do a deeper dive into functions and get creative.

End of Lecture 8 Notes