Programming Assignment 2: Bonus Content

SOC Methods Camp

September 4th and 5th, 2019

Your homework this week is not meant to be exceptionally challenging, but everyone is coming into camp with different levels of programming experience. This bonus content is to help make the first homework a little easier by pointing you in the direction of commands you might need or the structure a piece of code should have. The task numbers below correspond with the tasks on your assignment.

It is completely up to you if you want to reference this sheet for help with some, all, or none of the tasks- choose your own adventure! And remember- if you're finding yourself agonizing over a problem for hours, *don't.* Shoot Liv or Katie an email, and if it's late just put aside the assignment, get some sleep, and come in tomorrow morning with what you have done.

Step one

Task one: Let's remember what a for loop needs:

```
#you need an empty vector for the for loop to fill
empty_vector <- c()

#you will tell R that you want it to do something to each individual i in a
#given number of instances
for(i in 1:length(NumerOfDataRowsOrColumns)){
  #And then inside the squiggly brackets you tell R what you want to do to
  #each i. This is where the "meat" of your for loop goes. At the end of the
  #"meat" section, you usually want to tell R to put some output into the ith
  #position of your empty vector
  empty_vector[i] <- MeatOutput
}
```

Let's also review what an ifelse statement needs:

```
#a simple if else test:
ifelse(ATest, WhatToDoIfTrue, WhatToDoIfFalse)

#when there's more than two options:
if(ATest){
  ThenDoThisThing
} else (AnotherTest){
  ThenDoThisOtherThing
} else (YetAnotherTest){
  ThenDoThisThirdThing
} else {
  WhatToDoWhenAllTestsAreFalse
}
```

In this case, we want our for loop to apply the ifelse statement to things. So, we want the ifelse statement to be in the meat of our for loop. To make things easier for yourself, start by writing the ifelse statement you'd use to change just one entry, then fold that into a for loop. (Hint: the ifelse statement will have multiple steps.)

Task two: You will need the dplyr *filter* command here. Within the command, you will need to use *&* to apply multiple conditions to your filter.

Task three: Just read the annotated code, then run it! Easy task.

Task four: For this one, let's review the structure of a function:

```
FunctionName <- function(Argument1, Argument2, Argument3){
  -insert thing you want done involving Argument1, Argument2, and Argument3-
}

#Example:
FindMean <- function(VectorOfValues){
  sum(VectorOfValues) / length(VectorOfValues)
}
```

Task five: If you need help with the inputs for the *sapply* command, use RMarkdown's "Help" tab in the lower righthand pane.

Step two

Task one: You will need the filter command here. Use it to filter all items where age *does not equal* "Unknown Date."

Task two: Here you will need the *group_by*, *select*, and *summarise* commands. Remember that you can put multiple grouping variables into one *group_by* command. Order matters!

Step three

Task one: Remember the basic ggplot structure:

```
plotname <- ggplot(YourDataHere, aes(x = YourXHere)) +
  geom_GEOMTYPE
```

You can use the *ylim* subcommand in ggplot to control the y axis.

Also remember that in the last assignment you were able to display two plots side by side using a command from the gridExtra package!

Task two:

The wording of this question makes it sound like you need a for loop in the function. You don't! Instead, try subsetting your stateyear data (as input into the ggplot command) using your states of interest vector. Then, think about how you can set the *ylim* argument to select the appropriate maximum y axis.

As an additional hint, don't forget to create an empty list for your function to fill before you write the function.

Step four

Task one: The function you want here is *sapply*. Don't forget you need to supply the arguments for your function inside the *sapply* command!

Task two: Again, you'll want to use *sapply* here.

Task three: Here is a scaffolding for what your answer should look like and do:

```
FunctionName <- function(arg1, arg2, arg3, arg4, arg5){
  if(numdraws>nrow(data)){
    round as instructed
    do the rest of the original func
  } else {
    original func
  }
  return: Something
}
```

Task four: All you need to do here is use different inputs for the numdraw argument.