Applied Statistical Programming - Spring 2022

Jordan Duffin Wong

Problem Set 3

Due Wednesday, March 2, 10:00 AM (Before Class)

Instructions

1. The following questions should each be answered within an R script. Be sure to provide many comments in the script to facilitate grading. Undocumented code will not be graded.
2. Work on git. Fork the repository found at https://github.com/johnsontr/AppliedStatisticalProgramming2022 and add your code for Problem Set 3, committing and pushing frequently. Use meaningful commit messages because these will affect your grade.
3. You may work in teams, but each student should develop their own Rmarkdown file. To be clear, there should be no copy and paste. Each keystroke in the assignment should be your own.
4. For students new to programming, this may take a while. Get started.

Let's Make a Deal[1]

In the game show "Let's Make a Deal' ', the candidate gets to choose one of three closed doors, and receives the prize behind the door they choose. Behind one door is a new car; behind the other two doors are goats. After the contestant selects one of the 3 doors, the host opens one of the other two doors, and reveals a goat. Now, the candidate has the option of either sticking with the door they originally selected, or switching to the only other door that is still closed. What should the candidate do, and why? What are the probabilities of winning the car if they stay versus if they switch? This question is known as the Monty Hall Problem.

Your tasks

For this problem set, you will not solve the Monty Hall Problem, but you will have to code a slightly simplified version of the "Let's Make a Deal" game. More specifically, you will set up a new class, which contains information regarding the door a player chooses, and a method that simulates a modified version of the game. You will have to do this using the S3 class system. Here are the specific instructions:

1. Define a new class: `door`. Objects of this class simply take on one numeric value: 1, 2, or 3 – indicating which door a candidate chooses.
2. Create a method for `door` objects that is called `PlayGame`. This method is supposed to do the following:
   - take the numeric value that is stored in the `door` object,
   - draw a random number between 1 and 3 that presents the door behind which the car is hidden,

[1] https://en.wikipedia.org/wiki/Let's_Make_a_Deal

   - compare the two numbers, and print a message congratulating a winning candidate that chose the correct door, or expressing sympathies for a losing candidate that chose the wrong door.

3. Write:

   - a construction function that allows the user to create a `door` object,
   - and a validation function that checks whether the value stored in `door` is actually an integer

```r
# Code Let's Make a Deal! ##### 1 ### Creating a new class: `door`, which takes
# on a value in c(1:3) depending on the `choice` of the player.
door <- list(choice = c(1:3))
class(door) <- "door"

### 2 ### Creating the `PlayGame` generic, preparing it to take `door` as an
### input
PlayGame <- function(door) {
    UseMethod("PlayGame")
}

# And the function itself
PlayGame.Door <- function(choice) {
    # Simulating which door conceals the car It can be a number between 1 and 3
    # with equal probability
    cardoor <- as.numeric(sample(c(1:3), size = 1))

    if (player_choice == cardoor) {
        # If the choice matches correctly, the player wins
        return("Winner winner, chicken dinner! It's a match!")
    } else {
        # It's a coarse way to express sympathy, I guess
        return("What a fuckin' loser! No match!")
    }
}

### 3 ### The construction function for the new door
new_door <- function(choice) {
    output <- list(door = choice)
    class(output) <- "door"
    return(output)
}

# And a validator, ensuring that `choice` is actually a number 1, 2, or 3 we
# also have to define `not in`
`%!in%` <- Negate(`%in%`)

validate_door <- function(choice) {
    # Testing that the choice is actually a number
    if (!is.numeric(choice)) {
        stop("Gotta choose a number, buddy!")
    }

    # Testing if that number is a valid choice
    if (choice %!in% c(1:3)) {
        stop("Gotta choose something between 1 and 3, buddy!")
    }
}

# Thus, re-constructing the game with the constructor and validator
PlayGame.Door <- function(choice) {
    # The validator
    validate_door(choice)

    # and the constructor
    player_choice <- new_door(choice)

    # Simulating which door conceals the car It can be a number between 1 and 3
    # with equal probability
    cardoor <- as.numeric(sample(c(1:3), size = 1))

    if (player_choice == cardoor) {
        # If the choice matches correctly, the player wins
        return("Winner winner, chicken dinner! It's a match!")
    } else {
        # It's a coarse way to express sympathy, I guess
        return("What a fuckin' loser! No match!")
    }
}


# Testing it!  For some reason, I can't get validator testing to work when
# testing the knit process. I'm pretty sure it's because they necessarily use
# stop(), but I haven't devised a workaround.  I super promise it works,
# though.
set.seed(666)
# PlayGame.Door(choice = 'The letter 'M'') PlayGame.Door(choice = 0)
PlayGame.Door(choice = 1)
```

```
## [1] "What a fuckin’ loser! No match!"
```