Activity 4/5 - R Programming - Loops

Gioconda Prada

...

Activity 4

For Activity 4, you just need to finish Question 1 (a), 3, 4. Question 5 is for Extra Credits.

---

1. Translating `for` loops into their underlying sequence of commands and vice versa is good practice for understanding the use of loops as shortcuts.

a. The following chunk defines `x` to be a vector of 4 elements, then uses a line of code to define the 4th, the 5th, etc., out to the 100th element. Write out a `for` loop to accomplish this instead. Once you do, run `tail(x)` to print to the screen the last 6 elements of `x`. Sanity check. `x[12:15]` should be -1.112500 -1.102549 -1.092708 -1.084811.

```
x <- c(7,10,2,3)
#Tedious way of defining elements 4-100
x[4] <- (1/2)*x[2] - 1
x[5] <- (1/3)*x[3] - 1
x[6] <- (1/4)*x[4] - 1
x[7] <- (1/5)*x[5] - 1
x[8] <- (1/6)*x[6] - 1
x[9] <- (1/7)*x[7] - 1
#...
x[100] <- (1/98)*x[98] - 1

x <- c(7,10,2,3)


#Write a for loop to do this instead

x<-c(7,10,2)
for(position in 4:100 ){
  x[position]<- (1/(position-2)*x[position-2]-1)
}
tail(x)
## [1] -1.010872 -1.010755 -1.010641 -1.010529 -1.010419 -1.010312
```

---

3. Load in the `TIPS` dataframe from `regclass`.

a. Define a vector called `average.tip.by.day` to intially be empty. Then, write a `for` loop to populate the elements of `average.tip.by.day` with the average tip percentage for each day of the week in the `Weekday` column.

b. Define a vector called `median.bill.by.party` to intially be empty. Then, write a `for` loop to populate the elements of `median.bill.by.party` with the median bill amount for each unique value in the `PartySize` column.

```
#install.packages("regclass")
library(regclass)
## Loading required package: bestglm
## Loading required package: leaps
## Loading required package: VGAM
## Loading required package: stats4
## Loading required package: splines
## Loading required package: rpart
## Loading required package: randomForest
## randomForest 4.7-1.1
## Type rfNews() to see new features/changes/bug fixes.
## Important regclass change from 1.3:
## All functions that had a . in the name now have an _
## all.correlations -> all_correlations, cor.demo -> cor_demo, etc.
data(TIPS)

#a
average.tip.by.day <- c()
for ( day in levels(TIPS$Weekday) ) {
  SUB <- subset(TIPS,Weekday==day)
  average.tip.by.day[day] <- mean(SUB$Tip)
}
print(average.tip.by.day)
##   Friday Saturday   Sunday Thursday
## 2.734737 2.993103 3.255132 2.771452

#b
median.bill.by.party<- c()
for(party in unique(TIPS$PartySize) ) {
  SUB <- subset(TIPS,PartySize==party)
  median.bill.by.party[party] <- median(SUB$Bill)
}
 median.bill.by.party
## [1]  7.915 15.370 20.365 25.890 29.850 32.050
```

Walkthrough for part a:

a) Run the aggregate command so you know what numbers to expect.

b) Since you're going to be placing elements into `average.tip` one at a time, you need to make sure it's "initialized" first.

c) Think about what you're looping over. In this case it is "days of the week", or even more specifically "levels of `Weekday`", so `levels(TIPS$Weekday)` will be the vector of values to loop over.

d) Let the looping variable be called `day`, and try out `day <- "Saturday"`. Develop code that finds the average tip percentage on Saturday (instead of hard-coding Saturday anywhere, let it be represented by `day`). One way: take a subset or rows that correspond to Saturday, then find the average of the `TipPercentage` column for that subset.

e) Set up the `in` for the loop: `for ( in ) {}`. You've named the looping variable and know the vector of values to loop over.

f) Generalize your line from (d) so that it'll work inside the `for` loop.

---

4. The command `sample(x,size=n,replace=TRUE)` picks n elements from the vector `x` at random with replacement (the same element can be picked twice). Imagine that over a customer's lifetime they make either 1, 2, …, or 10 purchases (for all intents and purposes the value can be considered to be picked at random). Each purchase results in the customer spending either 5, 5.5, 6, 6.5, …, or 20 dollars (for all intents and purposes the value can be considered to be picked at random).

Using a `for` loop, generate the lifetime values (total spent) of 50000 customers and put them in a vector called `lifetimevalue`.

Include a histogram of the values with `hist(lifetimevalue,breaks=seq(from=0,to=max(lifetimevalue)+5,by=5))` (this makes bars of width 5 starting at 0), the output of `summary`, and the overall average lifetime value in your writeup. You'll want to follow best practices for writing `for` loops to get this right!

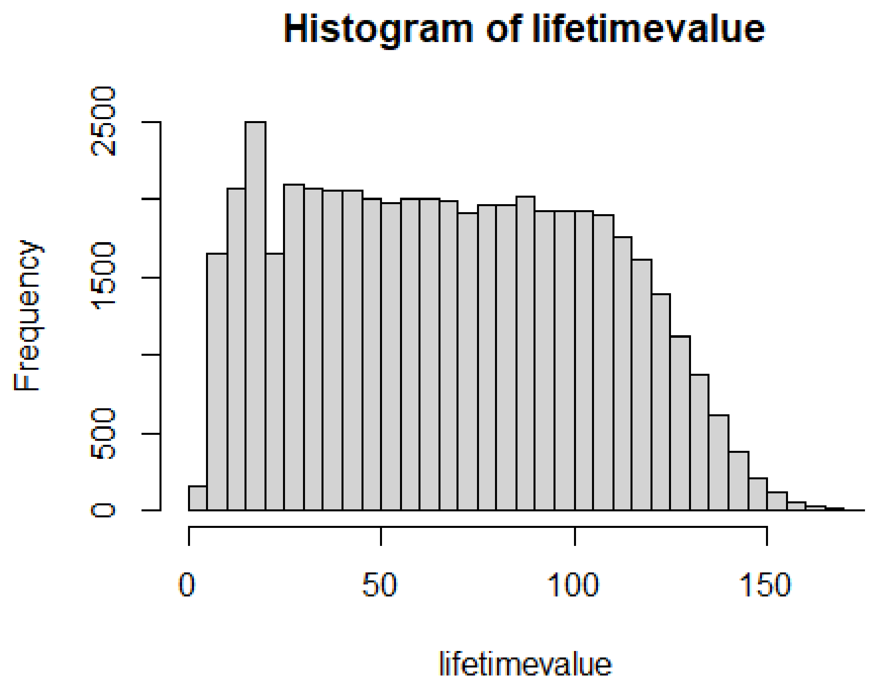
Hints:

- Get your code working for the first customer before writing the loop!
- First, create a variable called `purchases` which contains the (randomly picked) number of purchases that this customer makes during his or her lifetime (i.e., left-arrow into `purchases` the output of the appropriate `sample` command).
- Then, create a vector `amounts` by left-arrowing the output of the appropriate `sample` command. This vector will store the amounts of money spent on each of this customers purchases. For instance, if `purchases` happened to equal 6, then `amounts` will be a vector of length 6 containing 6 randomly chosen monetary values. If `purchases` happened to be 2, then this will be a vector of length 2 containing 2 randomly chosen monetary values. The sum of the values in `amounts` gives this customer's lifetime value.

- If you need help setting up the `sample` command to generate `amounts`, go back to the description of `sample`. The first argument is a vector of values to select from (use `seq` to specify this). The second argument is the number of values to pick. This number is "random", but is stored in `purchases`!
- Once you are confident in your code, initialize a vector `lifetimevalue` to be an empty vector, and have a `for` loop fill in its values. You're looping over the integers 1, 2, 3, (The i-th element of `lifetimevalue` will be the lifetime value of customer `i` and will equal the sum of the numbers in the `amounts` vector created).

```r
i<- 1

lifetimevalue<-c()
for(i in 1:50000){
purchases<- sample(1:10, size=1)
amount<-sample(seq(5,20, by=0.5), size=purchases, replace=TRUE)
lifetimevalue[i]<- sum(amount)
}
hist(lifetimevalue,breaks=seq(from=0,to=max(lifetimevalue)+5,by=5))
```



Extra Credits - Question 5

5. There is a line of 500000 switches, each initially switched to "on". One by one, 250000 people flip some of the switches.

- Person 1 flips off switch 1.
- Person 2 flips off switches 4, 6, 8, …, 499998, 500000
- Person 3 flips off switches 6, 9, 12, …, 499996, 499999 (any switch already off stays off)
- Person 4 flips off switches 8, 12, 16, …., 499996, 500000 (any switch already off stays off)
- etc.
- In other words, after person 1, person `i` flips off switch numbers `seq(from=2*i,to=500000,by=i)`

Determine which switches remain on after all people have completed their flips. Hint: work with a vector called `switches`. We'll let element `i` represent the status of the `i`-th switch: 1 for on and 0 for off. Define `switches` to be 500000 1s. Left-arrow the first element to be 0 (to represent the first person turning the first switch to off). Use a `for` loop to "turn off" the switches flipped off by person 2, 3, …, 250000. Sanity checks: `switches[5979:5989]` gives 0 0 1 0 0 0 0 0 1 0 0 and `sum(switches[32943:39999])` will give 672.

a) What is the total number of switches that remain on?

b) What is the largest switch number (1-500000) that remains on?

c) Print to the screen the first 20 or so numerical values of the switch numbers that are on. This sequence of numbers is a very famous one! What is it?

```
#switches[5979:5989]
#[1] 0 0 1 0 0 0 0 0 1 0 0
#sum(switches[32943:39999])
#672

#a)  how many are still on?



#b)  last switch that is still on (largest switch number)
```

```
#c)  What does the sequence look like
```

Walkthrough if you need it:

Step 1: We'll use a vector called `switches` to represent the status of each switch (1 = on, 0 = off). Initialize this vector to be `rep(1,5e5)` ("all on"), then change `switches[1]` to 0 to represent the action of person 1.

Step 2: Write a line of code that "left-arrows" elements 4, 6, 8, … 499998, 500000 of `switches` to 0 to represent what person 2 does to the switches.

Step 3: Write a line of code that "left-arrows" elements 6, 9, 12, … 499996, 499999 of `switches` to 0 to represent what person 3 does to the switches.

Step 4: Verify that first 25 elements of `switches` is now `0 1 1 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 1`

Step 5: Person 15843 will turn switches 31686 47529 63372 79215 95058 to "off". Left-arrow `i` to be 15843, then verify that `seq(from=2*i,to=100000,by=i)` indeed produces this sequence of numbers.

Step 6: Synthesize what you have done in Steps 1 to 5 - initialize `switches` to be a vector of 500000 1s (then change the first element to 0), then write a `for` loop to simulate the flips of persons 2, 3, …, 250000

Activity 5

For Activity 5, you need to finish Question 1 (a).

---

1. Let's write `while` loops to practice using the `while` loops.

a. Suppose you went to Vegas and you will bet until you are broke. You start with $100 and bet $1 each time for the slot machine, and you want to keep track of the money you had after each bet. The probability of winning is 1/216 for getting three jokers. Once you win, you earn $30. Write a `while` loop to simulate this. Also, set the seed number equal to 9750 in order to reproduce your simulation result. Show the money after first 6 bets on the screen after this is done.

```
money <- c()
money[1] <- 100
number.of.bets <- 0
set.seed(9750)

while ( min(money) > 0) {
  result <- sample( c(30,-1),size=1,prob=c(1/216,215/216))
  money <- c(money,tail(money,1) + result)
  number.of.bets <- number.of.bets + 1
}

head(money, 6)
## [1] 100  99  98  97  96  95
```
