Activity 4 - R Programming - Loops

Aryan Khandelwal

3/8/2022

For Activity 4, you just need to finish Question 1, 2, 3. Question 4 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.
2. 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)  
#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)  
  
  
for(i in 4:100){  
 x[i]=1/(i-2)\*x[i-2] -1  
}  
tail(x)  
## [1] -1.010872 -1.010755 -1.010641 -1.010529 -1.010419 -1.010312  
x[12:15]  
## [1] -1.112500 -1.102549 -1.092708 -1.084811

1. The following for loop goes through the words cannon, jackson, benford, paws, and moxie and defines elements of y, a named vector to be the letter of the alphabet in the position equal to the number of characters of each word (cannon has 6 letters, and the 6th letter of the alphabet is f). Write out the 9 lines of code for which the for loop is serving as a shortcut.

#The for loop as given  
y <- c()  
for( position in c("cannon","jackson","benford") ) {  
 n <- nchar(position)  
 y[position] <- letters[n]  
}  
y  
## cannon jackson benford   
## "f" "g" "g"  
  
#Starting with y <- c() and ending with y (print out the contents to the screen), write out   
#the 15 lines of code that is going on behind the scenes when the for loop is run  
y <- c()  
#... put in the 9 lines of code  
position="cannon"  
n = nchar(position)  
y[position] = letters[n]  
position="jackson"  
n = nchar(position)  
y[position] = letters[n]  
position="benford"  
n = nchar(position)  
y[position] = letters[n]  
y  
## cannon jackson benford   
## "f" "g" "g"

1. Forecasts of the number of items purchased in the future are desired, and the model to be used is (schematically):

N[t] = round( 0.5\*N[t-1] + sqrt(N[t-2]) + log10(1+N[t-3]) )

In other words, the forecast purchases today is 1/2 the purchases observed yesterday plus the square root of the purchases the day before that plus the log10 of the one plus the number purchases before that (rounded to the nearest integer after the sum is taken).

Let the three most recently observed number of purchases be 6, 11, and 21 (a vector called N is created in the chunk below to store these). Using the forecasting equation, fill in elements 4-20 of N (these will represent the forecasted purchases for the next 17 days). Print out the contents of N to the screen once you are finished. You’ll notice that eventually, the same number is forecast in-finitum (this is typical of forecasting models that don’t explicitly put in a trend).

N <- c(6, 11, 21)  
for(t in 4:20)  
{N[t] = round( 0.5\*N[t-1] + sqrt(N[t-2]) + log10(1+N[t-3]) )}  
N  
## [1] 6 11 21 15 13 12 11 10 9 9 9 8 8 8 8 8 8 8 8 8

Walkthrough:

1. Since you’re going to be placing elements into N one at a time, N needs to be “initialized” first. In this case, it is and the first three elements are given to you!
2. Think about what you’re looping over. You are creating a vector whose 4th element is given by the forecasting equation (which refers to the first 3 elements), whose 5th element is also given by the forecasting equation (which refers to the previous 3 elements), etc. The loop is over “positions of N”.

Call the looping variable t, let t <- 4, and get your code working so it defines the 4rd element of N correctly as round( 0.5\*N[3] + sqrt(N[2]) + log10(1+N[1])), but write your code using t instead of hard-coding the numbers!).

1. Figure out how to write the vector of values to loop over (this vector will go to the right of the in inside the for ( in ) {}).
2. Put it all together and generalize your line from (b) so that it’ll work inside the for loop.
3. Load in the TIPS dataframe from regclass.
4. 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.
5. 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  
## 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)  
names(TIPS)  
## [1] "TipPercentage" "Bill" "Tip" "Gender" "Smoker" "Weekday" "Time" "PartySize"  
levels(TIPS$Weekday)  
## [1] "Friday" "Saturday" "Sunday" "Thursday"  
  
day = "Friday"  
SUB= subset(TIPS, Weekday==day)  
average.tip.by.day = mean(SUB$TipPercentage)  
  
average.tip.by.day = c()  
for(day in levels(TIPS$Weekday)){  
 SUB = subset(TIPS, Weekday==day)  
average.tip.by.day[day] = mean(SUB$TipPercentage)  
}  
average.tip.by.day  
## Friday Saturday Sunday Thursday   
## 17.00000 15.31575 16.69145 16.12597  
class(TIPS$PartySize)  
## [1] "integer"  
median.bill.by.party = c()  
for(i in unique(TIPS$PartySize)){  
 SUB<-subset(TIPS, PartySize==i)  
 median.bill.by.party[i]<-median(SUB$Bill)  
}  
median.bill.by.party  
## [1] 7.915 15.370 20.365 25.890 29.850 32.050

Walkthrough for part a:

1. Run the aggregate command so you know what numbers to expect.
2. Since you’re going to be placing elements into average.tip one at a time, you need to make sure it’s “initialized” first.
3. 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.
4. 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.
5. Set up the in for the loop: for ( in ) {}. You’ve named the looping variable and know the vector of values to loop over.
6. Generalize your line from (d) so that it’ll work inside the for loop.

Extra Credits - Question 4

1. 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).