Lab9

2022-11-02

Tasks

- Loops: `for()`, `while()`
- Loop-related: `apply`, `lapply`,`sapply`, `replicate`
- Sampling in R: `sample`, `sample_n`, `rep_sample_n`
- Use one or more of these tools to solve Exercise 3 (Lab 6)
- More on conditional statements
- Lab 9 Exercise 1 and Lab 9 Exercise 2
- R Quiz 9: two chapters from Intermediate R on DataCamp ("Conditionals and Control Flow" and "Loops")

Conditional statements

The main conditional statements in R are:

- `if`: we saw this in Lab 8
- `if`; `else`: we saw this in Lab 8
- `else if`: new!

Conditional statements: `else if`

Basic syntax

```
if(condition){ #if condition is true
  statement #do something
}else if(condition 2){
  statement 2 #do this other thing
}else{ #if neither statement nor statement 2 are true
  statement 3 # do this other other thing
}
```

Conditional statements: `else if`

```r
lunch<-"Dog food"
if(lunch=="Pizza"){ #if first statatement
  print("Yay!")
}else if(lunch=="Rice & beans"){
  print("Sounds good!")
}else{
  print("Meh")
}
```

```
## [1] "Meh"
```

Conditional statements: `else if`

```
#you can use negatives too
lunch<-"Dog food"
if(lunch!="Pizza" & lunch!="Rice & beans"){ #if first stat
  print("Meh")
}else if(lunch=="Rice & beans"){
 print("Sounds good!")
}else{
  print("Yay!")
}
```

```
## [1] "Meh"
```

Loops in R

Three kinds of loops:

- `for` loop
- `while` loop
- `repeat` loop [we won't discuss]

Loops in R: `for` loops

Basic syntax

```r
for (value in sequence){

  statement #DO SOMETHING
}
```

Loops in R: `for` loops

```r
# iterate over elements of a sequence
for (i in 1: 4){

    print(i * 2)

}
```

```
## [1] 2
## [1] 4
## [1] 6
## [1] 8
```

Loops in R: `for` loops

```r
#iterate over elements of a vector
for (i in c(-8^2, 2*4, 239, 29321)){

    print(i)

}
```

```
## [1] -64
## [1] 8
## [1] 239
## [1] 29321
```

Loops in R: `for` loops

```r
#create vector outside of loop and then use it to iterate
x <- c(-8^2, 2*4, 239, 29321)
for (i in x){

    print(i)

}
```

```
## [1] -64
## [1] 8
## [1] 239
## [1] 29321
```

Loops in R: `while` loops

Basic syntax

```
while ( condition ) {

  statement #do something
}
```

Loops in R: while loops

```r
y<-0
while (y <= 5){ #logical
    # statements
    print(y) # if y <=5, print it
    y = y + 1 #if y <=5 add 1 to it
}
```

```
## [1] 0
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

```r
y
```

```
## [1] 6
```

Loops in R: `while` loops

```r
# whose factorial will be calculated
n <- 5
factorial <- 1 # assigning the factorial variable
i <- 1 # and iteration variable to 1
# using while loop
while (i <= n){
    # multiplying the factorial variable
    # with the iteration variable
    factorial = factorial * i
    # incrementing the iteration variable
    i = i + 1
}
print(factorial) # displaying the factorial
```

```
## [1] 120
```

Looping in the command line

`lapply()`: Loop over a `list` and evaluate a function on each element

`sapply()`: Same as `lapply` but try to simplify the result

`apply()`: Apply a function over the margins of an array

There are others, like `tapply()`, `mapply()`, but we won't discuss them.

Looping in the command line:`lapply()`

- it loops over a list, iterating over each element in that list
- it applies a function to each element of the list (a function that you specify)
- and returns a list (the `l` is for “list”).

Looping in the command line:`lapply()`

Basic syntax for `lapply` and `sapply`

```
lapply(X,FUN)
lapply(Data, Function to apply to data)
```

Looping in the command line:`lapply()`

```r
#rnorm samples numbers randomly from a normal distribution
set.seed(123)
x <- list(a = 1:4, b = rnorm(10), c = rnorm(20, 1), d = rnc
#str(x)
lapply(x, mean)
```

```
## $a
## [1] 2.5
##
## $b
## [1] 0.07462564
##
## $c
## [1] 0.8920315
##
## $d
## [1] 5.021617
```

Looping in the command line:sapply()

```
#rnorm samples numbers randomly from a normal distribution
set.seed(123)
x <- list(a = 1:4, b = rnorm(10), c = rnorm(20, 1), d = rno
#str(x)
sapply(x, mean)
```

```
##          a          b          c          d
## 2.50000000 0.07462564 0.89203154 5.02161711
```

Looping in the command line: apply()

Basic syntax

```
apply(X,MARGIN, FUN)
lapply(Data, Where the Function will be applied, Function t
#Margin: 1 is rows
#Margin: 2 is cols
```

Looping in the command line: `apply()`

```r
set.seed(123)
#make a matrix
mat<-matrix(rnorm(20,1), ncol=2)
apply(mat,1, mean) # 1 signifies rows
```

```
##  [1]  1.3318031  1.0648182  1.9797399  1.0905956  0.7867
##  [7]  1.4793833 -0.6158392  1.0072515  0.5407733
```

```r
rowMeans(mat) #equivalent
```

```
##  [1]  1.3318031  1.0648182  1.9797399  1.0905956  0.7867
##  [7]  1.4793833 -0.6158392  1.0072515  0.5407733
```

Looping in the command line: apply()

```r
set.seed(123)
#make a matrix
mat<-matrix(rnorm(20,1), ncol=2)
apply(mat,2, mean) # signifies columns
```

```
## [1] 1.074626 1.208622
```

```r
colMeans(mat) #equivalent
```

```
## [1] 1.074626 1.208622
```

The replicate function

```
#replicate the value 3 exactly 10 times
replicate(n=10, 3)
```

```
##  [1] 3 3 3 3 3 3 3 3 3 3
```

```
#replicate the letter 'A' exactly 7 times
replicate(n=7, 'A')
```

```
## [1] "A" "A" "A" "A" "A" "A" "A"
```

The replicate function

```
set.seed(1)
#generate 3 values that follow normal distribution (replic
replicate(n=4, rnorm(3, mean=0, sd=1))
```

```
##            [,1]       [,2]      [,3]       [,4]
## [1,] -0.6264538  1.5952808 0.4874291 -0.3053884
## [2,]  0.1836433  0.3295078 0.7383247  1.5117812
## [3,] -0.8356286 -0.8204684 0.5757814  0.3898432
```

Practice!

Lab9_ex2.Rmd

Sampling in R

`sample(x = , size = , replace = , prob = )`: Generate a sample of size size, from a vector `x`, with `(replace = TRUE)` or without `(replacement = FALSE)` replacement. By default the size is the length of `x`, sampling occurs without replacement and probabilities are equal. Change these defaults by specifying a value for the argument. For example, to have unequal sampling probabilities, include a vector of length `x`, in which the i^th entry describes the relative probability of sampling the i^th value in `x`.

`sample_n(tbl = , size = , replace = , weight = )`: Generate a sample of size `size`, from a `tibble` `tbl`, with `(replace = TRUE)` or without `(replacement = FALSE)` replacement. All arguments are the same as in `sample()` except `weight` replaces `prob`, and `tbl` replaces `x`. `sample_n()` is a function in the `dplyr` package, which is loaded with `tidyverse`.

New functions for sampling in R!

These functions extend the functionality of dplyr::sample_n() and dplyr::slice_sample() by allowing for repeated sampling of data. This operation is especially helpful while creating sampling distributions

These are related to `dplyr::sample_n` and ‘dplyr::slice_sample, respectively:

`rep_sample_n()`: Generate `reps` samples of size `size`

`rep_sample_slice()`: Generate `reps` samples of size `n`

Both return a tibble grouped by replicate. ‘

New functions for sampling in R!

```
library(infer) # a tidymodels package
#sample 2 rows from the iris dataset without replacement. 
set.seed(123)
test1<-iris %>%
  rep_sample_n(size = 3, replace = F, reps = 2) #uses size
test1
```

```
## # A tibble: 6 x 6
## # Groups:   replicate [2]
##   replicate Sepal.Length Sepal.Width Petal.Length Petal.
##       <int>        <dbl>       <dbl>        <dbl>
## 1         1          4.3         3            1.1
## 2         1          5           3.3          1.4
## 3         1          7.7         3.8          6.7
## 4         2          4.4         3.2          1.3
## 5         2          4.3         3            1.1
## 6         2          7.7         3.8          6.7
```

New functions for sampling in R!

```
library(infer) # a tidymodels package
#sample 2 rows from the iris dataset without replacement. I
set.seed(123)
test2<-iris %>%
  rep_slice_sample(n = 3, replace = F, reps = 2) #uses n ir
test2
```

```
## # A tibble: 6 x 6
## # Groups:   replicate [2]
##   replicate Sepal.Length Sepal.Width Petal.Length Petal.
##       <int>        <dbl>       <dbl>        <dbl>
## 1         1          4.3         3            1.1
## 2         1          5           3.3          1.4
## 3         1          7.7         3.8          6.7
## 4         2          4.4         3.2          1.3
## 5         2          4.3         3            1.1
## 6         2          7.7         3.8          6.7
```

New functions for sampling in R!

```
test1==test2
```

```
##      replicate Sepal.Length Sepal.Width Petal.Length Pet
## [1,]      TRUE         TRUE        TRUE         TRUE
## [2,]      TRUE         TRUE        TRUE         TRUE
## [3,]      TRUE         TRUE        TRUE         TRUE
## [4,]      TRUE         TRUE        TRUE         TRUE
## [5,]      TRUE         TRUE        TRUE         TRUE
## [6,]      TRUE         TRUE        TRUE         TRUE
```

Back to the Exercise: Making a distribution of sample means (from lab 6)

Last week you learned how to take a sample (random rows) from a dataset and calculate statistics on it.

Specifically, you took two independent samples from the `human_genes` dataset and calculated some descriptive statistics.

Exercise: Making a distribution of sample means

Step 1: Read in the `human_genes.csv` dataset and get summaries

```
library(dplyr) #load the dplyr package
library(readr) #load the readr package
#read in the human genes dataset
human_genes<-readr::read_csv("input_files/human_genes.csv"
```

```
## Rows: 22385 Columns: 4
## -- Column specification -------------------------------
## Delimiter: ","
## chr (3): gene, name, description
## dbl (1): size
##
## i Use `spec()` to retrieve the full column specificatio
## i Specify the column types or set `show_col_types = FAL
```

Exercise: Making a distribution of sample means

Step 1: Read in the `human_genes.csv` dataset and get summaries

```
glimpse(human_genes) #glimpse the dataset
```

```
## Rows: 22,385
## Columns: 4
## $ gene        <chr> "ENSG00000000003.14", "ENSG0000000000
## $ size        <dbl> 3796, 1339, 1161, 6364, 4355, 2729,
## $ name        <chr> "TSPAN6", "TNMD", "DPM1", "SCYL3", "
## $ description <chr> "tetraspanin", "tenomodulin", "dolic
```

```
#remove useless columns
human_genes<-human_genes %>%
  select(name, size)
```

Exercise: Making a distribution of sample means

Step 2: Calculate the following descriptive statistics for the dataset: `mean`, `median`,`standard deviation`, `mode`, `IQR`.

```
#summarise the human genes dataset creating three columns
human_genes_summ <- human_genes %>%
  summarise(
  MeanLength = mean(size),
  MedianLength = median(size),
  SDLength = sd(size),
  IQR = IQR(size)
)
human_genes_summ #
```

```
## # A tibble: 1 x 4
##   MeanLength MedianLength SDLength   IQR
##        <dbl>        <dbl>    <dbl> <dbl>
## 1      3511.         2744    2833.  2827
```

Exercise: Making a distribution of sample means

Step 3: take 10,000 random samples of length `n=100` from the `human_genes` dataset and calculate the same summaries as above.

Wow! Let's break this down:

- sample 10 random samples of length `n=100`
- calculate the same summaries as above for each sample
- if everything works, switch to 10,000

Exercise: Making a distribution of sample means

Step 3a: sample 10 random samples of length `n=100` (using `for`)

```
library(dplyr)
x<-seq(1:10) #create a vector with numbers from 1 to 10000
# for each iteration,take a sample of size 100 without rep
samps<-list() #create empty list to put resamps in
for(i in x){ #for loop
  samps[[i]]<-human_genes %>%
    sample_n(size=100, replace=F)
  print(i) #print each iteration
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
```

Exercise: Making a distribution of sample means

Step 3b: calculate the same summaries as above for each sample

```r
res<-list() #create empty list to put the summaries in
for(i in 1:length(samps)){ #for loop for each elements in 
  res[[i]]<- samps[[i]] %>%
   summarise(
    MeanLength = mean(size),
    MedianLength = median(size),
    SDLength = sd(size),
    IQR = IQR(size)
)
  print(i) #print each iteration
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

Exercise: Making a distribution of sample means

Can we do it all in one go?

```r
x<-seq(1:10) #create a vector with numbers from 1 to 10000
#create tibble with the columns produced by our summaries
#dummy tibble
resamps<-tibble(MeanLength=NA, MedianLength=NA, SDLength=NA
for(i in x){ #for loop
  temp<-human_genes %>% #assign to temp object
    sample_n(size=100, replace=F) %>% #sample
    summarise( # summaries
    MeanLength = mean(size),
    MedianLength = median(size),
    SDLength = sd(size),
    IQR = IQR(size)
    )
  resamps<-bind_rows(resamps, temp) #add new row to tibble
  print(i) #print each iteration
}
```

Exercise: Making a distribution of sample means

```
#the first row has NAs. we can easily get rid of it.

resamps<-na.omit(resamps) #remove lines with NA
resamps
```

```
## # A tibble: 10 x 4
##    MeanLength MedianLength SDLength   IQR
##         <dbl>        <dbl>    <dbl> <dbl>
##  1      3269.        2716.    2187. 2952.
##  2      3772.        3020.    2489. 3320.
##  3      3704.        2936     3283. 2936.
##  4      3945.        3260     2664. 2564.
##  5      3506.        2828     2611. 3014.
##  6      4064.        2756.    4847. 3218.
##  7      3220.        2633     2404. 2863
##  8      3362.        2709     2550. 2629.
##  9      3377.        2869     2132. 3260.
## 10      3437.        2380     3193. 2570.
```

Exercise: Making a distribution of sample means

Step 3c: if everything works, switch to 10,000

```
nsamps=10000
resamps<-tibble(MeanLength=NA, MedianLength=NA, SDLength=NA
for(i in 1:nsamps){ #for loop
  temp<-human_genes %>% #assign to temp object
    sample_n(size=100, replace=F) %>% #sample
    summarise( # summaries
    MeanLength = mean(size),
    MedianLength = median(size),
    SDLength = sd(size),
    IQR = IQR(size)
    )
  resamps<-bind_rows(resamps, temp) #add new row to tibble
  if(i %in%  seq(from=1, to=nsamps, by=500)){
  print(i) #print 1st,501st,1001st etc iterations
  }
}
```

Exercise: Making a distribution of sample means

Another option: using `rep_slice_sample`

Make a function that calculates all the summaries

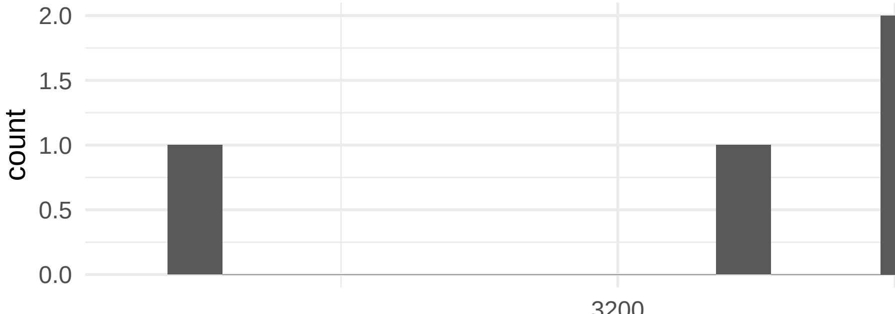
```
library(infer)
#samps
nsamps<-10 #once this works, switch to 10,000
reps<-rep_sample_n(human_genes, size=100, replace=FALSE, re
  group_by(replicate) %>%
  summarise( # summaries
  MeanLength = mean(size),
  MedianLength = median(size),
  SDLength = sd(size),
  IQR = IQR(size)
) %>%
  ungroup
```

Exercise: Making a distribution of sample means

Step 4: Plot the sampling distribution with ggplot

```r
library(ggplot2)
#rerun previous code with nsamps=10000
ggplot(reps, aes(x=MeanLength)) +
  geom_histogram() +
  theme_minimal()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `
```



Exercise: Making a distribution of sample means

Final step: Obtain the grand mean (mean of sample means) and the standard deviation of the sample means and: compare with the population mean; calculate SEM using the distribution and the formula and compare.

```
#Try it!
```

Exercises!

- Binomial hypothesis testing: `lab9_ex1.Rmd`
- Loops & Conditional Statements: Exercises in `lab9_ex2.Rmd`.