Programación Básica

Rodrigo Negrete Pérez

January 26, 2022

Section 1

For Loops

En la sesión pasada creamos un df usando datos pseudo-aleatorios

```
set.seed(2020)
n<-4000
df<-data.frame(
  id=1:n,
  sex=sample(c('h','m','o'), n, replace = T),
  edad=floor(runif(n, min=17, max=27)),
  carrera=sample(c('eco','cpol','ri','derecho','conta','mat'),
  prom=runif(n, 6, 10)
)
head(df)
```

```
##   id sex edad carrera     prom
## 1  1   o   25 derecho 7.571058
## 2  2   m   18   conta 9.999865
## 3  3   h   22   conta 6.506111
## 4  4   h   21 derecho 7.422582
## 5  5   m   17     mat 6.238837
```

- ¿Cuánto es el promedio de los promedios?

```
with(df, mean(prom))
```

```
## [1] 7.993034
```

- Pero técnicamente ya lo sabíamos. Pues estos datos provienen de una distribución uniforme.

$E[Unif(a, b)] = (a + b)/2$

Entonces

$E[prom] = (6 + 10)/2 = 8$

- ¿Fue una coincidencia? Después de todo, los datos son aleatorios
- Podemos generar los datos varias veces y tomar una foto del promedio de los promedios. ¿Cómo se ve esta ráfaga de fotos?

For Loops

- En ocasiones queremos hacer tareas repetitivas. Podríamos correr nuestro df y recordar el promedio de los promedios, pero es engorroso.
- Mejor usemos un loop.
- la idea básica es tener un vector sobre el cual vamos a iterar. La variable iteradora es cada una de las entradas de ese vector.

for (*variable_iterable* in **vector**){

chunk de codigo

}

- *Para cada uno de los elementos de este vector, haz lo que sigue*

- A menudo necesitamos vectores vacíos para ir guardando la información de cada uno de los ciclos que realicemos.

```
means<-NULL
```

- Las entradas de este vector vacío serán cada uno de los promedios que calculemos, y los iremos añadiendo según la variable iteradora

Aterricemos:

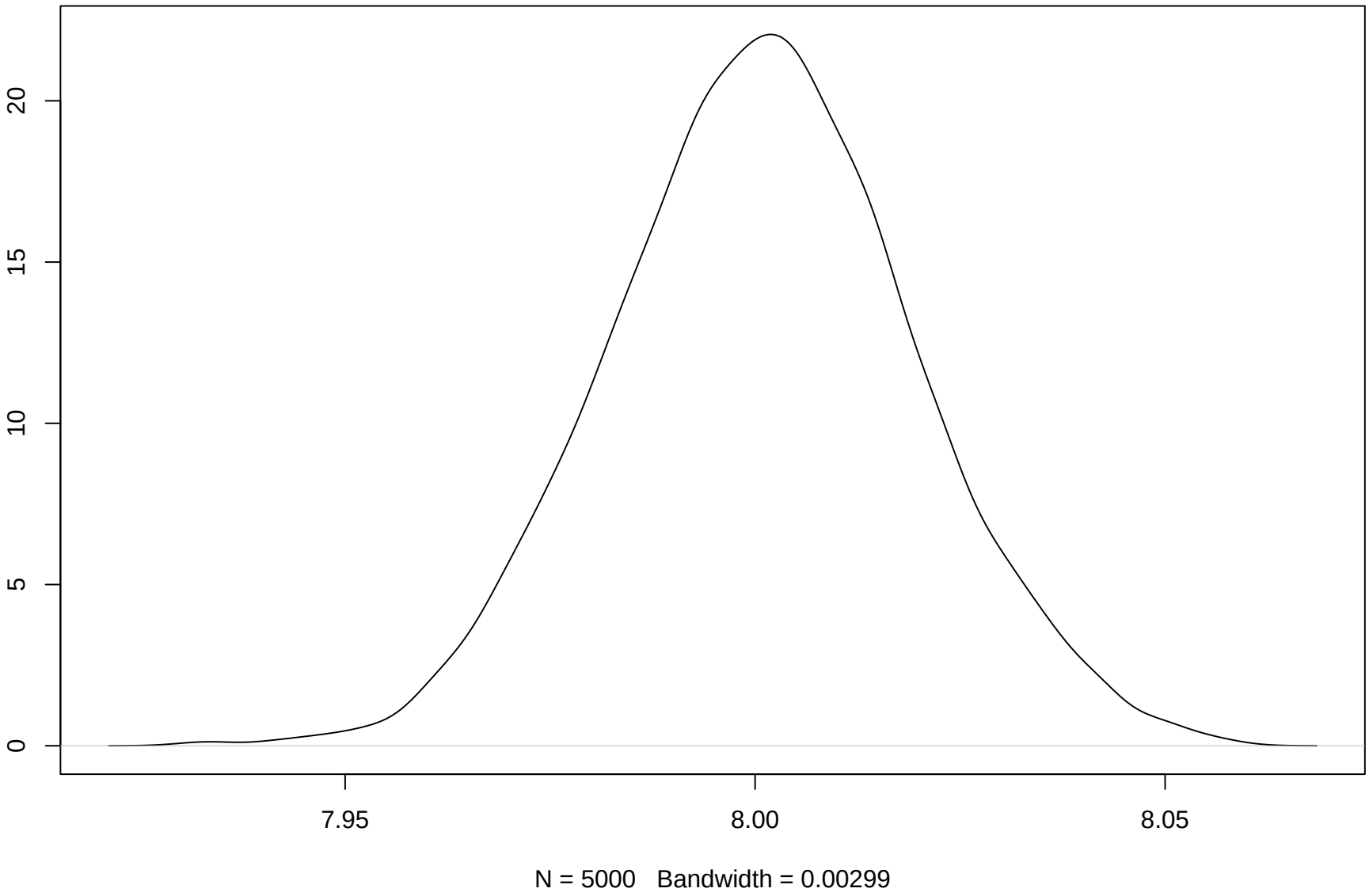
- Lo que queremos es crear el mismo df de alumnos del ITAM varias veces - digamos unas 5000 veces- y calcular el promedio de los promedios para cada uno de los df creados.
- Cada uno de esos promedios los guardaremos como entradas de nuestro vector **means** para luego hacer cosas con él.

Veamos cómo sería el loop for

```
means<-NULL #Vector vacio
num.reps<-5000 # # de repeticiones
for (i in 1:num.reps) {
  df<-data.frame(
  id=1:n,
  sex=sample(c('h','m','o'), n, replace = T),
  edad=floor(runif(n, min=17, max=27)),
  carrera=sample(c('eco','cpol','ri','derecho','conta','mat'),
  prom=runif(n, 6, 10)) #Creamos el df
  means[i]<-with(df, mean(prom)) #Guardamos en el vector
}
```

Para cada una de las entradas del vector 1:num.reps - que resulta en hacer este proceso num.reps veces - crea un df. Después, guarda el promedio de los promedios para cada una las repeticiones en el vector means.

- Grafiquemos el vector means. No se preocupen cómo, luego le dedicaremos un bloque entero a graficar


0
5
10
15
20
7.95
8.00
8.05
N = 5000 Bandwidth = 0.00299

Loops sobre variables

- Para hacer loops sobre variables, el loop se escribe ligeramente diferente.
- R tiene algunas bases de datos ya integradas: usémos la más famosa para un ejemplo:

```
data('mtcars')
str(mtcars)
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1
```

- Supongamos que queremos el promedio de mpg, hp, qsec y cyl.
- Podríamos calcular promedios para cada uno, o hacer un loop.
- Para mostrar el resultado dentro de un loop tenemos que correr la función **print()**

```
data(mtcars)
for (i in c('mpg','hp','qsec','cyl')) {
  print(mean(mtcars[[i]]))
}
```

```
## [1] 20.09062
## [1] 146.6875
## [1] 17.84875
## [1] 6.1875
```

- Intuitivamente, hubiéramos puesto mtcars$i, pero así no funciona R. Tenemos que poner mtcars[[i]]

- Podríamos usar la función paste0 para incluir un mensajito.
- Instalen y cargen el paquete “stringr”
- Usaremos la funcion **str_c**
- Muy intuitiva, solo ponemos el mensaje separado por espacios y comas

```
library(stringr)
```

```
data(mtcars)
for (i in c('mpg','hp','qsec','cyl')) {
  print(str_c('El promedio de ',i ,' es ', mean(mtcars[[i]])))
}
```

```
## [1] "El promedio de mpg es 20.090625"
## [1] "El promedio de hp es 146.6875"
## [1] "El promedio de qsec es 17.84875"
## [1] "El promedio de cyl es 6.1875"
```

Section 2

if

if

- A veces queremos que R haga cosas, pero solo si los datos cumplen cierta condición

if (*condición a probar*){

chunk de codigo

}

- En la condición a probar vamos a poner un test que involucre operadores lógicos. Solo si lo que está dentro es TRUE, R va a hacer lo que está en el chunk de código
- Podemos omitir las llaves

- Si se cumple la condición, hace lo que dice; de lo contrario, no hace nada.

```
if(2>0){
  print('Hola')
}
```

```
## [1] "Hola"
```

```
if(2==0){
  print('Hola')
}
```

else

- Podemos añadir un else{ } para especificar qué pasa en caso de que la condición sea falsa

```
a<-rnorm(1, mean=0, sd=1)
if(a>0){
  print('Hola')
} else {
  print('Adiós ')
}
```

```
## [1] "Hola"
```

Section 3

Nested loops

Nested loops

Podemos combinar los for e if para crear cosas bastante interesantes.
Necesitamos dos comandos antes

- **next** se va al próximo elemento del iterador, continuando con el loop.
- **break** rompe el loop.

- Imprime los números entre 57 y 65 que son pares

```
for (i in 57:65) {
  if(!i %%2){
    print(i)
    next
  }
}
```

```
## [1] 58
## [1] 60
## [1] 62
## [1] 64
```

- Si en lugar de todos, queremos solo el primero

```
for (i in 57:65) {
  if(!i %%2){
    print(i)
    break
  }
}
```

```
## [1] 58
```

Section 4

Funciones

Funciones

- Ya hemos utilizado algunas funciones incorporadas a R, pero nosotros podemos crear nuestras propias funciones

func_name <-**function**(input){

statement

return (output)

}

Por ejemplo, pensemos en una función que, si le pongo el df de alumnos del ITAM, calcule el promedio de los promedios por carrera.

- Recordemos la base de datos

```
n<-4000
df<-data.frame(
  id=1:n,
  sex=sample(c('h','m','o'), n, replace = T),
  edad=floor(runif(n, min=17, max=27)),
  carrera=sample(c('eco','cpol','ri','derecho','conta','mat'),
  prom=runif(n, 6, 10)
)
```

```
career.mean<-function(data_frame){
  career_mean<-NULL
  for (i in df$carrera ) {
  career_mean[i]<-with(subset(df, carrera==i), mean(prom))
  }
  return(career_mean)
}

career.mean(df)
```

```
##      mat      eco  derecho       ri     cpol    conta
## 8.026901 8.010372 8.038149 7.958560 7.931990 8.044815
```

- Las funciones pueden tener varios argumentos.
- Pueden aceptar cualquier tipo de objeto.
- Pueden tener argumentos default, si hacemos que a=2. La función tomaría el 2 como predeterminado
- Notemos que aparecen en el environment

```
potencia<- function(x, y=2) {
  result<-x^y
  return(result)
}

potencia(2,2); potencia(2,4)
```

```
## [1] 4

## [1] 16
```

Section 5

Conclusiones

Automatización

- Puede que los loops y funciones parezcan que estamos flexeando, pero ayudan a automatizar.
- Procesos que tenemos que hacer muchas veces son fácilmente resueltos por el poder de las computadoras.
- Siempre automaticen labores tediosas. Vale la pena invertir el tiempo extra en averiguar cómo se hace el loop.

Section 6

Ejercicios

- Crea una función que arroje el primer número divisible entre 17 en un vector
- Retoma el df de alumnos del ITAM.
  - Añade una variable que mida las faltas acumuladas de la carrera del individuo. Después, calcula el promedio de todas las variable numéricas que no sean el id usando un nested loop.
  - Crea un programa que reciba el input de la carrera y escupa el idd del estudiante con mayor calificación de dicha carrera. Asume que el input siempre va a estar bien escrito.