Tarea 1 | STAT NT

Luis Gagñevin 5.153.261-6

4/29/2021

1 Ejercicio 1

1.1 Parte 1: Vectores

1.1.1 Dado los siguientes vectores, indica a que tipo coercionan

```
w <- c(29, 1L, FALSE, "HOLA")
x <- c("Celeste pelela!", 33, NA)
y <- c(seq(3:25), 10L)
z <- paste(seq(3:25), 10L)
```

Las Coerciones ocurren del mas rigido al mas flexible (Logic -> Integer -> Numeric -> Character) Y en caso de combinaciones toma al mas flexible como su tipo, teniendo esto en cuenta:

W es un vector del tipo Character

W Tiene Elementos clasificados como: Logic, Integer, Numeric y Character, por lo que tomara el mas flexible, osea Character.

X es un vector del tipo Character

X Tiene Elementos clasificados como: Logic, Numeric y Character, por lo que tomara el mas flexible, osea Character.

Y es un vector de ltipo Integer

Y tiene Elementos clasificados solo como Integer

Z es un vector del tipo character

Aunque Z sea practicamente igual a y dentro de los (), el uso de paste concatena los vectores luego de convertirlos en texto, por lo que genera un vector con solo elementos del tipo Character

1.1.2 ¿Cual es la diferencia entre c(4,3,2,1) y 4:1?

Si bien su resultado se ve igual, la clasificacion de estos es distinta. Ya que 4:1 es tomado como una cadena entera, mientras que c(4,3,2,1) se toma como un vector numerico.

1.2 Parte 2: factor

Dado el siguiente factor x:

```
x <-
factor(
c(
"alto",
"bajo",
"medio",
"alto",
"muy alto",
"bajo",
"medio",
"alto",
"ALTO",
"MEDIO",
"BAJO",
"MUY ALTO",
"QUE LOCO",
"QUE LOCO",
"QUE LOCO",
"A",
"B",
"C",
"GUAU",
"GOL",
"MUY BAJO",
"MUY BAJO",
"MUY ALTO"
)
)
```

1.2.1 Genera un neuvo factor (llamalo xx) transformando el objeto x previamente generado de forma que quede como sigue:

[1] A B M A A B M A A M B A B B A Levels: B < M < A

Para reordenar y que quede de la misma forma, tomare los niveles que me sirven dejando como NA las palabras que no contienen ni “Alto, Medio o Bajo”.

Luego cambiare cada nivel por “alto”, “bajo” y “medio”

Luego elimino los NA y cambio los niveles por “A, B y M” dejandolo casi listo

Y para finalizar ordeno los niveles donde B sea el menor y A el mayor

```
xx <- factor(x, levels= c("alto","ALTO", "bajo", "BAJO", "medio",
                          "MEDIO", "muy alto", "MUY ALTO", "MUY BAJO"))
```

```
levels(xx)[levels(xx)=="ALTO"]<- "alto"
levels(xx)[levels(xx)=="MUY ALTO"]<- "alto"
levels(xx)[levels(xx)=="muy alto"]<- "alto"
levels(xx)[levels(xx)=="BAJO"]<- "bajo"
levels(xx)[levels(xx)=="MUY BAJO"]<- "bajo"
levels(xx)[levels(xx)=="muy bajo"]<- "bajo"
levels(xx)[levels(xx)=="MEDIO"]<- "medio"

xx<-xx[!is.na(xx)]
levels(xx)[levels(xx)=="alto"]<- "A"
levels(xx)[levels(xx)=="bajo"]<- "B"
levels(xx)[levels(xx)=="medio"]<- "M"

xx<- ordered(xx, levels=c("B","M","A"))
```

```
##  [1] A B M A A B M A A M B A B B A
## Levels: B < M < A
```

1.2.2. Generá el siguiente data.frame()

Para ello usá el vector xx que obtuviste en la parte anterior.

Para generar el data.frame() tome los valores unicos de x en una nueva variable, genere un df con los valores de la nueva variable y luego en una nueva df tomamos los niveles de x y los ordenamos haciendo match a los valores de las variables "Levels" para luego ordenarlo en un solo df, dejandolo de minusculas a mayusculas.

```
x2<-unique(x)
df<- data.frame(x2[1:15])
names(df)[1]<- "levels"

df2<-data.frame(levels(x))
df2$value<- c(1:15)
names(df2)[1]<- "levels"
df2<-df2[order(match(df2$levels, df$levels)),]
df$value<- df2[,2]
```

```
##      levels value
## 1      alto     2
## 2      bajo     5
## 3     medio    10
## 4  muy alto    12
## 5      ALTO     3
## 6     MEDIO    11
## 7      BAJO     6
## 8  MUY ALTO    13
## 9  QUE LOCO    15
## 10        A     1
## 11        B     4
## 12        C     7
## 13     GUAU     9
## 14      GOL     8
## 15 MUY BAJO    14
```

1.3 Parte 2: Listas

1.3.1 Genera una lista que se llame lista_t1 que contenga:

- Un vector numérico de longitud 4 (h).
- Una matriz de dimensión 4*3 (u).
- La palabra “chau” (palabra).
- Una secuencia diaria de fechas (clase Date) desde 2021/01/01 hasta 2021/12/30 (fecha)

```
h<- c(1,2,3,4)
u<- matrix(1:12, nrow=4)
palabra<- "chau"
fecha <- seq(as.Date('2021-1-1'), as.Date('2021-12-30'), by='day')
lista_t1<- list(h,u,palabra,fecha)
```

1.3.2 ¿Cual es el tercer elemento de la primera fila de la matriz m? ¿Que columna lo contiene?

Para realizar eso haremos lo siguiente:

```
lista_t1[[2]][3,1]
```

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

1.3.3 ¿Cual es la diferencia entre hacer lista_t1[[2]][]<-0 y lista_t1[[2]] <- 0?

Al realizar lista_t1[[2]][] <-0 quedan todos los valores de la matriz iguales a 0

```
lista_t1[[2]][]<-0
```

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

Mientras que al realizar lista_t1[[2]] <- 0, cambia la matriz por el valor 0.

```
lista_t1[[2]]<-0
```

0

1.3.4 Iteracion

> Iterá sobre el objeto lista_t1 y obtené la clase de cada elemento teniendo el cuenta que si la longitud de la clase del elemento es mayor a uno nos quedamos con el último elemento. Es decir, si class(x) es igual a c(“matrix”, .array“) el resultado debería ser “array”. A su vez retorná el resultado como clase list y como character

Para lograr esto, lo que realize es crear una funcion donde utiliza un for() con la variable i en la que mira cuales son las clases de cada parte de la lista y luego cuenta sus elementos, si estos son iguales a 1, coloca el valor correcto, si estos son mayores a 1 coloca el ultimo tipo de clase

```
clasificaciones <- function(lista){
  clases <- list()
for (i in 1:length(lista)) {
  cls<- class(lista[[i]])
  if (length(cls)==1) {
    clases[[i]]<- as.character(cls)
  }else{
    clases[[i]]<- as.character(cls[length(cls)])
  }
}
print(clases)
}
```

```
clasificaciones(lista_t1)
```

```
## [[1]]
## [1] "numeric"
##
## [[2]]
## [1] "array"
##
## [[3]]
## [1] "character"
##
## [[4]]
## [1] "Date"
```

1.3.5 Iteracion (2)

> Utilizando las últimas 10 observaciones de el elemento "fecha" del objeto "lista_t1" escriba para cada fecha "La fecha en este momento es . . . ." donde ". . ." debe contener la fecha para valor de lista$fecha. Ejemplo: "La fecha en este momento es '2021-04-28' ". Hacelo de al menos 2 formas y que una de ellas sea utilizando un for. Obs: En este ejercicio NO imprimas los resultados.

Forma 1)

En esta forma utilizamos una funcion del tipo for() donde toma los valores de un data.frame con 10 fechas y los acomoda en una oracion

```
lista <- data.frame(tail(lista_t1[[4]], 10))
names(lista)[1] <- "fecha"

for (i in 1:10) {
  print(paste("La fecha en este momento es ", as.character(lista$fecha[i])))
}
```

Forma 2)

En esta forma utilizamos una funcion del tipo while donde toma los valores de un data.frame con 10 fechas y los acomoda en una oracion

```r
lista <- data.frame(tail(lista_t1[[4]], 10))
names(lista)[1] <- "fecha"

i=1
while (i <= 10) {
  print(paste("La fecha en este momento es ", as.character(lista$fecha[i])))
  i=i+1
}
```

1.4 Parte 3: Matrices

1.4.1 Genera una matriz A de dimension 4x3 y una matriz B de dimension 4x2 con numeros aleatorios usando alguna funcion predefinida en R

Elegimos los valores aleatorios utilizando la funcion sample()

```r
A<- matrix(sample(1:100,12), ncol=3)
B<- matrix(sample(1:100,8), ncol=2)
```

1.4.2 Calcula el producto elemento a elemento de la primera columna de la matriz A por la ultima columna de la matriz B

```r
A[,1]*B[,2]
```

```
## [1]  264  552 1479 1680
```

1.4.3 Calcula el producto matricial entre D=AT B luego selecciona los elementos de la primera y tercera fila de la segunda columna (en un paso)

Para seleccionar ambos valores con un solo paso utilizamos un vector

```r
D<- t(A)%*%B
D[c(1,3),2]
```

```
## [1] 3975 4350
```

1.4.4

> Usa las matrices A y B de forma tal de lograr una matriz C de dimension 4x5. Con la funcion Attributes inspecciona los atributos de C. Posteriormente renombra filas y columnas como “fila_1”, “fila_2” ... “columna_1”, “columna_2”, vuelve a inspeccionar los atributos. Finalmente, generaliza y escribi una funcion que reciba como argumento una matriz y devuelva como resultado la misma matriz con columnas y filas con nombres.

Utilizamos la funcion cbind() para combinar ambas matrices y veo con el atributo que solo tiene el $dim

```r
c<- cbind(A,B)
attributes(c)
```

```
## $dim
## [1] 4 5
```

Utilizo como primer forma la funcion provideDimnames() y el atributo nos muestra los $dimnames de las columnas y filas en un formato parecido a una lista

```r
c<- provideDimnames(c, sep="_", base=list("fila", "columna"))
attributes(c)
```

```
## $dim
## [1] 4 5
##
## $dimnames
## $dimnames[[1]]
## [1] "fila"   "fila_1" "fila_2" "fila_3"
##
## $dimnames[[2]]
## [1] "columna"   "columna_1" "columna_2" "columna_3" "columna_4"
```

Aca realizo una funcion general que hace un nested loop con dos for donde renombra todas las filas y columnas de una matriz segun su posicion

```r
nombres <- function(matriz){
  filas<-dim(matriz)[1]
  columnas<- dim(matriz)[2]

  for (i in 1:filas) {
      for (j in 1:columnas) {
    rownames(matriz)[i]<- paste("fila_",i, sep="")

    colnames(matriz)[j]<- paste("columna_",j, sep="")
      }}

  return(matriz)
}
```

```
##        columna_1 columna_2 columna_3 columna_4 columna_5
## fila_1        88        86        89        41         3
## fila_2        23        34        43         4        24
## fila_3        17        93        13        90        87
## fila_4        42        87        48        27        40
```

1.4.5 Puntos Extra: Generaliza la funcion para que funcione con arrays de forma que renombre filas, columnas y matrices

```
renombre <- function(ary){
  ary <- provideDimnames(ary, sep="_", base=c("fila","columna","matriz"), unique=TRUE)

  return(ary)
  }
```

```
##        columna columna_1 columna_2 columna_3 columna_4
## fila        88        86        89        41         3
## fila_1      23        34        43         4        24
## fila_2      17        93        13        90        87
## fila_3      42        87        48        27        40
```

Ejercicio 2

2.1 Parte 1: ifelse()

2.1.1 ¿Que hace la funcion ifelse() del paquete base de R?

La funcion ifelse() funciona como un conjunto de if y else, utiliza tres valores e itera sobre todo el vector o matriz dado de ser asi:

- El primero es un test logico, por ejemplo si a>1.
- El segundo es el resultado en caso de que el objeto logico sea verdadero
- El tercero es el resultado en caso de que el objeto logico sea falso

2.1.2 Dado el vector x tal que: x <- c(8, 6, 22, 1, 0, -2, -45), utilizando la funcion ifelse() del paquete base, reemplaza todos los elementos mayores estrictos a 0 por 1, y todos los elementos menores o iguales a 0 por 0.

```
x<-c(8, 6, 22, 1, 0, -2, -45)
x<-ifelse(x>0, x<-1, x<- 0)
```

2.1.3 ¿Porque no fue necesario usar un loop?

No fue necesario ya que el funcionamiento de ifelse itera sobre el vector

2.2 Parte 2: while() loops

2.2.1 ¿Que es un while loop y como es la estructura para generar uno en R? ¿En que se diferencia de un for loop?

Un while loop es una herramienta de iteracion para realizar una accion de manera repetida segun mientras se mantenga un objeto logico verdadero

Para generar un while loop es necesario completar 3 partes:

- Un valor que sera testeado para ver si es verdadero o no. Ejemplo: i=1 // i<20
- Una proceso que se realizara mientras que el valor sea verdadero.

- Algo que haga cambiar el valor para que pueda romperse y finalizar la iteracion. Ejemplo: i=i+1

Se diferencia de un for loop ya que el while itera durante todo momento que el valor logico sea verdadero, mientras que un for loop itera mientras que la variable dada se encuentre dentro de los valores del vector dado. Ejemplo: i in 1:50

2.2.2 Dada la estructura siguiente, ¿ Cual es el valor del objeto suma? Responda sin realizar el calculo en R

```
x <- c(1,2,3)
suma <- 0
i <- 1
while(i < 6){
suma = suma + x[i]
i <- i + 1
}
```

El valor del objeto suma sera dado de la manera siguiente:

x es un vector conteniendo los numeros de 1 al 3

suma vale 0 e i vale 1, el while funcionaria siempre que i sea menor a 6 y sumara 1 al final de cada iteracion

Por lo que se seguira el siguiente proceso

- suma= suma+x[1] => suma=1 // i=2
- suma= suma+x[2] => suma=3 // i=3
- suma= suma+x[3] => suma=6 // i=4
- suma= suma+x[4] => x[4] no existe, por lo tanto es NA // i=5
- suma= suma+x[5] => x[5]=NA // i=6 (deja de iterar)

Bien, podemos ver como el while funciona perfectamente hasta i=3, sucede que i=4 e i=5 valen NA y al sumar un numero y un NA nos dara NA como suma ya que estas sumando un numero y un objeto inexistente.

2.2.3

> Modifica la estructura anterior para que suma valga 0, si el vector tiene largo menor a 5 o que sume los primeros 5 elementos si el vector tiene largo mayor a 5. Apartir de ella genera una funcion que se llame sumar_si y verifica que funcione utilizando los vectores y<-c(1:3), z<-(1:15).

Para que funcione, colocamos un if dentro del while que mida el vector y si es menor a 5 da 0.

```
x <- c(1,2,3)
suma <- 0
i <- 1
while(i < 6){
  if(length(x)<5){
    suma=0;
  } else {
    suma=suma+x[i]
  }
i <- i + 1
}
```

Creamos la funcion sumar_si cambiando las variables x por vec y haciendola universal.

```
sumar_si <- function(vec){
suma <- 0
i <- 1
while(i < 6){
  if(length(vec)<5){
    suma=0;
  } else {
    suma=suma+vec[i]
  }
i <- i + 1
}
return(suma)
}
```

Probamos su funcionamiento con z<-(1:15) y con y<-c(1:3)

```
y<-c(1:3)
z<-(1:15)
```

```
## [1] "La suma de el vector y da: 0"
```

```
## [1] "La suma de el vector z da: 15"
```

2.2.4 Genera una estructura que multiplique los numeros naturales (empezando por el 1) hasta que dicha multiplicacion supere el valor de 10000. ¿Cuanto vale dicha productoria?

```
mult<- 1
i=1
while (mult<10000) {
  mult=mult*i
  i=i+1
}
```

```
## [1] "La productoria vale: 40320"
```

```
## [1] "Dicho producto sale de hacer: 5040 * 8"
```

2.3 Parte 3: Ordenar

2.3.1 Genera una funcion ordenar_x() que para cualquier vector numerico, ordene sus elementos de menor a mayor.

La primera forma es utilizando la funcion sort

```
ordenar_x <- function(ordenar){
  sort(ordenar, decreasing = FALSE)
}
```

La segunda forma es con una funcion recursiva que lo que hace es tomar un valor del vector dado, en este caso es el primer valor y mide el resto de elementos en un lapply donde evalua cada valor del vector con respecto al primer valor del mismo y coloca los valores segun si estos son mayores o menores al primero en 2 vectores, si estos vectores son mas largos que 1 recurre en la funcion cambiando el vector inicial por este tramo y luego los almacena en un vector.

```
ordenar_x <- function(ordenar){
  centro<- ordenar[1]
  menor<-c()
  mayor<-c()

  lapply(ordenar[2:length(ordenar)], function(k){
    if (k < centro){
      menor <<- c(menor, k)
    } else {
      mayor<<- c(mayor, k)
    }
  })

  if (length(menor)>1){
    menor<-ordenar_x(menor)
  }

  if (length(mayor)>1){
    mayor <- ordenar_x(mayor)
  }
  ordenado<-c(menor, centro, mayor)
  return(ordenado)
}
```

Testeos, utilizaremos estos vectores::

```
x <- c(3,4,5,-2,1)
y <- c(5,5,1,0,-2,-4,99,99)
```

- El vector ordenado de x queda como: -2, 1, 3, 4, 5
- El vector ordenado de y queda como: -4, -2, 0, 1, 5, 5, 99, 99

2.3.2 ¿Que devuelve order(order(x))?

La funcion order() evalua las posiciones de cada valor del vector y ordena sus posiciones segun donde se encuentre en el vector, por lo que si hacemos x[order(x)[1]], nos dara el valor mas chico de el vector x

Por lo que order(order(x)) va a tomar las posiciones de los vectores de x ordenados y devolvera las posiciones ordenadas de x tal que si hacemos order(x)[order(order(x))[1]] nos dara el valor mas chico de el order(x)

Ejercicio 3

3.1 Extra 1

3.1.1 ¿Que funcion del paquete base es la que tiene mayor cantidad de argumentos?

```
funciones <- mget(ls("package:base", all = TRUE), inherits = TRUE)
funs<- Filter(is.function, funciones)

cantargs <- sapply(funs, function(x) length(formals(x)))
cantargs[which.max(cantargs)]
```

La funcion base con mas argumentos es la funcion 1044 en la lista, siendo esta la funcion scan() con 22 argumentos.

3.2 Extra 2

```
valores <- 1:20
```

Dado el siguiente vector:

3.2.1 Obtene la suma acumulada, es decir 1, 3, 6, 10 de dos formas y que una de ellas sea utilizando al funcion Reduce

La primera forma es con la funcion reduce, donde con la funcion sum, suma de forma acumulada todos los valores

```
Reduce(sum,valores, accumulate=TRUE)
```

La segunda forma es iterando con un for, donde suma los valores y separa el 1.

```
sumacum<- c()
for (i in valores) {
  if (i==1) {
    sumacum<-c(valores[1])
  }else{
    sumacum<- c(sumacum,sumacum[i-1]+valores[i])}
}
```

Ambas formas nos daran: 1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66, 78, 91, 105, 120, 136, 153, 171, 190, 210

3.3 Extra 3

3.3.1

> Escribi una funcion que reciba como input un vector numerico y devuelva los indices donde un numero se repite almenos k veces. Los parametros deben ser el vector, el numero a buscar y la cantidad minima de veces que se debe repetir. Si el numero no se encuentra, retorne un warning y el valor NULL

- Para poder realizar dicho proceso, necesitamos definir la funcion.

- Luego iteramosen un for que va de 1 al largo - la cantidad de repeticiones+1
- Alli un if vera si existe dicha repeticion en algun punto del vector y guarda el indice .
- Luego tenemos un if que ve si el vector de indices esta vacio, de estarlo envia un warning null

```
aparece <- function(v,n,r){
  ap<-c()
  for (i in 1:length(v)-r+1) {
    if((v[i]==n) && (v[i:i+r-1]==n) ){
        ap<-c(ap,i)
    }
  }

  if(length(ap)==0){
    print(paste("WARNING",ap<-c())) }
 return(ap)
}
```

```
vec<- c(3, 1, 2, 3, 3, 3, 5, 5, 3, 3, 0, 0, 9, 3, 3, 3)
aparece(vec,3,3)
```

Ejemplo utilizando c(3, 1, 2, 3, 3, 3, 5, 5, 3, 3, 0, 0, 9, 3, 3, 3), con 3 repeticioens del numero 3

```
## [1]  4 14
```

3.4 Extra 4

Dado el siguiente factor

```
f1 <- factor(letters)
```

3.4.1 ¿Que hace el siguiente codigo? Explica las diferencias o semejanzas

```
levels(f1) <- rev(levels(f1))
f2 <- rev(factor(letters))
f3 <- factor(letters, levels = rev(letters))
```

- 1. La primera parte de la funcion voltea la funcion f1 entera
- 2. La segunda parte de la funcion voltea el factor entero sin voltear los niveles
- 3. La tercera parte de la funcion genera un factor con los niveles volteados