Tarea 1

Lorena Pérez 49264899

Entrega 7 de Mayo

La fecha para entregar la Tarea 1 es el Viernes 7 de Mayo 23:59 PM. La tarea es individual por lo que cada uno tiene que escribir su propia versión de la mism aunque se incentiva la consulta de dudas con estudiantes del curso así como en foros de EVA.

La tarea debe ser realizada en RMarkdown disponible en tu repositorio de GitHub llamado **Tareas_STAT_NT** (generado en la Actividad 5) donde van a ir poniendo todas las tareas y actividades del curso en diferentes carpetas y cada uno con su correspondiente proyecto de RStudio.

En el **YAML** del .Rmd incluí tu nombre y CI (cambiar donde dice author: "STAT NT").

MUY IMPORTANTE Cuando generen el proyecto de RStudio siempre revisar que están utilizando la codificación de texto UTF-8 (text encoding UTF8). Para ello, debes ir a:

Tools -> Project Options -> Code Editing -> Text encoding seleccionar UTF-8.

El repositorio de GitHub para esta tarea debe contener el únicamente archivo .Rmd con la solución de la tarea 1 (en la carpeta correspondiente).

Para que podamos ver sus tareas y corregir las mismas nos tienen que hacer colaboradores de su repositorio de GitHub a Federico (fedemolina) y a Natalia (natydasilva).

Utilicen el archivo .Rmd de esta tarea como base para la solución, incorporando debajo de la pregunta su respuesta. Comienzá con los ejercicios más sencillos y intentá ser ordenado/a, enumerá los ejercicios y **utilizá un archivo .Rmd el cual debe compilar a .pdf mostrando el código (chunks), o sea echo = TRUE**.

Verán que tal vez algunos ejercicios tienen alguna dificultad adicional que las actividades, se espera que revisando el material sugerido en el curso y leyendo la ayuda en R deberían ser capaces de resolver los problemas. Si las preguntas no son suficientemente claras, pregunten en el foro de EVA. Si las dudas no son de comprensión de la letra se aconseja primero **buscar por su cuenta inicialmente** ya que es parte del aprendizaje.

Si un ejercicio no lo pudiste realizar pero intentaste diferentes formas **no** dejes en blanco el ejercicio, mantené el código junto al razonamiento que utilizaste.

1. Ejercicio 1

1.1. Parte 1: Vectores

1.1.1. Dado los siguientes vectores, indicá a qué tipo de vector 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)
```

```
typeof(w) #character
typeof(x) #character
typeof(y) #integer
typeof(z) #character
```

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

Aunque ambos son un vector (verificación mediante is.vector), el primero es de tipo numérico mientras que el segundo es integer (is.vector)

Comentario: Correcto

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 A
```

1.2.1. Generá un nuevo `factor` (llamalo `xx`) transformando el objeto `x` previamente generado de forma que quede como sigue:

```
xx
```

```
[1] A B M A A B M A A M B A B B A
```

```
Levels: B < M < A
```

```
unique(x)
```

```
##  [1] alto     bajo     medio    muy alto ALTO     MEDIO    BAJO     MUY ALTO
##  [9] QUE LOCO A        B        C        GUAU     GOL      MUY BAJO
## 15 Levels: A alto ALTO B bajo BAJO C GOL GUAU medio MEDIO muy alto ... QUE LOCO
```

```
dejar<-c("alto", "bajo","medio","muy alto","ALTO","MEDIO","BAJO","MUY ALTO", "MUY BAJO")
xx<-x[which(x%in%dejar)]
xx<-factor(xx,levels=c("bajo","medio","alto","muy alto","ALTO","MEDIO","BAJO","MUY ALTO","MUY BAJO"))
xxx<-c()
for (i in 1:length(xx)){
  if (xx[i]%in%c("ALTO","MUY ALTO", "muy alto")){
    xxx[i]="alto"
  } else if (xx[i]%in%c("BAJO","MUY BAJO")){
    xxx[i]="bajo"
  } else if (xx[i]=="MEDIO"){
    xxx[i]="medio"
  } #else {xxx[i]=xx[i]
}
xx<-factor(xxx,levels=c("bajo","medio","alto"),labels=c("B","M","A"))
xx
```

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

*Observación**:

- El largo es de 23.

- Se deben corregir (y tomar en cuenta) todos los casos que contengan las palabras: bajo, medio, alto. Es decir, "MUY ALTO", "ALTO" deben transformarse a "alto" y así sucesivamente.

1.2.2. Generá el siguiente `data.frame()`

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

```
dataf<- as.data.frame(xx)
```

Comentario: Buen uso de funciones pero los resultados son incorrectos, revisar solución.

1.3. Parte 2: Listas

1.3.1. Generá 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(1:4)
u<-matrix(c(1:12),nrow=4,ncol=3)
palabra<-"chau"
fecha<-seq(as.Date("2021/1/1"), as.Date("2021/12/30"), by="days")

lista_t1<-list(h,u,palabra,fecha)
```

1.3.2. ¿Cuál es el tercer elemento de la primera fila de la matriz u? ¿Qué columna lo contiene?

```
lista_t1[[2]][1,3] #El elemento es el 9 y se encuentra en la tercera columna
```

```
## [1] 9
```

Comentario: Correcto

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

La primer opción asigna el valor 0 a cada elemento del segundo elemento de la lista, es decir a la matriz u. La segunda por el contrario asigna el valor cero al segundo elemento de la lista, es decir este deja de ser la matriz original u y ahora es cero.

Comentario: Correcto

1.3.4. Iteración

Iterá sobre la 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`.

```
lista_t2<-list()
vect<-c()
for (i in 1:length(lista_t1)){
  for (j in 1:length(lista_t1[[i]])){
    clase<-class(lista_t1[[i]])
```

```
    if (length(class(lista_t1[[i]]))>1){
      lista_t2[[i]]<-tail(clase,1)
      vect[i]<-tail(clase,1)
    }
  }
}
vect
```

```
## [1] NA      "array"
```

Pista: Revisá la familia de funciones `apply`.

Comentario: Incorrecto

1.3.5. Iteración (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.

```
ultimas10<-tail(lista_t1[[4]],10)
hoyes<-c()
for (i in 1:length(ultimas10)){
  hoyes[i]<-paste("La fecha en este momento es",as.character(ultimas10[i]))
}
hoyes2<-paste("La fecha en este momento es",ultimas10)
```

Comentario: Correcto

1.4. Parte 3: Matrices

1.4.1.
Generá una matriz A de dimensión $4 * 3$ y una matriz B de dimensión $4 * 2$ con números aleatorios usando alguna función predefinda en R.

```
A<-matrix(sample(1:12),nrow=4,ncol=3)
B<-matrix(sample(1:8),nrow=4,ncol=2)
```

Comentario: Correcto

1.4.2.
Calculá el producto elemento a elemento de la primera columna de la matriz A por la última columna de la matriz B.

```
prodelem<-A[,1]*B[,ncol(B)]
```

Comentario: Correcto

1.4.3.
Calculá el producto matricial entre $D = A^T B$. Luego seleccioná los elementos de la primer y tercera fila de la segunda columna (en un paso).

```
pto = function(M1, M2){
  prod = matrix(0 , dim(M1)[1] , dim(M2)[2] , TRUE )
```

```
  for ( i in 1:dim(M1)[1] ) {
    for ( j in 1:dim(M2)[2] ) {
      for (k in 1:dim(M1)[2] ) {
        prod[i,j] = prod[i,j] + M1[i,k]*M2[k,j]
      }
    }
  }
  return (prod)
}
pto(t(A),B)
```

```
##      [,1] [,2]
## [1,]  109  182
## [2,]   72  157
## [3,]   59  137
```

```
D=t(A)%*%B
seleccionar<-cbind(D[1,2],D[3,2])
```

Comentario: Correcto

1.4.4. Usá las matrices A y B de forma tal de lograr una matriz C de dimensión $4 * 5$. Con la función `attributes` inspeccioná los atributos de C. Posteriormente renombrá filas y columnas como "fila_1", "fila_2"... "columna_1", "columna_2", vuelvé a inspeccionar los atributos. Finalmente, generalizá y escribí una función que reciba como argumento una matriz y devuelva como resultado la misma matriz con columnas y filas con nombres.

```
C<-cbind(A,B)
attributes(C)
```

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

```
rownames(C) <- c("fila_1", "fila_2","fila_3","fila_4")
colnames(C)<-c("columna_1","columna_2","columna_3","columna_4","columna_5")

nombrarmatriz = function(M1){
  filas<-c()
  columnas<-c()
  for (i in 1:nrow(M1)){
    filas[i]<-paste("fila_",i)}
  for (j in 1:ncol(M1)){
    columnas[j]<-paste("columna_",j)}
  rownames(M1)<-filas
  colnames(M1)<-columnas
  return (M1)
}

C<-cbind(A,B)
nombrarmatriz(C)
```

```
##         columna_ 1 columna_ 2 columna_ 3 columna_ 4 columna_ 5
## fila_ 1          2          4          7          1          4
## fila_ 2          9          5          1          7          6
```

```
## fila_ 3          8          3         11          3          5
## fila_ 4         10         12          6          2          8
```

Comentario: Muy bien!

1.4.5. Puntos Extra: genelarizá la función para que funcione con arrays de forma que renombre filas, columnas y matrices.

2. Ejercicio 2

2.1. Parte 1: `ifelse()`

2.1.1. ¿Qué hace la función `ifelse()` del paquete `base` de R?

La función tiene la misma lógica que usar if y else, pero en una misma línea de código. Requiere 3 argumentos: la condición a evaluar, resultado si dicha condición se cumple, y resultado si la misma no se cumple.

Comentario: Correcto

2.1.2. Dado el vector x tal que: `x <- c(8, 6, 22, 1, 0, -2, -45)`, utilizando la función `ifelse()` del paquete `base`, reemplazá 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>0,1,0)
```

Comentario: Correcto

3. ¿Por qué no fué necesario usar un loop ?

Porque la función ifelse recorre todo el objeto implícitamente, en este caso x, por lo tanto no hay que especificarle los i a evaluar.
Comentario: Correcto, es vectorizada

3.1. Parte 2: `while()` loops

3.1.1. ¿Qué es un while loop y cómo es la estructura para generar uno en R? ¿En qué se diferencia de un for loop?

El while es un método de iteración al igual que el for e if. La principal diferencia es que el while requiere una condición de parada. Es decir la función seguirá iterando siempre y cuando esa condición no sea alcanzada. En cambio en el for previamente tenemos que definir todos los elementos a evaluar.

while (condición) { acción }
Comentario: Correcto

3.1.2. Dada la estructura siguiente, ¿Cuál es el valor del objeto `suma`? Responda sin realizar el cálculo en `R`.

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

```
  i <- i + 1
}
```

0+1+2+3+1+2+3=12
Comentario: Incorrecto es NA

3.1.3. Modificá 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. A partir de ella generá una fución que se llame `sumar_si` y verificá que funcione utilizando los vectores `y <- c(1:3)`, `z <- c(1:15)`.

```
x <- c(1,2,3)

suma <- 0
if (length(x)<5){
  suma=0
} else {
  for (i in 1:5){
    suma=suma+x[i]
  }
}

sumar_si<-function(v){
  suma <- 0
  if (length(v)<5){
    suma=0
  } else {
    for (i in 1:5){
      suma=suma+v[i]
    }
  }
  return(suma)
}

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

Comentario: Correcto

3.1.4. Generá una estructura que multiplique los números naturales (empezando por el 1) hasta que dicha multiplicación supere el valor `10000`. Cuánto vale dicha productoria?

```
productoria <- 1
numero <- 0
while( productoria<=10000){
  numero=numero+1
  productoria = productoria*numero
}
productoria
```

Comentario: Es numero en vez de i. Estaba mal, pero cambiando eso esta ok.

3.2. Parte 3: Ordenar

3.2.1. Generá una función `ordenar_x()`que para cualquier vector numérico, ordene sus elementos de menor a mayor. Por ejemplo:

Sea `x <- c(3,4,5,-2,1)`, `ordenar_x(x)` devuelve `c(-2,1,3,4,5)`.

Para controlar, generá dos vectores numéricos cualquiera y pasalos como argumentos en `ordenar_x()`.

Observación: Si usa la función `base::order()` entonces debe escribir 2 funciones. Una usando `base::order()` y otra sin usarla.

```
ordenar_x<-function(a){
  aux<-c()
  j<-1
while (j<length(a)){
for (i in 1:(length(a)-1)){
  if (a[i]>a[i+1]){
    aux[i]<-a[i]
    a[i]<-a[i+1]
    a[i+1]<-aux[i]
    }
  }
  j<-j+1}
  a
}

prueba<-c((sample(1:5)))
prueba2<-c((sample(3:10)))
ordenar_x(prueba)
ordenar_x(prueba2)
```

Comentario: Correcto, agregar set.seed

3.2.2. ¿Qué devuelve `order(order(x))`?

```
x <- c(3,4,5,-2,1)
#La función order me devuelve la posición en la que se encuentran los elementos ordenados, por default
order(order(x))
```

Comentario: Correcto

4. Ejercicios Extra

Esta parte es opcional pero de hacerla tendrán puntos extra.

4.1. Extra 1

4.1.1. ¿Qué función del paquete base es la que tiene mayor cantidad de argumentos?

Pistas: Posible solución:

0. Argumentos = `formals()`
1. Para comenzar use `ls("package:base")` y luego revise la función `get()` y `mget()` (use esta última, necesita modificar un parámetro ó formals).

2. Revise la funcion Filter
3. Itere
4. Obtenga el índice de valor máximo

4.2. Extra 2

Dado el siguiente vector:

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

4.2.1. Obtené la suma acumulada, es decir 1, 3, 6, 10... de dos formas y que una de ellas sea utilizando la función `Reduce`.

Dados los siguientes data.frame

```
a = data.frame(a1 = 1:10,
               b1 = 1:10,
               c1 = 1:10,
               key = 1:10)
b = data.frame(d1 = 1:10,
               e1 = 1:10,
               f1 = 1:10,
               key = 1:10)
c = data.frame(g1 = 1:10,
               h1 = 1:10,
               i1 = 1:10,
               key = 1:10)
```

Uní en un solo data.frame usando la función `Reduce()`. Pista: Revisá la ayuda de la función `merge()` y buscá en material adicional si es necesario que es un join/merge.

4.3. Extra 3

4.3.1. Escribí una función que reciba como input un vector númerico y devuelva los índices donde un número se repite al menos k veces. Los parámetros deben ser el vector, el número a buscar y la cantidad mínima de veces que se debe repetir. Si el número no se encuentra, retorne un `warning` y el valor `NULL`.

A modo de ejemplo, pruebe con el vector `c(3, 1, 2, 3, 3, 3, 5, 5, 3, 3, 0, 0, 9, 3, 3, 3)`, buscando el número 3 al menos 3 veces. Los índices que debería obtener son 4 y 14.

4.4. Extra 4

Dado el siguiente `factor`

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

4.4.1. ¿Qué hace el siguiente código? Explicá las diferencias o semejanzas.

```
levels(f1) <- rev(levels(f1))
#rev me cambia el orden a la inversa del objeto f1, y lo que hace es asignar este nuevo orden a los niv
f2 <- rev(factor(letters))
#Se crea un nuevo objeto, f2, en el que los levels son iguales a los del f1 original a:z, la diferencia
f3 <- factor(letters, levels = rev(letters))
#En este caso los elementos del factor f3 son a:z, tal como se definen originalmente. La función rev se
#En conclusión, f1 y f2 coinciden en sus elementos mientras que difieren en sus levels. f1 y f3 coincid
```