Tarea 1

Ciencia de Datos con R

Ivan Arriola 55366796

Entrega 12/4 23:59 PM

La fecha para entregar la Tarea 1 es el 12 de Abril a las 23:59 PM. La tarea es individual por lo que cada uno tiene que escribir su propia versión de la misma 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 (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: "Alumno").

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 Mauro (maurolopretе) 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.

Ejercicio 1: Vectores

1. Dado los siguientes vectores, indicá a qué tipo de vector coercionan.

```
w <- c(29, 1L, "22020202", "1")
cat("Cooerciona a", typeof(w))
```

```
## Cooerciona a character
```

```
m <- c("22+3", 33, NA)
cat("Cooerciona a", typeof(m))
```

```
## Cooerciona a character
```

```
y <- c(seq(3:25), 10L)
cat("Cooerciona a", typeof(y))
```

```
## Cooerciona a integer
```

```
z <- paste(seq(3:25), 10L)
cat("Cooerciona a", typeof(z))
```

```
## Cooerciona a character
```

2. Almacená los vectores de la parte 1 en una lista.

```
lista <- list(w, m, y, z)
```

3. Creá una función que intente obtener el total del vector. En el caso que no pueda cohercionar a vector númerico, la función debe de contar la cantidad de elementos diferentes y de un mensaje de advertencia.

```
total <- function(x){
  if (is.numeric(x)) {
    return(sum(x))
  } else {
    newX <- as.numeric(x)
    if (!any(is.na(newX))) {
      return(sum(newX))
    } else {
      elemDif <- length(unique(x))
      warning(sprintf("No se pudo obtener el total del vector. Se encontraron %d valores distintos.", el
      return(NA)
    }
  }
}
```

4. Evalua la función anterior en cada elemento de la lista llamando una única vez la función creada en la sección anterior.

Pudes usar un `for loop` pero en este caso probá hacerlo de una forma alternativa con la familia de funciones `apply`

```
sapply(lista, total)
```

```
## [1] 22020233       NA      286       NA
```

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

```
c(4, 3, 2, 1)
```

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

```
4:1
```

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

Crean exactamente el mismo vector. Las formas de crearlo son distintas, en el primero se explicito cada uno de los elementos, mientras que el segundo se creo como una secuencia que va de 4 a 1 , con pasos de -1

Ejercicio 2: Factores

1. Describa brevemente la utilidad de trabajar con factores y los posibles inconvenientes vistos en clase.

Dado el siguiente `factor` `x`:

```
x <-
   factor(
      c(
         "a lto",
         " b a j o ",
         "med i          o",
         "alt o",
         "muy alto",
         "bajo",
         "medio",
         "alto",
         "ALTO ",
         "  MEDIO",
         "BAJO  ",
         "MUY_ ALTO ",
         "muy muy muy alto",
         "no se que contestar",
         "Alto",
         "Ni idea",
         " muy alto ",
         "alto, creo",
         "Bajo ",
         "_______M ______",
         "GU    AU    ",
         "GOL",
         "MUY BAJO!!!",
         "MUY BAJO                          ",
         "MuY AlTo",
         "Me parece que alto",
         "demasiado bajo"
      )
   )
```

2. Generá un nuevo `factor` (llamalo `xx`) transformando el objeto `x`:

Debes de seguir los siguientes pasos:

- Conviertan todos los valores a minúsculas mediante una función de R, NO lo hagan a mano.
- Borren "muy" para que "bajo" y "muy bajo" sean iguales, pueden existir espacios, hay que removerlos.
- Todo lo que no sea "bajo", "medio", "alto" debe ser excluido.
- Las etiquetas deben ser : B bajo, M medio y A alto.
- Recuerden (como vieron en las actividades) que pueden usar *labels*.

El ejercicio tiene dos respuestas, una refleja un nivel básico-medio de manipulación de cadenas de texto y el otro un nivel medio-avanzado. El nivel básico-medio implica no poder extraer los niveles de `'Me parece que alto'`, `'demasiado bajo'` y `'alto,creo'`, el siguiente nivel implica poder extraer los niveles `alto`, `bajo` y `alto` respectivamente.

Si querés pasar de un nivel a otro podes buscar sobre regex [1], la función `base::regexpr` y `base::regmatches`.

[1] Visitá https://regex101.com/

Pistas:

- Para el nivel básico-medio se obtiene la siguiente tabla de frecuencias:
  - alto: 10
  - medio: 3
  - bajo: 6
- Para el nivel avanzado, se le agregan al nivel alto dos cantidades y al bajo una observación.
- 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.

```
x_min <- tolower(x)
x_sin_espacios <- gsub(' ', '', x_min)

x_bajo <- grepl("bajo", x_sin_espacios)
x_medio <- grepl("medio", x_sin_espacios)
x_alto <- grepl("alto", x_sin_espacios)

x[x_bajo] <- c('bajo')
x[x_medio] <- c('medio')
x[x_alto] <- c('alto')
x[!(x_bajo | x_medio | x_alto)] <- c(NA)


x_labels <- c('A', 'M', 'B')
x_levels <- c('alto', 'medio', 'bajo')

xx <- factor(x, levels = x_levels , labels = x_labels)
```

3. Generá el siguiente `data.frame()`

```
df <- as.data.frame(table(xx))
names(df) <- c("levels", "value")
df[c(2, 3),] <- df[c(3, 2),]
df
```

```
##   levels value
## 1      A    12
## 2      B     7
## 3      M     3
```

Ejercicio 3: Listas

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, 2, 3, 4)
u <- matrix(1:12, nrow = 4, ncol = 3)
palabra <- "chau"
fecha <- seq(as.Date("2021-01-01"), as.Date("2021-12-30"), by = "day")
lista_t1 <- list(h, u, palabra, fecha)
```

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

```
primera_fila <- u[1,]
primera_fila[3] # esta en la tercera columna
```

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

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

El primero reemplaza los valores de la matriz `lista_t1[[2]]`, el segundo reemplaza `lista_t1[[2]]` con 0

Ejercicio 4: Matrices

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(1:100, 12), nrow = 4, ncol = 3)
B <- matrix(sample(1:100, 8), nrow = 4, ncol = 2)
```

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

```
A[,1] * B[,-1]
```

```
## [1]   33  344 1408  910
```

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).

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

```
## [1]  2695 15121
```

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) <- paste0("fila_", 1:4)
  colnames(C) <- paste0("columna_", 1:5)
  attributes(C)
```

```
## $dim
## [1] 4 5
##
## $dimnames
## $dimnames[[1]]
## [1] "fila_1" "fila_2" "fila_3" "fila_4"
##
## $dimnames[[2]]
## [1] "columna_1" "columna_2" "columna_3" "columna_4" "columna_5"
```

```
    renombrar_matriz <- function(matriz) {
      rownames(matriz) <- paste0("fila_", 1:nrow(matriz))
      colnames(matriz) <- paste0("columna_", 1:ncol(matriz))
      return(matriz)
    }
    renombrar_matriz(C)
```

```
##        columna_1 columna_2 columna_3 columna_4 columna_5
## fila_1        33        27         6        92         1
## fila_2        86        34        88        95         4
## fila_3        16        26        84        65        88
## fila_4        10        94        81        59        91
```

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

Ejercicio 5: Estructuras de control y funciones vectrorizadas

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

Dada una condicion, cuando es TRUE devuelve un valor, y cuando es FALSE devuelve otro. Es vectorizada, por lo que si la condicion es vectorizada, la respuesta va a ser un vector con la respuesta de cada elemento del vector

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)
ifelse(x > 0, 1, 0)
```

```
## [1] 1 1 1 1 0 0 0
```

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

Porque la función `ifelse()` es vectorizada

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

Un while loop es una estructura de control que sirve para iterar mientras una condición sea verdadera. La estructura es la siguiente:

```
while(condicion){
  # codigo a ejecutar
}
```

5. Dada la estructura siguiente:

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

6. 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 función que se llame `sumar_si` y verificá que funcione utilizando los vectores `y <- c(1:3)`, `z <- c(1:15)`.

```
sumar_si <- function(x) {

  if(length(x) < 5) return(0)

  suma <- 0
  i <- 1
  while(i < 6 ){
    suma = suma + x[i]
    i <- i + 1
  }
  return(suma)
```

```
}

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

sumar_si(y)
```

```
## [1] 0
```

```
sumar_si(z)
```

```
## [1] 15
```

7. Intentá escribir en función de forma vectorizada.

```
sumar_si_vectorizado <- function(x) {
  if(length(x) < 5) return(0)
  return(sum(x[1:5]))
}

sumar_si_vectorizado(y)
```

```
## [1] 0
```

```
sumar_si_vectorizado(z)
```

```
## [1] 15
```

Probá que la función devuelva el mismo resultado usando `all.equal`.

```
all.equal(sumar_si(y), sumar_si_vectorizado(y))
```

```
## [1] TRUE
```

```
all.equal(sumar_si(z), sumar_si_vectorizado(z))
```

```
## [1] TRUE
```

Ejercicio 6: Ordenar

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(x) {
  if (length(x) <= 1) return(x)
  else {
    pivot <- x[1]
    left <- x[-1][x[-1] <= pivot]
    right <- x[-1][x[-1] > pivot]
    return(c(ordenar_x(left), pivot, ordenar_x(right)))
  }
}
```

```
ordenar_x2 <- function(x) {
  return(x[order(x)])
}

x <- c(3,4,5,-2,1)
y <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
ordenar_x(x)
```

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

```
ordenar_x2(x)
```

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

```
ordenar_x(y)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
ordenar_x2(y)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

2. ¿Qué devuelve `order(order(x))`?

`order(order(x))` devuelve el lugar que ocuparia cada elemento de x si estuviera ordenado de menor a mayor

Ejercicio 7: Argumentos de funciones

- ¿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 del valor máximo

```
  lista_base <- ls("package:base")
  cant_arg <- sapply(lista_base, function(name_f){
    fun <- get(name_f)
    if(is.function(fun)) {
      return(length(formals(fun)))
    } else {
      print(name_f)
      return(NA)
    }
  })
```

```
## [1] "F"
## [1] "letters"
## [1] "LETTERS"
## [1] "month.abb"
## [1] "month.name"
## [1] "pi"
## [1] "R.version"
## [1] "R.version.string"
## [1] "T"
## [1] "version"
```

```
  indice_max <- which.max(cant_arg)
  lista_base[indice_max]
```

```
## [1] "scan"
```

Ejercicio 8: Factores

Dado el siguiente `factor`

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

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

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

El primero invierte el orden de los niveles del factor, no modifica el orden de los elementos del vector. El segundo invierte el orden de los elementos del vector, no modifica el orden de los niveles del factor. El tercero invierte el orden de los niveles del factor y el orden de los elementos del vector.