Redacción de artículos académicos con R

Episodio 2: Estructuras y primeras operaciones

Bajaña Alex

Chanatasig Evelyn

Heredia Aracely

2022-06-04

Episodio 2: Estructuras y primeras operaciones

En la clase anterior

Vectores en el `environment`

Como mencionamos en la clase anterior el `.globalEnvironment` es el espacio virtual donde se almacenan los vectores y cualquier otro objeto que creemos en nuestro análisis. Es así como si creamos un vector que coleccione los nombres de los encuestados en la ENEMDU, el resultado de la función `ls()` ahora devolverá los nombre de los elementos creados.

```
nombres <- c("Mario","Juan","Paul",
             "Pepe","Camila","Andrés")    #vector de nombres
ls()
```

```
## [1] "nombres"
```

Recuerda que cuando llamamos a la función `library()` estamos expandiendo los elementos que interactúan en nuestro análisis. Por ejemplo si queremos observar que funciones pertenecen a la libreria `base` utilizamos:

```
# algunas de las varias funciones en esta libreria

ls("package:base")
```

```
## [1] "as.logical"          "as.logical.factor"    "as.matrix"          "as.matrix.data.frame"
```

Los elementos de un vector

En un vector nos importa el *orden* de sus elementos ya que cada elemento que ocupa una *posición* representa una persona, evento, o unidad de análisis. Los vectores manejan un **índice** que va desde 1 hasta el número de elementos que almacenemos en su interior. Este índice puedetambién ser un texto para el caso de *vectores nombrados*.

Ya sea por su indice y por el nombre del elemento dentro del vector emplea el operador corchetes `[]` y uno o más indices para extraer sus elementos.

Un solo elemento:

```
nombres[1]            #extraer elemento de la posición 1
```

```
## [1] "Mario"
```

Para extraer más de un elemento empleamos un vector de posiciones:

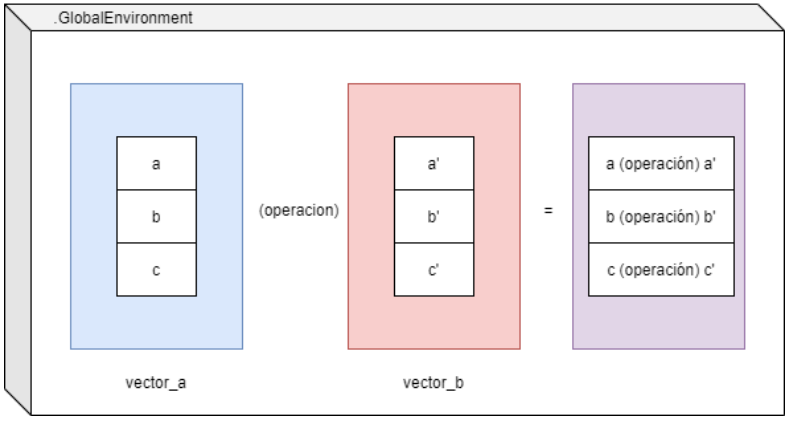
```
nombres[c(3,5)]       #extraer el tercer y quinto elemento
```

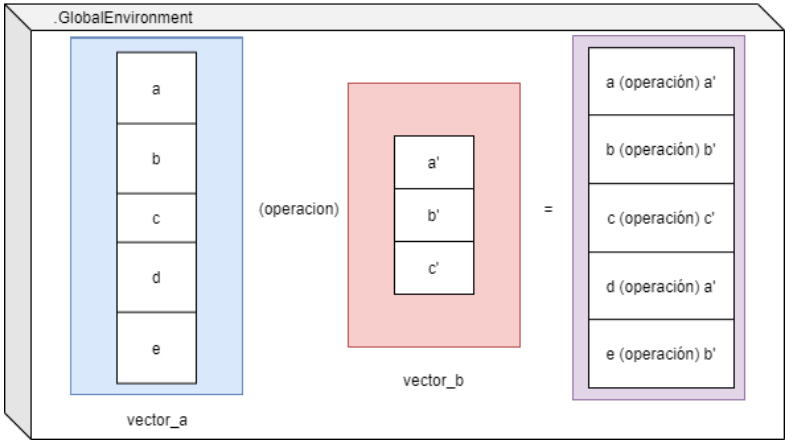
```
## [1] "Paul"   "Camila"
```

¿Como opera R?

- R emplea operaciones arítmeticas de forma vectorial, es decir cuando operamos con vectores de **igual tamaño** la operación se aplica para cada par de elementos de los vectores.
- Los dos vectores deben pertenecer al mismo `environment` para poder interactuar en una operación.

> Cuando se opera con elementos de distinto tamaño, R replica los valores del vector de menor tamaño para hacer posible la operación





Operaciones aritméticas básicas

Al inicio de una sesión de R están incluidas las operaciones aritméticas fundamentales.

Operador	Descripción
+	Adición
-	Sustracción
*	Multiplicación
/	División
^ or **	Exponente
x %% y	Modulo o residuo de la división
x %/% y	Parte entera de la división

Las operaciones aritméticas se realizan entre vectores que tengan el **mismo número de elementos** ó **entre un escalar y un vector**.

```
pob_prov <- c( 4387434,3228233,
               1562079,921763,
               881394)
area_pro <- c(15927,9692,
              19427,7100,
              8189)

pob_prov/area_pro # Vectorial:
```

```
## [1] 275.47146 333.08223  80.40763 129.82577 107.63146
```

```
pob_prov/1e6 # Escalar vs vector
```

```
## [1] 4.387434 3.228233 1.562079 0.921763 0.881394
```

Variables aleatorias, vectores y los resultados de un experimento

Variables aleatorias

- Una variable aleatoria al igual que un vector es una **colección** de elementos.
- Es más, un vector puede ser la representación de una variable aleatoria en el **entorno de programación**.
- También podemos entender a una variable aleatoria como la **asociación** de los resultados de un **experimento** con un valor numérico denominado **medida.**

Proposito de año nuevo

Imaginemos el sigui

Experimentando con R

Al trabajar con datos queremos corroborar hipótesis, comparar eventos o realizar predicciones. En este sentido, `R` nos ofrece funciones que permiten simular eventos aleatorios. Consideremos la función `sample`.

```
sample(c("...","...","..."), size=..., replace=T)
```

- Explicación:
  - `c("...","...","...)`: población dada, puede ser un vector de cualquier clase
  - `size`: tamaño de la muestra deseada
  - `replace = T`: si la muestra deseada es más grande que la población inicial, este argumento permite hacer una muestra con reposición de los elementos en la población

Experimentando con R

```r
# EJEMPLO:
# Crear un vector aleatorio de tipo character:

nivel_instrucc <- sample(c("escuela","colegio","universidad"),
                         size = 100,
                         replace = T)
head(nivel_instrucc)
```

```
## [1] "colegio"     "universidad" "universidad" "colegio"     "escuela"     "colegio"
```

Tabla de frecuencias

- En el vector `nivel_instrucc` simulamos una población de personas a quienes se ha asignado de manera *aleatoria* y *uniforme* una de las tres categorias: "escuela", "colegio", "universidad".
- En este caso observamos cómo se comporta con un vector de caracteres.
- El resumen que generamos con la función `table()` es de clase `table`.

```
resumen_educ <- table(nivel_instrucc)

class(resumen_educ)
```

```
## [1] "table"
```

- R ordena por **orden alfabético** las categorías cuando se trata de vectores de tipo `character`.

Tabla de frecuencias

```
print(resumen_educ)
```

```
## nivel_instrucc
##     colegio     escuela universidad
##          33          30          37
```

Medidas de resumen

- En R estas medidas se calculan empleando `funciones`, los vectores serán los argumentos de estas.
- Una **función** es una serie de operaciones e instrucciones para llegar a un fin.
- Cuando existen **valores perdidos** en el vector, R los muestra como `NA`
- Las operaciones con `NA` dan como resultado `NA`, el argumento `na.rm` se usa para omitir los valores perdidos en el cálculo.

Operador	Descripción
sum(...)	Suma de todos los elementos
cumsum(...)	Suma acumulada
cut(...)	Cortar una variable en intervalos
Dispersión	
max(...)	Valor máximo
min(...)	Valor mínimo
sd(...)	Desviación estandar
quantile(...)	Cuantiles
Tendencia central	
mean(...)	Valor promedio
median(...)	Valor de la mediana

Objetos complejos o S3

Objetos que tienen al menos un atributo de `clase`


factors,
dates,
etc....
integer
numeric
logical
character

Fechas

Fechas

Existe un formato definido de fechas en R, es conocido como `Date`.

- Cuando trabajamos con fechas usualmente necesitamos hacer operaciones con estas.
- Para ello se debe declarar al `vector` o a la `fecha` como tal, mediante el comando `as.Date(...)`

```
as.Date("1970-02-01")


# Fechas declaradas como caracter
c("1970-02-01", "1971-02-01")
fechas_texto <- c("2020-08-20","2020-05-06","2020-04-07")


# Fechas declaradas como fechas
as.Date(c("1970-02-01", "1971-02-01"))    #2 fechas
as.Date(fechas_texto)                     #fechas guardadas en vector
```

Fechas

Por default R busca un formato de fechas en el siguiente orden: `año, mes, día`, separados por un guión.

- En caso de no disponer de este formato, es necesario el argumento adicional `format` en la función `as.Date()`
- Caso contrario el resultado será `NA` o resultados extraños que no corresponden a las fechas reales.

```
# Declaración de fechas
as.Date(c("12-04-2021","25-08-2021","05-01-2021"), format = "%d-%m-%Y")
```

```
## [1] "2021-04-12" "2021-08-25" "2021-01-05"
```

```
# Fechas guardadas en un vector
fechas_2 <- c("12-04-2021","25-08-2021","05-01-2021")
fechas_2 <- as.Date(fechas_2, format = "%d-%m-%Y")
fechas_2
```

```
## [1] "2021-04-12" "2021-08-25" "2021-01-05"
```

Fechas

- Los símbolos para construir el texto a ser tomado por el argumento `format` de la función `as.Date`

Símbolo	Significado
%d	día (numérico, de 0 a 31)
%a	día de la semana abreviado a tres letras
%A	día de la semana (nombre completo)
%m	mes (numérico de 0 a 12)
%b	mes (nombre abreviado a tres letras)
%B	mes (nombre completo)
%y	año (con dos dígitos)
%Y	año (con cuatro dígitos)

Fechas y Horas

Fechas y Horas

Para el manejo de fechas y horas se dispone del formato `POSIXct` el cual se incluyen horas, minutos y segundos.

```
now_ct <-
  as.POSIXct("2018-08-01 22:00",
                     tz = "UTC")
now_ct
```

```
## [1] "2018-08-01 22:00:00 UTC"
```

```
#typeof(now_ct)
attributes(now_ct)
```

```
## $class
## [1] "POSIXct" "POSIXt"
##
## $tzone
## [1] "UTC"
```

- La función a emplear es `as.POSIXct`, así mismo se dispone del argumento `format`.

Símbolo	Significado
%H	Horas
%M	Minutos
%S	Segundos

- En el argumento `tz` podemos definir la zona horaria para realiza transformaciones.

Fechas y Horas

La ventaja del uso de este formato es la transformación de horas de acuerdo al uso horario:

Podemos ver qué hora es en las distintas zonas horarias:

```
structure(now_ct, tzone = "Asia/Tokyo")
```

```
## [1] "2018-08-02 07:00:00 JST"
```

```
structure(now_ct, tzone = "Australia/Lord_Howe")
```

```
## [1] "2018-08-02 08:30:00 +1030"
```

Factores

Factores:

Los **factores** ayudan a darle un **sentido de ordinal a las categorías** de una variable descriptiva, es decir, permiten ordenar los niveles según la lógica de la variable en lugar de ordenar alfabéticamente como lo hace R por default.

Orden en el vector `character`

1. Colegio
2. Escuela
3. Universidad

Orden deseado

1. Escuela
2. Colegio
3. Universidad

Este tipo de elemento es principalmente utilizado cuando disponemos de **variables categóricas** en las que el **orden importa** en cuanto a la interpretación de variables.

- Existen varias formas de crearlos:
  - De caracter a factor
  - De numérico a factor

Cohersión de caracter a factor

- Mediante función `factor()`

```
factor(..., levels = c("...","...","..."))
```

- Explicación:
  - `...`: vector de caracteres que se desea ordenar
  - `levels = c("...","...","...")`: se definen las categorías en el orden adecuado

Ejemplo:

```
factor_nivel_instrucc <- factor(x = nivel_instrucc,
       levels = c(c("escuela","colegio","universidad")))

resumen_f_n <- table(factor_nivel_instrucc)

print(resumen_f_n)
```

```
## factor_nivel_instrucc
##     escuela     colegio universidad
##          30          33          37
```

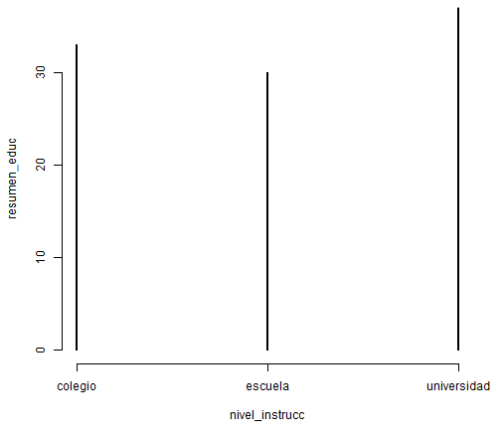
Al final ¿Que hacen los objetos S3?

Vector `character`

```
plot(nivel_instrucc) # Error
```

Primero debemos pasar el vector `character` por la función `table`:

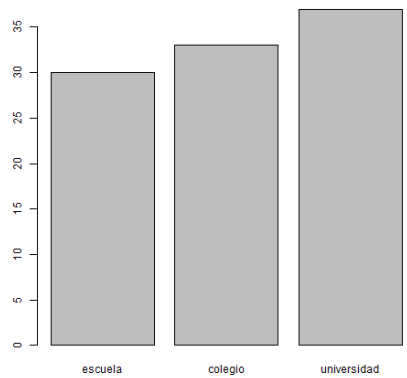
```
plot(resumen_educ)
```



Factor

El factor cambia el comportamiento y resultado de la función `plot`. Los atributos de este nuevo vector permiten omitir la aplicación de `table` para generar el gráfico.

```
plot(factor_nivel_instrucc)
```



Cohersión de numérico a factor

Mediante la función `cut()`. Esta permite pasar de una variable numérica a una de tipo factor entregando los límites que definen cada categorías

```
cut(x=..., breaks = c(...,...,...,Inf),labels = c("...","...","..."))
```

Argumentos de la función `cut()`:

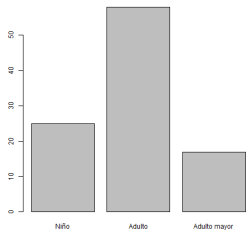
- `x`: es el vector numérico que se desea transformar a categórico mediante límites
- `breaks`: vector indica los límites que definen las categorias
    - `Inf`: define el límite superior .
- `labels=c()`: definen las etiquetas de las categorías definidas por los límites

Ej: Crear grupos de edad a partir de un vector de edades

```
edades <- sample(c(3:81),
                 size = 100,
                 replace = T)   #crear muestra de vector numérico
edad_funcion <- cut(x=edades,
                    breaks = c(3,18,65,Inf), # Limites
                    labels = c("Niño","Adulto","Adulto mayor"))
```

Gráfico

```
plot(edad_funcion)
```



Atributos

```
attributes(edad_funcion)
```

```
## $levels
## [1] "Niño"          "Adulto"        "Adulto
##
## $class
## [1] "factor"
```

Listas y tablas


listas

Operadores lógicos

Operadores lógicos

- Se usan para hacer comparaciones, evaluar condiciones, o conocer casos puntuales que cumplen una serie de parámetros
- Se pueden usar con valores numéricos, caracteres u otro tipo de elemento

```
vector_a >= 3

table(vector_a >= 3)
```

La función `table()` permite hacer un conteo breve de los elementos de un vector.

Operador	Descripción
<	Menor a
<=	Menor o igual a
>	Mayor a
>=	Mayor o igual a
==	Igual
!=	No igual
!x	No x
x | y	x ó y (elemento a elemento)
x || y	x ó y (de vectores)
x & y	'x' y 'y'
isTRUE(x)	Evaluación verdadera

- Esta función es útil para contar los elementos de un vector **lógico**, las **categorías de una variable descriptiva**, o la aparición de textos en un **caracter**. En valores numéricos se usa la función `summary`

Listas

Listas y tablas

El material visto hasta el momento ha sido de **vectores**. Sin embargo, estos solo pueden almacenar elementos de un mismo tipo y que cuando intentamos unir elementos de distinto tipo en un solo vector se aplicarán las **reglas de coerción**. Para dar solución a esta problemática están las listas y tablas

Listas

Una lista en R se define como una colección indexada de cualquier tipo de elemento que se puede generar en R (entre ellos los vectores).

Para crear una lista usamos el comando `list()` y dentro de esta podemos declarar cualquier objeto de R.

Ejemplo de uso de las listas

Metadatos son todas las otras especificaciones que no están dentro de la base de datos pero son necesarias para entender cómo fue construída esta. **Ejemplo:** nombres de las variables, fecha de descarga, tamaño de la base de datos, formato de la base.

```
metadatos <- list("35 mb",           #caracter: tamaño de la base
                  1:200,             #enteros: índice de filas
                  c(TRUE, FALSE),    #lógico
                  (1:4)/2,           #numérico decimal
                  as.Date("03/07/2020",
                          format="%d/%m/%y"))    #fecha

metadatos

str(metadatos)         #Muestra los elementos que tiene
```

> En este caso, hemos almacenado elementos de distintos tipos y tamaños en una lista metadatos pero no hemos definido un nombre específico para cada elemento.

Listas

En la lista que hemos llamado `metadatos` vemos una colección de 5 vectores. El primero es de tipo `caracter`, el segundos de tipo `entero`, el tercero de tipo `lógico`, el cuarto de tipo `numérico` y el quinto es de tipo `Date`.

Listas nombradas:

En una lista podemos asignar nombres a los elementos que la componen. Esto permite un acceso más fácil a los elementos de una lista.

```
metadatos2 <- list(`tamaño de la base`="35 mb",
                   `fecha descarga`=as.Date("03/07/2020",
                          format="%d/%m/%y"))

names(metadatos2)              #Muestra los nombres de (...)

attributes(metadatos2)
```

Extraer los elementos de un vector que esté dentro de la lista

Se aplica una lógica similar a la extracción de elementos de un vector al utilizar una posición o un vector de posiciones.

- Si se conoce la **posición** del vector deseado se usa el **doble corchete** `[[]]`.

```
metadatos2[[1]]
```

- Si se conoce el nombre del vector deseado se usa el **símbolo de dólar** `$`

```
metadatos2$`tamaño de la base`
```

Extraer un elemento como lista

Utilizo un sólo corchete `[]`

```
metadatos2[1]
```

Tablas o data.frames

Tablas o data.frames

En R las tablas se conocen como `data.frames`. De manera específica, son **listas** con nombres que albergan vectores de un **mismo tamaño** y que entre todos componen una **tabla de datos** con un tema específico.

En los **data.frames** las **filas u observaciones** corresponden a una unidad (personas, unidades de salud, carros, plantas) y las **columnas o variables** constituyen características de estas unidades.

```
head(iris, 2)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
```

```
class(iris)
```

```
## [1] "data.frame"
```

> `iris` es un `data.frame` que viene por default en el environment de R. Úsalo para hacer tus pruebas de código.

Ejemplo de un data.frame

Los objetos de clase `data.frame` tienen atributos importantes. Por ejemplo un `data.frame` tiene **dimensiones** (filas y columnas que se pueden contabilizar con la función `dim()`).

```
df <- data.frame(x = 1:3,
                 y = c("a", "b", "c"))
```

```
attributes(df)
```

```
## $names
## [1] "x" "y"
##
## $class
## [1] "data.frame"
##
## $row.names
## [1] 1 2 3
```

```
dim(df)
```

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

Extracción de elementos de una tabla

Dado que el `data.frame` o `tabla` tiene dimenciones, para la extracción de elementos se necesita de dos indicadores: fila y columna, de la forma:

```
tabla[indicador_fila, indicador_columna]

# Ejemplo 1: extracción del elemento que está en la primero fila y segunda columna de la t
df[1,2]

# Ejemplo 2: extracción de la tercera fila de la tabla:
df[3,]

# Ejemplo 3: extracción de la segunda columna de la tabla:
df[,2]
```

- **Ej 2:** El contador de columnas está **vacio**, esto indica que queremos la tercera fila y todas las columnas de la tabla.
- **Ej:3:** El contador de filas está **vacio**, esto indica que queremos la segunda columna y todas las filas de la tabla.

Formas de extracción

- Mediante la posición

```
df[,2]
```

- Usando el nombre del vector

```
df[,"y"]
```

- Usando el símbolo:$

```
df$y
```

- Usando dos corchetes [[]]
  - Mediante la posición
  - Usando el nombre del vector

```
df[[2]]        #este sirve sólo para columnas
df[["y"]]
```

Extracción de varias columnas y filas

Cuando extraemos más de un elemento se debe usar la función concatenar `c(...,...,...)`. La estructura es la siguiente:

```
tabla[c(conjunto de filas), c(conjunto de columnas)]
```

En este caso, también existen varias formas de extracción:

- Mediante la posición

```
df[c(1,3),c(1,2)]
```

- Usando el nombre del vector

```
df[,c("x","y")]
```

- Usando una mezcla de los dos anteriores

```
df[c(1,3),c("x","y")]
```

Creación de nuevas variables en una tabla

Para generar una columna adicional a la tabla se usa el símbolo $ seguido por el nombre de la columna que deseamos agregar

Las formas de extracción de variables (como el $), cuando están en la parte izquierda de la asignación <- sirven para crear una nueva variable. Lo que está en la parte derecha de <- es la información que se va a guardar en esa columna.

```
df$nueva_columna <- 5+8
```

También se puede operar con los vectores que son elementos de la tabla.

```
#suma de columna "x" y "nueva columna"
df$nueva_columna_2 <- df$x + df$nueva_columna
```

Filtro de una tabla

Usualmente queremos crear tablas que cumplan con alguna condición, para ello se combina las operaciones lógicas.

Para ejemplificar este tema se va a crear una nueva tabla o data frame que la llamaremos nuestra base de datos, esta es una muestra con 20 individuos de ambos sexos, con edades de 18 a 65 y con un peso de entre 45 y 80 kilogramos.

```r
#Primero creamos una tabla (para poder filtrar)
base_datos <- data.frame(
  sexo = sample(c("Hombre","Mujer"),size = 20,replace = T),
  edad = sample(18:65,size = 20,replace = T),
  peso = sample(45:80,size = 20,replace = T))
```

Filtro de una tabla

Para filtrar la base por un argumento:

- Se llama a la base de datos y se abre corchetes `[]`
- Se escribre el argumento o filtro en la parte de **filas**, mientras que la sección de columnas se deja en blanco
- Para escribir el argumento, primero se debe escribir la **variable que se desea filtrar**, a continuación se escribe el **operador lógico** (`==, >, <`) y finalmente se escribe el **argumento deseado** (si es caracter se pone entre paréntesis)

```
# Filtrar la base para que me queden los datos sólo de las mujeres
base_datos[base_datos$sexo=="Mujer",]
```

Para filtrar una base usando varias condiciones se debe usar el símbolo `&` para unir todos los filtros.

```
#Filtrar para hombres mayores de 55 años y peso menor a 60 kilos:
base_datos[base_datos$sexo=="Hombre" & base_datos$edad>25 & base_datos$peso<75,]
```


THE
END