Capítulo 11: Vectores no disponibles

En *R* se manejan los datos no disponibles de distintas maneras, dependiendo del dato, principalmente se manejan a través del `NA` (NOT AVAILABLE).

Algunas formas de trabajar con datos no disponibles en R incluyen:

1. **Identificar datos faltantes**: Puedes usar funciones como `is.na()` para identificar si un valor es `NA`.
2. **Eliminar datos faltantes**: Puedes utilizar funciones como `na.omit()` para eliminar filas con valores faltantes en un marco de datos.
3. **Reemplazar datos faltantes**: Puedes reemplazar los valores faltantes con un valor específico usando la función `na.fill()` o `replace()`.
4. **Manejo de datos faltantes en cálculos**: R maneja los datos faltantes de manera coherente en cálculos matemáticos y operaciones, devolviendo `NA` en el resultado si uno de los valores es faltante.

Algunos ejemplos:

1. con 'is.na()'

```
# Creamos un valor faltante (NA)
x <- NA

# Verificamos si x es un valor faltante (NA)
is.na(x)
```

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

2. con 'na.omit()'

```
# Creamos un marco de datos con valores faltantes
datos <- data.frame(
  A = c(1, 2, NA, 4),
  B = c("a", "b", NA, "d")
)
print(datos)
```

```
##    A    B
## 1  1    a
## 2  2    b
## 3 NA <NA>
## 4  4    d
```

```
# Eliminamos las filas con valores faltantes
(datos_sin_na <- na.omit(datos))
```

```
##   A B
## 1 1 a
## 2 2 b
## 4 4 d
```

3. con 'replace()'

```
# Crear un dataframe de ejemplo con valores faltantes
df <- data.frame(
  A = c(1, 2, NA, 4, 5),
  B = c("a", NA, "c", "d", "e")
)

# Imprimir el dataframe original
print("DataFrame original:")
```

```
## [1] "DataFrame original:"
```

```
print(df)
```

```
##    A    B
## 1  1    a
## 2  2 <NA>
## 3 NA    c
## 4  4    d
## 5  5    e
```

```
# Reemplazar los valores faltantes con un valor específico
df_filled <- df
df_filled$A <- replace(df_filled$A, is.na(df_filled$A), 0)
df_filled$B <- replace(df_filled$B, is.na(df_filled$B), "missing")

# Imprimir el dataframe con los valores faltantes reemplazados
print("DataFrame con valores faltantes reemplazados:")
```

```
## [1] "DataFrame con valores faltantes reemplazados:"
```

```
print(df_filled)
```

```
##   A       B
## 1 1       a
## 2 2 missing
## 3 0       c
## 4 4       d
## 5 5       e
```

Además del NA, existe el **Not a Number** y se usa cuando el resultado de una operación matemática resulta en algo imposible:

```
print(0/0)
```

```
## [1] NaN
```

También, tenemos también **NULL** que indica la ausencia de todo dato. Puede usarse también para designar variable.

Para finalizar, existe la posibilidad de obtener de respuesta **character(0)**

```
(var <- c('ejemplo', 1))
```

```
## [1] "ejemplo" "1"
```

```r
(var <- var[-2])  # Eliminar el segundo elemento del vector
```

```
## [1] "ejemplo"
```

```r
(var <- var[-1])
```

```
## character(0)
```

Cuando un vector se encuentra vacío, ya sea porque se eliminaron todos sus elementos o porque no se crearon elementos en primer lugar, R lo representa como **character(0)** si es un vector de caracteres, **numeric(0)** si es un vector numérico, **logical(0)** si es un vector lógico, y así sucesivamente, dependiendo del tipo de datos del vector.

Aprender a manipular este tipo de datos es importante para cosas como limpieza de datos.

Capítulo 12: Creación de archivos a partir de R

Crear un archivo directamente desde la consola de R es muy sencillo. Puedes hacerlo utilizando funciones como `file.create()`, `writeLines()` o `write.table()`, dependiendo del tipo de archivo que quieras crear. Aquí tienes 4 diferentes opciones según lo que se necesite:

1.Crear un archivo vacio

Si solo necesitas crear un archivo vacío (como un archivo .txt, .csv, o cualquier extensión) vamos a usar la función `file.create()`:

```r
file.create("/Users/alexisxavier/Desktop/mi_archivo.txt")
```

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

```r
#Poner la ruta completa para saber donde está tu archivo
```

O saber desde un principio dónde estas trabajando con la función `getwd()`y después:

```r
file.create("mi_archivo.txt")
```

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

El archivo se creará en el directorio donde estés ubicado al momento de ejectuar esa función.

2. Escribir contenido en un archivo (texto simple)

Si deseas crear un archivo con contenido desde la consola, se usará la función `writeLines()`:

```r
writeLines(c("Primera línea", "Segunda línea"), "mi_archivo2.txt")
```

Observe que como primer argumento está un vector `character` y como segundo argumento está el lugar donde se va a guardar ese vector.

3. Crear un archivo con datos estructurados (como .csv)

Para escribir datos tabulares (por ejemplo, un data.frame) en un archivo:

```r
write.csv(mtcars, "mi_archivo3.csv", row.names = FALSE)
```

Esto creará un archivo con el data frame de mtcars. Se recomienda saber el directorio de trabajo antes de ejecutar la función para encontrar el archivo que se está creando.

4.Crear un archivo RMarkdown o script R

```r
# Crear un script R vacío
file.create("mi_script.R")
```

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

```r
# Crear un archivo RMarkdown vacío
file.create("mi_documento.Rmd")
```

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

Ejemplo:

```r
# Escribir código inicial en el script
writeLines(c("# Este es mi script", "print('Hola mundo')"), "mi_script.R")

# Crear un RMarkdown básico
writeLines(c(
  "---",
  "title: 'Mi Documento'",
  "output: html_document",
  "---",
  "",
  "## Introducción",
  "",
  "Este es un ejemplo de RMarkdown."
), "mi_documento.Rmd")
```

Los documentos Markdown son importantes a la hora de presentar trabajos con lineas de código y tienen salida en formato html, pdf y word. Este documento fue creado precisamente en Markdown con salida en pdf.

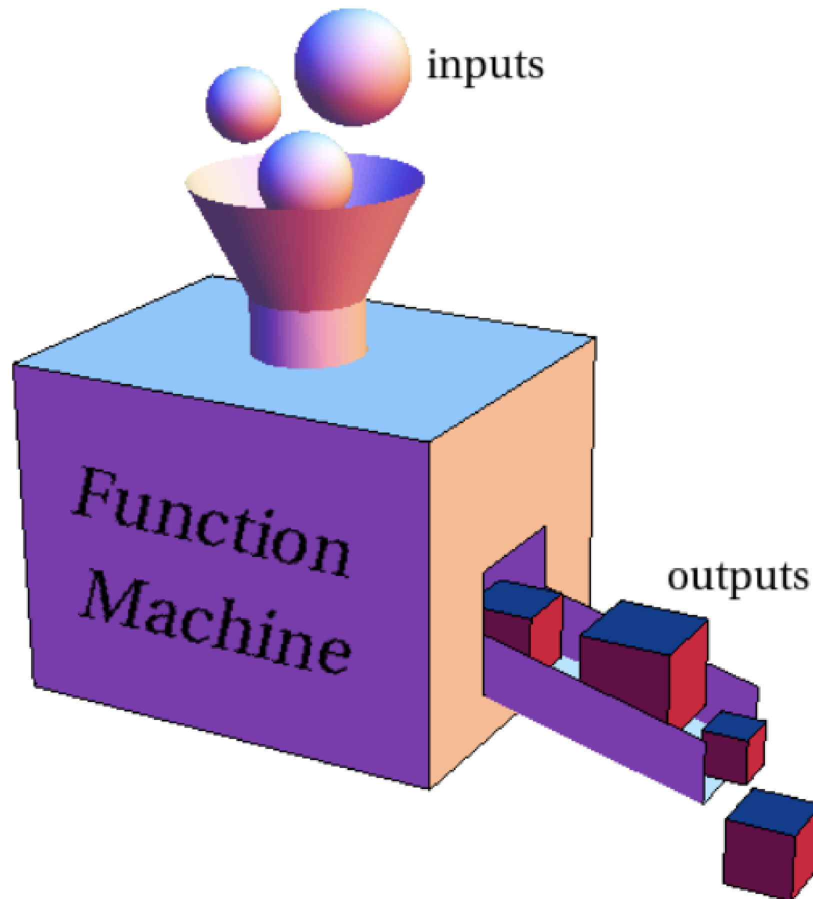
Los script y los markdown van a tener diferente proposito y utlidad según sea el caso. Por ejemplo, markdown es útil a la hora de reportar código.

Capítulo 13: Funciones

Una de las ventajas de R es la gran cantidad de funciones que contiene para realizar distintos procedimientos.

¿Qué es una función?

Como se define en el manual de R (Hernández & Usuga, 2024), "una función es un conjunto de instrucciones que convierten las entradas (inputs) en resultados (outputs) deseados." En la siguiente figura se muestra una ilustración de lo que es una función.



Estructura de una función:

```
fun1 <- function(p1,p2,...){
  cuerpo
  cuerpo
  cuerpo
  cuerpo
  return(resultados)
}
```

Ejemplos

Ejemplo 1: Construir una función que salude.

```
saludar <- function(nombre){
  paste("Hola", nombre)

}
saludar(nombre = "Alexis")
```

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

```
saludar("Alexis")
```

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

Ejemplo 2: Construir una función que reciba dos números y que entregue la suma de estos números.

```
suma <- function(x,y){
  resultado <- x + y
  return(resultado)
}
```

Para acceder a la función deseada solamente se debe ejecutar:

```
suma(x=4,y=6)
```

```
## [1] 10
```

Para funciones simples es posible escribirlas de manera más sencillas como se muestra a continuación:

```
suma2 <- function(x,y){
  return(x+y)
}

suma2(2,8)
```

```
## [1] 10
```

Ejemplo 3: Construya una función que reciba dos números de la recta real y que entregue el punto médio de estos números. El resultado debe ser un mensaje por pantalla.

```
medio <- function(a, b) {
  medio <- (a + b) / 2
  cat("El punto medio de los valores", a, "y", b,
      "ingresados es", medio)
}

medio(a=-3, b=3)  # Probando la función
```

```
## El punto medio de los valores -3 y 3 ingresados es 0
```

La función 'cat' es muy útil para presentar resultados por pantalla.

Capítulo 14: Estructuras de control

En R se disponen de varias estructuras de control para facilitar los procedimientos que un usuario debe realizar. A continuación se explican esas instrucciones de control.

If

Esta instrucción sirve para realizar un conjunto de operaciones si se cumple cierta condición.

Estructura:

«if» («Condición») {

«operación 1»

«operación 2»

...

«operación final»

}

Ejemplo:

Crear un código para calcular el salario final de un empleado:

```
sal_bas <- 420   # Salario básico por semana
horas_lab <- 45 # Horas laboradas por semana

if (horas_lab > 40) {
  horas_ext <- horas_lab - 40
  sal_ext <- horas_ext * 0.05
  salario <- sal_bas + sal_ext
}

salario # Salario semanal
```

```
## [1] 420.25
```

If - Else

`Else` sirve para realizar un conjunto de operaciones cuando NO se cumple cierta condición evaluada por un `if`.

Estructura:

«if («Condición») {

«operación 1»

«operación 2»

...

«operación final»

} else {

«operación 1»

«operación 2»

...

«operación final»

}

Ejemplo:

Supongamos que trabajas en una tienda de ropa y deseas implementar un sistema que determine el descuento que se aplicará a un cliente basado en el total de su compra. La política de descuentos es la siguiente:

- Si el total de la compra es mayor o igual a 100, se aplica un descuento del 20% y se suma un bono de 10 $.
- Si el total es menor a 100, no hay descuento, pero se suma un cargo adicional de 5$.

Escriba un procedimiento que reciba como argumento el total de la compra y devuelva el monto final después de aplicar las operaciones correspondientes.

```
total_compra <- 450 # Supongamos que el total de compra es 450

if (total_compra >= 100) {
    descuento <- total_compra * 0.20
    precio_final <- total_compra - descuento + 10
} else {
    precio_final <- total_compra + 5
}

precio_final
```

```
## [1] 370
```

Ifelse

Se recomienda usar la instrucción `ifelse` cuando hay una sola instrucción para el caso `if` y para el caso `else`.

Estructura:

ifelse(«Condición», «operación SI cumple», «operación NO cumple»)

Ejemplo:

Suponga que usted recibe un vector de números enteros, escriba un procedimiento que diga si cada elemento del vector es par o impar.

```
# Supongamos que tenemos un vecto x con varios números:
x <- c(5, 3, 2, 8, -4, 1)
ifelse(x %% 2 == 0, "Es par", "Es impar")
```

```
## [1] "Es impar" "Es impar" "Es par"   "Es par"   "Es par"   "Es impar"
```

Capítulo 15: Bucles

En R, los bucles `for` y `while` son estructuras de control que permiten repetir un bloque de código varias veces. A continuación, se explican con detalle y se muestran ejemplos para cada uno.

For

Estructura

for («variable» in «vector») {

«Código a ejecutar»

«salida»

}

Este bucle se utiliza para iterar sobre una secuencia (como vectores listas, conjuntos de números, etc.) y ejecutar un bloque de código para cada elemento de la secuencia.

Ejemplos:

Ejemplo 1

```
#Sintáxis básicas
#Supongamos que queremos imprimir los números del 1 al 5:
for (i in 1:5) {
    print(i)
}
```

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

En este ejemplo, i toma los valores de 1 a 5 en cada iteración, y la función print imprime cada valor.

Ahora vamos a ver que se va a poder hacer algo similar pero con `character´s`:

```
nombres <- c("Alexis","Xavier","Ivo", "Victor", "Emanuel", "Rodrigo", "Ana",
             "Paola", "María", "Pedro")

for (j in nombres){
  print(j)
}
```

```
## [1] "Alexis"
## [1] "Xavier"
## [1] "Ivo"
## [1] "Victor"
## [1] "Emanuel"
## [1] "Rodrigo"
## [1] "Ana"
## [1] "Paola"
## [1] "María"
## [1] "Pedro"
```

Ejemplo 2:

Suma de Gauss

```
x <-0
for (i in 1:10){
  x <- x+i
  print(x)
}
```

```
## [1] 1
## [1] 3
## [1] 6
## [1] 10
## [1] 15
## [1] 21
## [1] 28
## [1] 36
## [1] 45
## [1] 55
```

Ejemplo 3:

Ejemplo con mtcars

```
for (col in colnames(mtcars)) {
  print(col)
  print(mtcars[,col])
  print(" ")
}
```

```
## [1] "mpg"
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4
## [16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7
## [31] 15.0 21.4
## [1] " "
## [1] "cyl"
##  [1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8 4 4 4 4 8 8 8 8 4 4 4 8 6 8 4
## [1] " "
## [1] "disp"
##  [1] 160.0 160.0 108.0 258.0 360.0 225.0 360.0 146.7 140.8 167.6 167.6 275.8
## [13] 275.8 275.8 472.0 460.0 440.0  78.7  75.7  71.1 120.1 318.0 304.0 350.0
## [25] 400.0  79.0 120.3  95.1 351.0 145.0 301.0 121.0
## [1] " "
## [1] "hp"
##  [1] 110 110  93 110 175 105 245  62  95 123 123 180 180 180 205 215 230  66  52
## [20]  65  97 150 150 245 175  66  91 113 264 175 335 109
## [1] " "
## [1] "drat"
##  [1] 3.90 3.90 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 3.92 3.07 3.07 3.07 2.93
## [16] 3.00 3.23 4.08 4.93 4.22 3.70 2.76 3.15 3.73 3.08 4.08 4.43 3.77 4.22 3.62
## [31] 3.54 4.11
## [1] " "
## [1] "wt"
##  [1] 2.620 2.875 2.320 3.215 3.440 3.460 3.570 3.190 3.150 3.440 3.440 4.070
## [13] 3.730 3.780 5.250 5.424 5.345 2.200 1.615 1.835 2.465 3.520 3.435 3.840
## [25] 3.845 1.935 2.140 1.513 3.170 2.770 3.570 2.780
```

```
## [1] " "
## [1] "qsec"
##  [1] 16.46 17.02 18.61 19.44 17.02 20.22 15.84 20.00 22.90 18.30 18.90 17.40
## [13] 17.60 18.00 17.98 17.82 17.42 19.47 18.52 19.90 20.01 16.87 17.30 15.41
## [25] 17.05 18.90 16.70 16.90 14.50 15.50 14.60 18.60
## [1] " "
## [1] "vs"
##  [1] 0 0 1 1 0 1 0 1 1 1 1 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 0 1 0 0 0 1
## [1] " "
## [1] "am"
##  [1] 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1
## [1] " "
## [1] "gear"
##  [1] 4 4 4 3 3 3 3 4 4 4 4 3 3 3 3 3 3 4 4 4 3 3 3 3 3 4 5 5 5 5 5 4
## [1] " "
## [1] "carb"
##  [1] 4 4 1 1 2 1 4 2 2 4 4 3 3 3 4 4 4 1 2 1 1 2 2 4 2 1 2 2 4 6 8 2
## [1] " "
```

Notar como la salida es seriada en cuanto a que imprime las tres cosas que pedi y después continua hasta hasta acabar el Loop

Ejemplo 4:

For dentro de otro For

```r
for (i in c("Primero", "Segundo", "Tercero")) {
  for (j in c("a", "b", "c")) {
    print(paste(i,j, sep = "-"))

  }

}
```

```
## [1] "Primero-a"
## [1] "Primero-b"
## [1] "Primero-c"
## [1] "Segundo-a"
## [1] "Segundo-b"
## [1] "Segundo-c"
## [1] "Tercero-a"
## [1] "Tercero-b"
## [1] "Tercero-c"
```

While

Estructura

while («Condición»){

«Código a ejecutar mientra la condición se cumpla»

}

Este bucle se utiliza para repetir un bloque de código mientras una condición especificada sea verdadera. La condición se evalúa antes de cada iteración del bucle.

Ejemplo 1:

```
#Sintáxis básica
#Supongamos que queremos imprimir los números del 1 al 5 usando un bucle while:
i <- 1
while (i <= 5) {
    print(i)
    i <- i + 1
}
```

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

En este ejemplo, se inicializa i a 1 antes de comenzar el bucle. Luego, en cada iteración, se imprime el valor de i y se incrementa en 1. El bucle continúa ejecutándose mientras i sea menor o igual a 5.

Ejemplo 2:

```
x <- 0

while (x < 100) {
  print("Hola")
  print(x)
  x <- x+10

}
```

```
## [1] "Hola"
## [1] 0
## [1] "Hola"
## [1] 10
## [1] "Hola"
## [1] 20
## [1] "Hola"
## [1] 30
## [1] "Hola"
## [1] 40
## [1] "Hola"
## [1] 50
## [1] "Hola"
## [1] 60
## [1] "Hola"
## [1] 70
## [1] "Hola"
## [1] 80
## [1] "Hola"
## [1] 90
```

Ejemplo 3:

Podemos crear un proceso infinto:

```
# while (TRUE) {
  # y <- y+1
  # print(y)
# }
```

Hasta que nosotros lo detengamos con STOP en la parte superior derecha de donde se tenga la consola, nunca se detendrá.

Pero agregando un `break` dentro del `while` se va a detener sin la necesidad de hacerlo nosotros manualmente:

```
y <-90
while (TRUE) {
  y = y+1 #Note el uso de un "=" en lugar de la flecha "<-"
  print(y)
  if (y>100){
    break
  }

}
```

```
## [1] 91
## [1] 92
## [1] 93
## [1] 94
## [1] 95
## [1] 96
## [1] 97
## [1] 98
## [1] 99
## [1] 100
## [1] 101
```

Repeat:

La instrucción `repeat` es muy útil para repetir un procedimiento siempre que se cumple una condición.

Estructura:

repeat {

«operación 1»

«operación 2»

...

«operación final»

if («condición») break

}

Ejemplo: Escribir un procedimiento para ir aumentando de uno en uno el valor de z hasta que z sea igual a siete El procedimiento debe imprimir por pantalla la secuencia de valores de z.

```
z <- 3  # Valor de inicio

repeat {
   print(z)
   z <-  z + 1
```

```
  if (z == 8) {
    break
  }
}
```

```
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
```

- **for es útil cuando el número de iteraciones es fijo o conocido de antemano.**
- **while es útil cuando las iteraciones dependen de una condición lógica y el número no es fijo.**
- **repeat es útil cuando necesitas un bucle infinito controlado por una condición manual (break).**

Ejemplos adicionales

```
vec <- c("a", "b", "c", "d")
for (letra in vec) {
    print(letra)
}
```

```
## [1] "a"
## [1] "b"
## [1] "c"
## [1] "d"
```

En este ejemplo, letra toma cada valor del vector vec en cada iteración y se imprime.

```
x <- 1
while (x^2 < 100) {
    print(x)
    x <- x + 2
}
```

```
## [1] 1
## [1] 3
## [1] 5
## [1] 7
## [1] 9
```

En este caso, el bucle imprime valores de x mientras el cuadrado de x sea menor que 100, incrementando x en 2 en cada iteración.

```
# Contar del 1 al 5 usando repeat
contador <- 1
repeat {
  print(contador)
  contador <- contador + 1

  if (contador > 5) {  # Termina cuando contador sea mayor a 5
    break
  }
}
```

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

Ejercicio:

Realice un `while` que active la alerta sismica en caso que la frecuencia aleatoria monitoreada sea mayor a 99:

```r
set.seed(10) #Establecemos una semilla para los números aleatorios
#Esta función se retomará más adelante
frecuencia <- 0  # Primero definimos la variable frecuencia y le asignamos 0.
registros <- c() # Luego creamos un vector vacio.
while (frecuencia < 99) {
  print( paste("Frecuencia estable. Frecuencia:", frecuencia))
  #Imprimir que la freecuencia fue estable en caso de que no de mayor a 99
  frecuencia <- rnorm(1, mean = 97, sd=2)
  #Definir la aleatoriedad de la frecuencia en base a la distribucición normal.
}
```

```
## [1] "Frecuencia estable. Frecuencia: 0"
## [1] "Frecuencia estable. Frecuencia: 97.0374923418836"
## [1] "Frecuencia estable. Frecuencia: 96.6314949158619"
## [1] "Frecuencia estable. Frecuencia: 94.257338900155"
## [1] "Frecuencia estable. Frecuencia: 95.8016645684326"
## [1] "Frecuencia estable. Frecuencia: 97.589090253135"
## [1] "Frecuencia estable. Frecuencia: 97.7795886014003"
## [1] "Frecuencia estable. Frecuencia: 94.583847649141"
## [1] "Frecuencia estable. Frecuencia: 96.2726479650583"
## [1] "Frecuencia estable. Frecuencia: 93.7466546365938"
## [1] "Frecuencia estable. Frecuencia: 96.487043211752"
```

Note que la función `rnorm()` genera numeros pseudoaleatorios en base a la distribución normal, se dicen pseudoaleatorios porque los números que produce no son verdaderamente aleatorios, sino que son el resultado de un algoritmo determinista que simula el comportamiento de dicha distribución. Hablaremos de aleatoriedad en el siguiente capítulo.