Clase 3 (Beta)

Alexis Zúñiga & Ernesto Barrios

Capítulo 5: Listas

Las listas pueden entenderse como vectores más complejos, cada elemento de la lista puede tener una estructura diferente, para crear una lista se utiliza la funición `list()`.

```
ejemplo_lista <- list(1,2,3,4)
```

En su forma más basica la lista puede usarse como vector, pero pueden llegar a ser muy complejas y por ende, utiles para recopilación de datos.

```
primer_lista <- list(1,2,"a","b", NA, T, FALSE)
```

Notar que: Las respuestas de la consola son diferentes a las que genera un vector,los valores que ingresamos mantiene su estuctura original y separación entre rengolones.

Las listas pueden incluir vectores.

```
Lista_2 <- list(x=c('Alexis','Xavier'), y=c(1,2,3), z= c(NA, NA,NA),
                w=c(T,F,T))
Lista_2
```

```
## $x
## [1] "Alexis" "Xavier"
##
## $y
## [1] 1 2 3
##
## $z
## [1] NA NA NA
##
## $w
## [1]  TRUE FALSE  TRUE
```

Acceso a datos de las listas

Para acceder a los datos que tenemos en nuestras listas podemos usar índices:

```
ejemplo_lista[3] #Accede a la tercer posición de mi primer ejemplo de lista
```

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

Tambien se puede **modificar elementos** con el operador '<-'

```
primer_lista[1] <- 'José'
primer_lista
```

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

```
## [1] "José"
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] "a"
##
## [[4]]
## [1] "b"
##
## [[5]]
## [1] NA
##
## [[6]]
## [1] TRUE
##
## [[7]]
## [1] FALSE
```

Acceder a varios elementos:

```
Lista_2['w']
```

```
## $w
## [1]  TRUE FALSE  TRUE
```

En el proceso para acdeder a datos en las listas, si hacemos lo mismo que haciamso para acceder a vectore, obtendremos otra lista de un solo elemento que contiene al dato buscado, en lugar de el dato buscado por sí solo.

```
Lista_2[1]
```

```
## $x
## [1] "Alexis" "Xavier"
```

Para acceder al primer elemento del primer vector de la lista:

```
Lista_2[[1]]
```

```
## [1] "Alexis" "Xavier"
```

Vemos como la salida ahora es un vector.

Como nuestro resultado es un vector esto significa que va a responder exactamente como lo haría un vector. Si quisiéramos acceder al tercer elemento de este vector escribimos:

```
Lista_2[[1]][1]
```

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

Los elementos de las listas con nombre también se pueden acceder a través del operador de acceso `$`. Al usarlo, ya no utilizamos el doble corchete ni las comillas sino que usamos el operador y el nombre directamente.

```
Lista_2$y
```

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

Por último veamos la función `unlist()`, la cual convierte una lista a un vector.

```
unlist(ejemplo_lista)
```

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

Capítulo 6: Factores

En R , los factores se usan para trabajar con variables categóricas, es decir, variables que tienen un conjunto fijo y conocido de valores posibles. También son útiles cuando quieres mostrar vectores de caracteres en un orden no alfabético.

```
medallas <- c('Oro','Plata','Bronce','Oro','Plata','Plata','Plata','Oro')
factor(medallas)
```

```
## [1] Oro    Plata  Bronce Oro    Plata  Plata  Plata  Oro
## Levels: Bronce Oro Plata
```

Ahora el vector nos indica cuáles son los niveles del vector proporcionado. Esto puede ser útil para graficar o mucho más adelante, para entrenar modelos de clasificación.

```
medallas2 <- c('Oro','Plata','Oro','Plata','Plata','Plata','Oro')
fac_medallas <- factor(medallas2, levels = c('Oro','Plata','Bronce'))
fac_medallas
```

```
## [1] Oro   Plata Oro   Plata Plata Plata Oro
## Levels: Oro Plata Bronce
```

También le podemos indicar cuáles son los niveles de un factor, aunque el vector que estamos convirtiendo no los contenga. Esto lo hacemos con el argumento levels.

```
#Revisamos sólo los niveles
levels(fac_medallas)
```

```
## [1] "Oro"    "Plata"  "Bronce"
```

Incluso le podemos indicar a R que las categorías tienen orden y cuál es la jerarquía dentro de estas.

```
(fac_medallas_2 <- factor(medallas2, levels = c('Oro','Plata','Bronce'), ordered = T) )
```

```
## [1] Oro   Plata Oro   Plata Plata Plata Oro
## Levels: Oro < Plata < Bronce
```

Intuitivamente vemos que esa jerarquía no es correcta. Modifiquemosla con la función ordered().

```
ordered(fac_medallas_2, c('Bronce','Plata','Oro'))
```

```
## [1] Oro   Plata Oro   Plata Plata Plata Oro
## Levels: Bronce < Plata < Oro
```

Capítulo 7 : Data Frames

Los Data Frames son estructuras de datos fundamentales en R, especialmente útiles para trabajar con conjuntos de datos tabulares. Puedes pensar en ellos como tablas en una base de datos o hojas de cálculo en Excel. Aquí tienes algunos conceptos básicos:

Para crear un data frame usamos la función `data.frame()` y le proporcionamos vectores atómicos como columnas.

```
arreglo1 <- data.frame(Columna_1=c("Alexis","Zúñiga", "García"), #importante la coma
                       Columna_2=c(6,6,6),
                       Columna_3=c(T,F,T),
                       row.names = c("renglón 1","renglón 2", "renglón 3")) #función row.names()
arreglo1
```

```
##           Columna_1 Columna_2 Columna_3
## renglón 1    Alexis         6      TRUE
## renglón 2    Zúñiga         6     FALSE
## renglón 3    García         6      TRUE
```

Nótese que, como las columnas están generadas por vectores atómicos, cada una solo puede tener un tipo de dato.

Para acceder a los valores individuales usamos la misma sintaxis que usamos para los vectores pero ahora con dos posiciones separadas por una coma. La primera corresponde a la fila que queremos y la segunda a la columna.

```
arreglo1[1]
```

```
##           Columna_1
## renglón 1    Alexis
## renglón 2    Zúñiga
## renglón 3    García
```

Sin embargo la forma propia de acceder este valor es con un espacio en blanco en la sección de las filas y el índice de la columna después de la coma.

```
#primera columna
arreglo1[,1]
```

```
## [1] "Alexis" "Zúñiga" "García"
```

Por otro lado, si queremos solo la primera fila, escribimos la coma pero dejamos en blanco lo que viene después.

```
#Primera fila
arreglo1[1,]
```

```
##           Columna_1 Columna_2 Columna_3
## renglón 1    Alexis         6      TRUE
```

De manera similar a las listas con nombre, podemos usar el operador de acceso $ para seleccionar columnas por nombre.

```
arreglo1$Columna_1
```

```
## [1] "Alexis" "Zúñiga" "García"
```

Capítulo 8: Explorar las estructuras

Podemos también explorar las estructuras de datos para obtener información básica que nos ayuda a comprender los datos y el tipo de estructura con el que estamos trabajando. En el caso de los arreglos, si queremos saber el número de renglones y columnas usamos `dim()`. Esta función regresa un vector con dos entradas numéricas que nos indican el número de renglones y columnas. Alternativamente podemos usar `ncol()` o `nrow()` para obtener estos datos individualmente.

```
dim(arreglo1)
```

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

```
ncol(arreglo1)
```

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

```
nrow(arreglo1)
```

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

```
dim(arreglo1)[1]
```

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

Además de examinar las dimensiones de nuestras estructuras, también podemos explorar otras características. La función `summary()` es útil tanto para vectores como para matrices, proporcionándonos datos básicos que facilitan la comprensión de la estructura subyacente.

Por otra parte, podemos inspeccionar una porción pequeña de un data frame o matriz, ya sea al principio o al final, utilizando las funciones `head()` y `tail()`. Estas funciones requieren dos argumentos: el primero debe ser la estructura de datos que se va a inspeccionar, mientras que el segundo (opcional) indica cuántas filas deben mostrarse. Si no se especifica el segundo argumento, por defecto estas funciones mostrarán las primeras 6 filas.

R mantiene una serie de matrices de ejemplo cargadas en la memoria en todo momento. En nuestro caso de estudio, para la exploración de datos, emplearemos el data frame mtcars.

```
mtcars
```

```
##                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL        17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC       15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128          32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic       30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
```

```
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

A primera vista es demasiada información para procesar rápidamente. Usemos las técnicas de exploración de esta sección para conocer lo más posible acerca de mtcars.

```
dim(mtcars)
```

```
## [1] 32 11
```

```
colnames(mtcars)
```

```
##  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
## [11] "carb"
```

```
row.names(mtcars)
```

```
##  [1] "Mazda RX4"           "Mazda RX4 Wag"       "Datsun 710"
##  [4] "Hornet 4 Drive"      "Hornet Sportabout"   "Valiant"
##  [7] "Duster 360"          "Merc 240D"           "Merc 230"
## [10] "Merc 280"            "Merc 280C"           "Merc 450SE"
## [13] "Merc 450SL"          "Merc 450SLC"         "Cadillac Fleetwood"
## [16] "Lincoln Continental" "Chrysler Imperial"   "Fiat 128"
## [19] "Honda Civic"         "Toyota Corolla"      "Toyota Corona"
## [22] "Dodge Challenger"    "AMC Javelin"         "Camaro Z28"
## [25] "Pontiac Firebird"    "Fiat X1-9"           "Porsche 914-2"
## [28] "Lotus Europa"        "Ford Pantera L"      "Ferrari Dino"
## [31] "Maserati Bora"       "Volvo 142E"
```

```
head(mtcars)#primeros 5 por dafault
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
head(mtcars,2) #primeros 2
```

```
##               mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4      21   6  160 110  3.9 2.620 16.46  0  1    4    4
## Mazda RX4 Wag  21   6  160 110  3.9 2.875 17.02  0  1    4    4
```

```
tail(mtcars,3)
```

```
##                mpg cyl disp  hp drat   wt qsec vs am gear carb
```

```
## Ferrari Dino  19.7   6  145 175 3.62 2.77 15.5  0  1    5    6
## Maserati Bora 15.0   8  301 335 3.54 3.57 14.6  0  1    5    8
## Volvo 142E    21.4   4  121 109 4.11 2.78 18.6  1  1    4    2
```

```
summary(mtcars)
```

```
##       mpg             cyl             disp             hp       
##  Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0  
##  1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5  
##  Median :19.20   Median :6.000   Median :196.3   Median :123.0  
##  Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7  
##  3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0  
##  Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0  
##       drat             wt             qsec             vs        
##  Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000  
##  1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000  
##  Median :3.695   Median :3.325   Median :17.71   Median :0.0000  
##  Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375  
##  3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000  
##  Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000  
##        am              gear            carb      
##  Min.   :0.0000   Min.   :3.000   Min.   :1.000  
##  1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000  
##  Median :0.0000   Median :4.000   Median :2.000  
##  Mean   :0.4062   Mean   :3.688   Mean   :2.812  
##  3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000  
##  Max.   :1.0000   Max.   :5.000   Max.   :8.000
```

En este último vemos que nos da el resumen de cada vector (columna) de nuestro arreglo ejemplo. Esto es el mínimo, el máximo, la media, la mediana y los cuantiles. Si hubiera algún valor `NA`, `summary()` nos daría el número total como parte del reporte.

Capítulo 9: 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"
```

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

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

```
(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.