Estructuras de datos en R

Vectores atómicos, listas, matrices, factores, *data frames* y *tibbles*

Dr. Samuel D. Gamboa Tuz

Vectores

- Conjuntos de uno (*scalar*) o varios valores, o elementos.
- Propiedades:
  - **Tipo**. Se checa con la función `typeof()`.
  - **Longitud**. Se checa con la función `length()`.
  - **Atributos**. Se checa con la función `attributes()`.
- Se dividen en **vectores atómicos** y **listas**.
- Vectores atómicos:
  - Contiene valores homogéneos (un solo tipo).
  - Se construyen con la funcion `c()` (*combine*).
  - Cuatro tipos: *integer*, *double*, *character*, *logical*.
- Listas:
  - Pueden contener valores heterogéneos (varios tipos).
  - Se construyen con la función `list()`.
  - Único tipo: *list*.

Tipos de vectores atómicos y listas

integer, *double*, *character*, *logical*, *list*

integer (int)

- Valores numéricos enteros seguidos por la letra *L*.

```
int_vector <- c(1L, 4e3L, NA, 0L, -5L, 600L)
int_vector
```

```
## [1]    1 4000   NA    0   -5  600
```

```
typeof(int_vector)
```

```
## [1] "integer"
```

```
is.integer(int_vector)
```

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

```
length(int_vector)
```

```
## [1] 6
```

- También se pueden crear con la función `seq()` o el operador `:` usando números enteros. ¿Habrá más formas?

double (dbl)

- *Double-precision floating-point*.
- Valores numéricos con punto decimal. Ej. 0.1, 10.0, -5.5, 6.6e-3.
- En la práctica, R convierte todos los números que ingresamos a tipo *double*.

```
dbl_vector <- c(NaN, 4e-05, -3, Inf, 5.0, NA)
dbl_vector
```

```
## [1]    NaN  4e-05 -3e+00    Inf  5e+00     NA
```

```
typeof(dbl_vector)
```

```
## [1] "double"
```

```
is.double(dbl_vector)
```

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

```
length(dbl_vector)
```

```
## [1] 6
```

character (chr)

- Valores (*strings*) encerrados en comillas simples (') o dobles (").

```
chr_vector <- c("1", "", "frase o palabra", NA, "NA", 200)
chr_vector
```

```
## [1] "1"                "" "frase o palabra"
## [4] NA                 "NA"               "200"
```

```
typeof(chr_vector)
```

```
## [1] "character"
```

```
is.character(chr_vector)
```

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

```
length(chr_vector)
```

```
## [1] 6
```

logical (lgl)

- Valores de verdadero o falso.

```
lgl_vector <- c(TRUE, FALSE, NA, F, T, TRUE)
lgl_vector
```

```
## [1]  TRUE FALSE    NA FALSE  TRUE  TRUE
```

```
typeof(lgl_vector)
```

```
## [1] "logical"
```

```
is.logical(lgl_vector)
```

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

```
length(lgl_vector)
```

```
## [1] 6
```

Coerción de vectores atómicos

- *character* > *double* > *integer* > *logical*.
- ¿Cuál será el tipo de estos vectores atómicos?

```
a <- c("a", 2.0, 1L, TRUE, NA)
a
```

```
## [1] "a"    "2"    "1"    "TRUE" NA
```

```
b <- c(5.5, 1L, FALSE, NA)
b
```

```
## [1] 5.5 1.0 0.0  NA
```

```
c <- c(1L, TRUE, FALSE, NA)
c
```

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

Coerción explícita

- Se puede convertir de un tipo de vector a otro con funciones `as.*()`.
- Cuando no sea posible, se asignan `NA`s.

```
x <- c("1", "2", "3", "cuatro", "T", "FALSE")
is.character(x)
```

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

```
as.double(x)
```

```
## Warning: NAs introduced by coercion
```

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

```
as.integer(x)
```

```
## Warning: NAs introduced by coercion
```

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

```
as.logical(x)
```

```
## [1]    NA    NA    NA    NA  TRUE FALSE
```

```r
y <- c(1, 2, -3, 4.3, 5.5, 0)
is.double(y)
```

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

```r
as.logical(y)
```

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

```r
as.character(y)
```

```
## [1] "1"   "2"   "-3"  "4.3" "5.5" "0"
```

```r
as.integer(y)
```

```
## [1]  1  2 -3  4  5  0
```

- En la coerción de *double* o *integer* a *logical*, los ceros se convierten en `FALSE` y cualquier otro número se convierte en `TRUE`.
- En la coerción de *double* a *integer*, los decimales se pierden.

```r
z <- c(TRUE, FALSE, TRUE, FALSE, NA)
is.logical(z)
```

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

```r
as.character(z)
```

```
## [1] "TRUE"  "FALSE" "TRUE"  "FALSE" NA
```

```r
as.double(z) # o as.integer(z)
```

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

- Puedes sumar el número o proporción de valores `TRUE` en un vector con `sum()` o `mean()`, pero hay que elimnar los `NA`s antes con `na.omit()`.

```r
sum(na.omit(z))
```

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

```r
mean(na.omit(z))
```

```
## [1] 0.5
```

list

- Una lista es una colección de objetos, los cuales pueden ser vectores atómicos (de diferentes tipos), otras listas, matrices, *data frames*, etc.
- Se crean con la función `list()`.

```
list_vector <- list(int_vector, dbl_vector, chr_vector, lgl_vector)
list_vector
```

```
## [[1]]
## [1]    1 4000   NA    0   -5  600
##
## [[2]]
## [1]    NaN  4e-05 -3e+00    Inf  5e+00     NA
##
## [[3]]
## [1] "1"               ""                "frase o palabra"
## [4] NA                "NA"              "200"
##
## [[4]]
## [1]  TRUE FALSE    NA FALSE  TRUE  TRUE
```

```
str(list_vector)
```

```
## List of 4
##  $ : int [1:6] 1 4000 NA 0 -5 600
##  $ : num [1:6] NaN 4e-05 -3e+00 Inf 5e+00 ...
##  $ : chr [1:6] "1" "" "frase o palabra" NA ...
##  $ : logi [1:6] TRUE FALSE NA FALSE TRUE TRUE
```

```
typeof(list_vector)
```

```
## [1] "list"
```

```
is.list(list_vector)
```

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

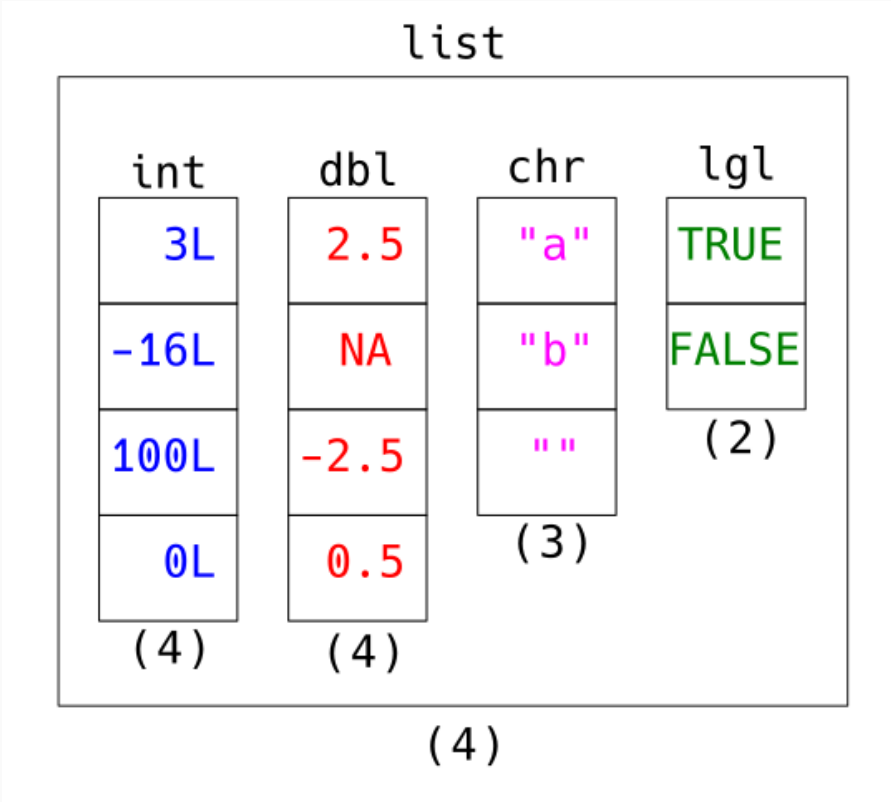
```
length(list_vector)
```

```
## [1] 4
```

- En este ejemplo, no importaría si un elemento de la lista tuviera una longitud de 1000, la longitud de la lista seguiría siendo cuatro.

En resumen...

- Hay cuatro tipos principales de vectores atómicos: *integer*, *double*, *character*, *logical*.
- Una lista (tipo *list*) es una colección de otros elementos; por ejemplo, vectores atómicos de diferentes tipos y longitudes, u otras listas.



Vectores con nombres

Atributo *names*

attributes

- Metadatos asociados a un vector o a sus elementos.
- Se pueden checar con la función `attributes()`.

```
attributes(dbl_vector)
```

```
## NULL
```

```
attributes(list_vector)
```

```
## NULL
```

names

- El atributo *names* es un vector tipo *character* de la misma longitud que el vector nombrado.
- Se utiliza la función `names()` para checar, asignar y borrar nombres.

```
names(dbl_vector)
```

```
## NULL
```

```
names(dbl_vector) <- c("A", "B", "C", "D", "E", "F")
dbl_vector
```

```
##      A      B      C      D      E      F
##    NaN  4e-05 -3e+00    Inf  5e+00     NA
```

```
names(dbl_vector)
```

```
## [1] "A" "B" "C" "D" "E" "F"
```

```
names(dbl_vector) <- NULL # Quitar nombres
names(dbl_vector)
```

```
## NULL
```

- También se puede asignar los nombres directamente al momento de crear un vector.

```
named_vector1 <- c("a" = 1, "b" = 2, "c" = 3)
named_vector2 <- c("1" = "a", "2" = "b", "3" = "d")
named_vector1
```

```
## a b c
## 1 2 3
```

```
named_vector2
```

```
##   1   2   3
## "a" "b" "d"
```

```
attributes(named_vector1)
```

```
## $names
## [1] "a" "b" "c"
```

```
names(named_vector2)
```

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

- En una lista, se puede nombrar sus elementos y los elementos de los vectores que contiene; se decir, puede contener dos o más juegos de nombres (depediendo de la estructura de la lista).

```
named_list <- list(named_vector1, named_vector2)
names(named_list) <- c("vector1", "vector2")
named_list
```

```
## $vector1
## a b c
## 1 2 3
##
## $vector2
##   1   2   3
## "a" "b" "d"
```

```
attributes(named_list)
```

```
## $names
## [1] "vector1" "vector2"
```

```
names(named_list)
```

```
## [1] "vector1" "vector2"
```

```
str(named_list)
```

```
## List of 2
##  $ vector1: Named num [1:3] 1 2 3
##   ..- attr(*, "names")= chr [1:3] "a" "b" "c"
##  $ vector2: Named chr [1:3] "a" "b" "d"
##   ..- attr(*, "names")= chr [1:3] "1" "2" "3"
```

- Nos podemos referir a los elementos de una lista por su nombre haciendo uso del operador `$` (adelanto de *subsetting*).

```
named_list$vector1
```

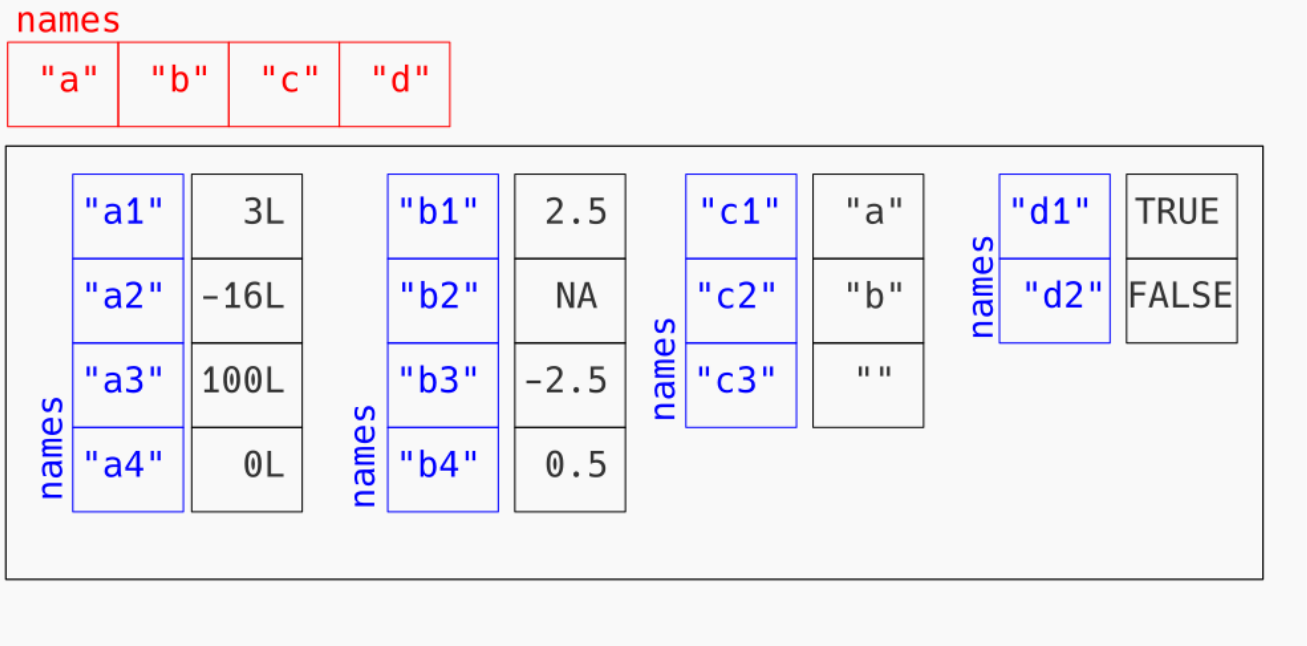
```
## a b c
## 1 2 3
```

```
named_list$vector2
```

```
##   1   2   3
## "a" "b" "d"
```

En resumen...

- El atributo `names` son metadatos en la forma de vectores tipo *character* de la misma longitud que los vectores que nombran.
- Sirve para identificar y referirse a los elementos de un vector.



Matrices

Atributos *dim* y *dimnames*

Matrices

- Una **matriz** es un vector atómico con dos dimensiones: número de filas y número de columnas.
- Se utiliza la función `dim()` para checar, agregar, y quitar dimensiones (similar a `names()`).

```
m1 <- c(1, 2, 3, 4, 5, 6)
dim(m1)
```

```
## NULL
```

```
dim(m1) <- c(2, 3)
m1
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

```
attributes(m1)
```

```
## $dim
## [1] 2 3
```

```r
is.matrix(m1)
```

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

```r
typeof(m1)
```

```
## [1] "double"
```

```r
dim(m1) <- NULL
is.matrix(m1)
```

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

- Por lo general, las matrices se crean con la función `matrix()`.

```r
mat <- matrix(
  data = m1,
  nrow = 2, ncol = 3, byrow = TRUE,
  dimnames = list(c("row1", "row2"), c("col1", "col2", "col3"))
)
mat
```

```
##      col1 col2 col3
## row1    1    2    3
## row2    4    5    6
```

```r
dimnames(mat) <- NULL
mat
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

```r
rownames(mat) <- c("ROW1", "ROW2")
colnames(mat) <- c("COL1", "COL2", "COL3")
mat
```

```
##      COL1 COL2 COL3
## ROW1    1    2    3
## ROW2    4    5    6
```

```r
attributes(mat)
```

```
## $dim
## [1] 2 3
##
## $dimnames
## $dimnames[[1]]
## [1] "ROW1" "ROW2"
##
## $dimnames[[2]]
## [1] "COL1" "COL2" "COL3"
```

- Puedes preguntarle a R el número de filas o columnas de una matriz con `nrow()` y `ncol()`, o dim.

```
nrow(mat)
```

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

```
ncol(mat)
```

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

```
dim(mat)
```

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

- La longitud de una matriz corresponde al número de elementos en ella, como un vector atómico.

```
length(mat)
```

```
## [1] 6
```

- Si dos matrices tienen el mismo número de filas, se pueden unir con `cbind()`.

```
mat1 <- matrix(1:10, 5); mat2 <- matrix(1:15, 5)
cbind(mat1, mat2)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    6    1    6   11
## [2,]    2    7    2    7   12
## [3,]    3    8    3    8   13
## [4,]    4    9    4    9   14
## [5,]    5   10    5   10   15
```

- Si dos matrices tienen el mismo número de columnas, se pueden unir con `rbind()`.

```
mat3 <- matrix(letters[1:10], 2, 5); mat4 <- matrix(1:15, 3, 5)
rbind(mat3, mat4)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,] "a"  "c"  "e"  "g"  "i"
## [2,] "b"  "d"  "f"  "h"  "j"
## [3,] "1"  "4"  "7"  "10" "13"
## [4,] "2"  "5"  "8"  "11" "14"
## [5,] "3"  "6"  "9"  "12" "15"
```

- Se aplican las mismas reglas de coerción que con los vectores atómicos.

- Las matrices pueden tranponerse con la función `t()`.

```
mat <- matrix(rep(c(T,F), 10), 2)
mat
```

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

```
t(mat)
```

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

- Cuando la matriz es numérica (*integer* o *double*) or *logical* se pueden sumar los elementos u obtener su promedio por columna o fila con `colSums()`, `colMeans()`, `rowSums()` y `rowMeans()`.

```
colSums(mat)
```

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

```
colMeans(mat)
```

```
##  [1] 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
```

```
rowSums(mat)
```

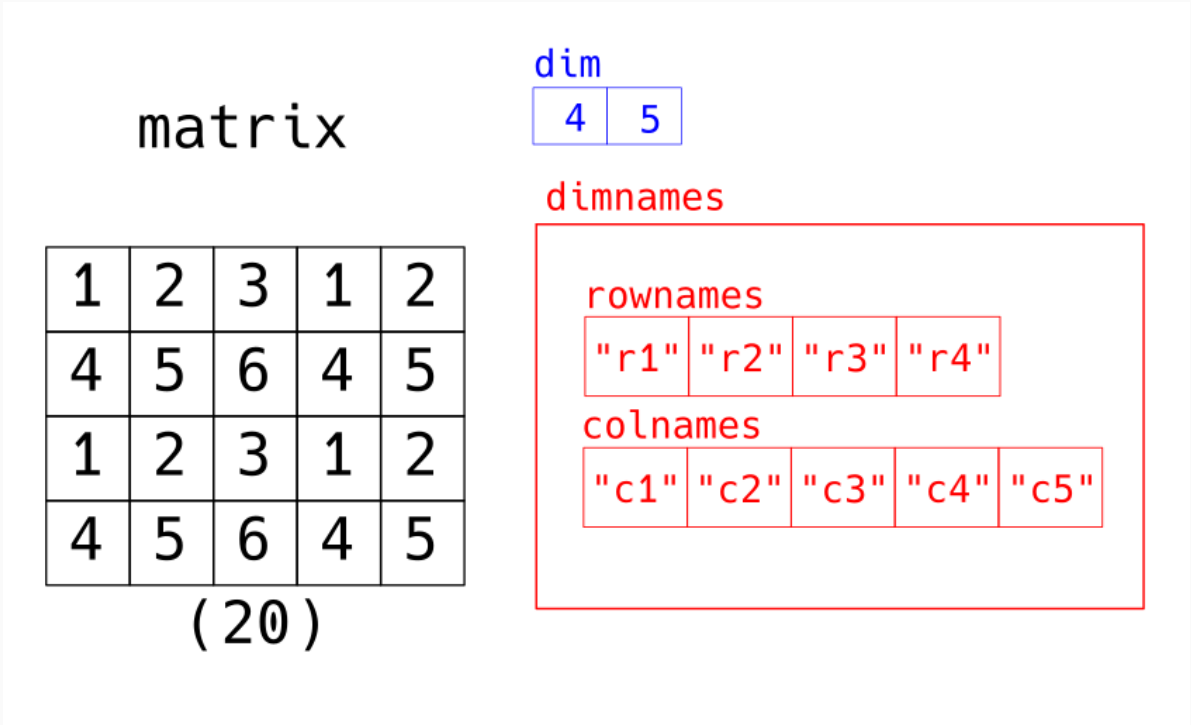
```
## [1] 10  0
```

```
rowMeans(mat)
```

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

En resumen...

- Una matriz es un vector atómico de dos dimensiones indicadas por el atributo *dim* (número de filas y columnas); *dim* es un vector de tipo *integer* y longitud 2.
- Las columnas y filas pueden estar nombradas con el atributo *dimnames* (nombre de las dimensiones); *dimnames* es una lista con dos vectores de tipo *character*.



Factores, *data frames* y *tibbles*

Atributo *class*

factor (fct)

- Vectores que se utilizan para almacenar datos categóricos.
- Se crean con la función `factor()` a partir de un vector atómico.

```
x <- as.integer(c(1, 2, 3, 1, 2, 3))
fct_vector <- factor(x)
fct_vector
```

```
## [1] 1 2 3 1 2 3
## Levels: 1 2 3
```

- El tipo de un factor es *integer*, pero **no** se comporta como numérico.

```
typeof(fct_vector)
```

```
## [1] "integer"
```

```
fct_vector * 10
```

```
## Warning in Ops.factor(fct_vector, 10): '*' not meaningful for
## factors

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

- Tiene dos atributos: *levels* y *class factor*.

```
attributes(fct_vector)
```

```
## $levels
## [1] "1" "2" "3"
##
## $class
## [1] "factor"
```

- Se pueden checar el número de *levels* con `nlevels()` y sus valores con `levels()`

```
nlevels(fct_vector)
```

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

```
levels(fct_vector)
```

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

- El atributo *levels* es un vector de tipo *character* que contiene los valores (categorías) posibles, por *default* en orden alabético.
- Se puede especificar o cambiar el orden de los levels, y las etiquetas, con `factor()`.

```
x
```

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

```
fct_vector <- factor(x,
                     levels = c(3, 2, 1),
                     labels = c("three", "two", "one"))
fct_vector
```

```
## [1] one   two   three one   two   three
## Levels: three two one
```

- El orden y las etiquetas de los `levels()` es relevante al momento de crear gráficas.

data.frame

- Los *data frames* se crean con la función `data.frame()` a partir de vectores atómicos (de "una dimensión") de una misma longitud; estos vectores pueden ser de diferentes tipos.

```
df <- data.frame(int_vector, dbl_vector, chr_vector, lgl_vector, fct_vector)
df
```

```
##   int_vector dbl_vector      chr_vector lgl_vector fct_vector
## 1          1        NaN               1       TRUE        one
## 2       4000      4e-05                       FALSE        two
## 3         NA     -3e+00 frase o palabra          NA      three
## 4          0        Inf            <NA>       FALSE        one
## 5         -5      5e+00              NA       TRUE        two
## 6        600         NA             200       TRUE      three
```

```
str(df)
```

```
## 'data.frame':    6 obs. of  5 variables:
##  $ int_vector: int  1 4000 NA 0 -5 600
##  $ dbl_vector: num  NaN 4e-05 -3e+00 Inf 5e+00 ...
##  $ chr_vector: chr  "1" "" "frase o palabra" NA ...
##  $ lgl_vector: logi  TRUE FALSE NA FALSE TRUE TRUE
##  $ fct_vector: Factor w/ 3 levels "three","two",..: 3 2 1 3 2 1
```

- Un *data frame* tiene tres atributos:
    1. ***names.*** Nombres de las columnas.
    2. ***class***. Tiene la clase *data.frame*.
    3. ***row.names***. Nombres de las filas.

```
attributes(df)
```

```
## $names
## [1] "int_vector" "dbl_vector" "chr_vector" "lgl_vector"
## [5] "fct_vector"
##
## $class
## [1] "data.frame"
##
## $row.names
## [1] 1 2 3 4 5 6
```

- La longitud del *data frame* es igual a su número de columnas y su tipo es *list*.

```
length(df)
```

```
## [1] 5
```

```
typeof(df)
```

```
## [1] "list"
```

- Al igual que una matriz, un *data.frame* tiene dos dimensiones: número de filas y número de columnas.

```
dim(df); nrow(df); ncol(df)
```

```
## [1] 6 5
```

```
## [1] 6
```

```
## [1] 5
```

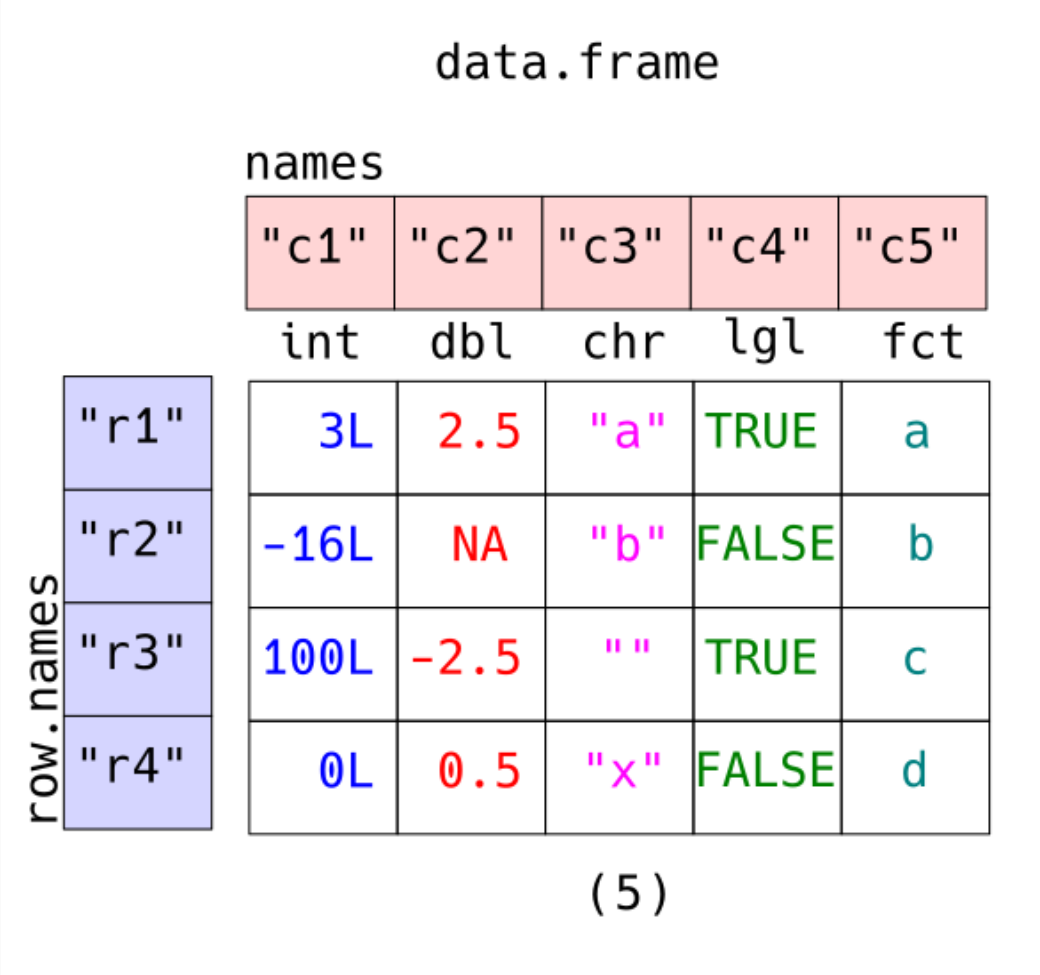
- Los nombres de las filas pueden modificarse con `row.names()` y los nombres de las columnas con `names()`. También con `dimnames()`.

```
row.names(df) <- letters[1:6] # o rownames()
names(df) <- LETTERS[1:5] # o colnames()
df
```

```
##      A      B                C     D     E
## a    1    NaN                1  TRUE   one
## b 4000  4e-05                  FALSE   two
## c   NA -3e+00 frase o palabra    NA three
## d    0    Inf             <NA> FALSE   one
## e   -5  5e+00               NA  TRUE   two
## f  600     NA              200  TRUE three
```

En resumen...

- En la práctica, un *data frame* funciona como una tabla.
- Un *data frame* tiene características de matriz y de lista.



tibble

- Un *tibble* es muy similar a un *data frame*.
- Se crea con la función `tibble()` del paquete **tibble** (si instalaste el Tydiverse, ya tienes instalado Tibble):

```
(
  tbl <- tibble::tibble(int_vector, dbl_vector,
                        chr_vector, lgl_vector,
                        fct_vector)
  )
```

```
## # A tibble: 6 x 5
##   int_vector dbl_vector chr_vector        lgl_vector fct_vector
##        <int>      <dbl> <chr>             <lgl>      <fct>
## 1          1  NaN       "1"               TRUE       one
## 2       4000    0.00004 ""                FALSE      two
## 3         NA   -3       "frase o palabra" NA         three
## 4          0  Inf        <NA>             FALSE      one
## 5         -5    5       "NA"              TRUE       two
## 6        600   NA       "200"             TRUE       three
```

```
class(tbl)
```

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

```
str(tbl)
```

```
## tibble [6 × 5] (S3: tbl_df/tbl/data.frame)
##  $ int_vector: int [1:6] 1 4000 NA 0 -5 600
##  $ dbl_vector: num [1:6] NaN 4e-05 -3e+00 Inf 5e+00 ...
##  $ chr_vector: chr [1:6] "1" "" "frase o palabra" NA ...
##  $ lgl_vector: logi [1:6] TRUE FALSE NA FALSE TRUE TRUE
##  $ fct_vector: Factor w/ 3 levels "three","two",..: 3 2 1 3 2 1
```

- Por default, un *tibble* solo imprime las primeras 10 filas y te indica cuantas más contiene el data set.
- Cuando no todas las columnas alcanzan a mostrarse en pantalla, solo imprime los nombres de las faltantes en la parte de abajo (puedes ver el objeto completo con `View()`).
- Debajo del nombre de cada columna se indica el tipo de vector de la columna.
- Los tibbles no usan *row.names*.
- El resultado applicar un subset con `[]` a un *tibble* siempre es un *tibble*.

Veamos un ejemplo con dataset más grande...

- Puedes transformar un *data frame* en un *tibble* con `as_tibble()`

```
tbl_mtcars <- tibble::as_tibble(mtcars) # Datos precargados en R
tbl_mtcars
```

```
## # A tibble: 32 x 11
##      mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1  21       6  160    110  3.9   2.62  16.5     0     1     4
##  2  21       6  160    110  3.9   2.88  17.0     0     1     4
##  3  22.8     4  108     93  3.85  2.32  18.6     1     1     4
##  4  21.4     6  258    110  3.08  3.22  19.4     1     0     3
##  5  18.7     8  360    175  3.15  3.44  17.0     0     0     3
##  6  18.1     6  225    105  2.76  3.46  20.2     1     0     3
##  7  14.3     8  360    245  3.21  3.57  15.8     0     0     3
##  8  24.4     4  147.    62  3.69  3.19  20       1     0     4
##  9  22.8     4  141.    95  3.92  3.15  22.9     1     0     4
## 10  19.2     6  168.   123  3.92  3.44  18.3     1     0     4
## # … with 22 more rows, and 1 more variable: carb <dbl>
```

- También puedes construir un *tibble* con `tribble()`, pero a veces no es muy práctico:

```
tbl_tr <- tibble::tribble(
  ~x, ~y, ~z,
  1, 2, "a",
  3, 4, "b",
  5, 6, "c",
  7, 8, "d",
  9, 10, "e`"
)

tbl_tr
```

```
## # A tibble: 5 x 3
##       x     y z
##   <dbl> <dbl> <chr>
## 1     1     2 a
## 2     3     4 b
## 3     5     6 c
## 4     7     8 d
## 5     9    10 e`
```

Importar y exportar *data frames* y *tibbles*

read.table()

- Para importar debes pasar el **path** y nombre del archivo a la función `read.table()`. Puede ser una liga de internet.
- Importemos el dataset de *palmer penguins*.

```
penguin_filename <- paste0("https://raw.githubusercontent.com/",
       "rfordatascience/tidytuesday/master/data/",
       "2020/2020-07-28/penguins.csv")
penguin_data <- read.table(penguin_filename, sep = ",", header = TRUE,
                           stringsAsFactors = FALSE, comment.char = "",
                           check.names = FALSE)
head(penguin_data, 2)
```

```
##   species    island bill_length_mm bill_depth_mm flipper_length_mm
## 1  Adelie Torgersen           39.1          18.7               181
## 2  Adelie Torgersen           39.5          17.4               186
##   body_mass_g    sex year
## 1        3750   male 2007
## 2        3800 female 2007
```

- Hay varias variantes de `read.table()`, como `read.csv()`, etc.
- Una vez importado el dataset se puede convertir en una matriz con `as.matrix()` o en un tibble con `as_tibble()`.

readr::read_csv()

- Se puede importar directamente una tabla de datos como tibble con las funciones `read_csv()` o `read_tsv()`, etc. del paquete ***readr*** (viene con el tidyverse).

```
tbl_penguins <- readr::read_csv(penguin_filename)
tbl_penguins
```

```
## # A tibble: 344 x 8
##    species island bill_length_mm bill_depth_mm flipper_length_…
##    <chr>   <chr>           <dbl>         <dbl>             <dbl>
##  1 Adelie  Torge…           39.1          18.7               181
##  2 Adelie  Torge…           39.5          17.4               186
##  3 Adelie  Torge…           40.3          18                 195
##  4 Adelie  Torge…           NA            NA                  NA
##  5 Adelie  Torge…           36.7          19.3               193
##  6 Adelie  Torge…           39.3          20.6               190
##  7 Adelie  Torge…           38.9          17.8               181
##  8 Adelie  Torge…           39.2          19.6               195
##  9 Adelie  Torge…           34.1          18.1               193
## 10 Adelie  Torge…           42            20.2               190
## # … with 334 more rows, and 3 more variables: body_mass_g <dbl>,
## #   sex <chr>, year <dbl>
```

`write.table()` y `readr::write_csv()`

- Los datos se pueden exportar como tablas con cuaquiera de las funciones `write*()`:

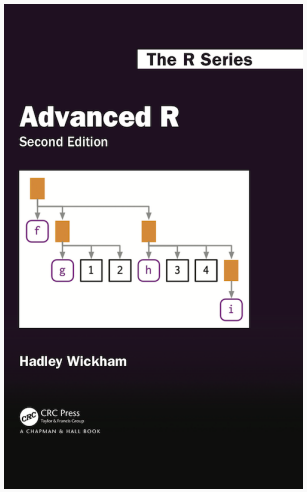
```
write.table(penguin_data, file = "penguins1.tsv", sep = "\t",
            row.names = FALSE, quote = FALSE)
readr::write_csv(penguin_data, "penguins2.csv")
```

Actividades recomendadas

- ¿Qué información nos dan las funciones `mode()`, `typeof()` y `class()`?
- Aplica esas funciones a las diferentes estructuras de datos que vimos en esta presentación y compara.
- ¿Para qué sirven y cómo se utilizan las funciones `setNames()` y `unname()`?
- ¿Que otros tipos de vectores atómicos existen?
- ¿Qué vectores almacenan fechas y horas en R?
- ¿Cómo se importan datos desde una hoja de cálculo de Excel?

Bibliografía recomendada

Advanced R - Hadley Wickham



Capítulo 3 - Vectores

https://adv-r.hadley.nz/vectors-chap.html