# Manejo de Datos con R

## EP7144 - Técnicas de Minería de Datos

Mg. Enver Gerald Tarazona Vargas

enver.tarazona@pucp.edu.pe

Escuela de Post-Grado

**Universidad Nacional Agraria La Molina (UNALM)**

# Resumen

# Resumen

1. Manejo Básico de R
   - Instalación
   - Funciones básicas
   - Ayuda
2. Objetos en R
   - Vectores
   - Matrices
   - Arrays
   - Listas
   - Marco de Datos
   - Tipos de datos
   - Trabajando con Factores
   - Datos Perdidos

# ¿Qué es R? I

- R es un lenguaje computacional de alto nivel y un programa para realizar análisis estadístico y gráficos.
  - Permite aplicar una variedad de métodos estadísticos básicos y avanzados.
  - Produce gráficos de alta calidad.
  - R es un lenguaje de programación, es decir, podemos escribir nuevas funciones y extender el uso del R.
- R fue inicialmente escrito Ross Ihaka y Robert Gentleman del Departamento de Estadística de la Universidad de Auckland en Auckland, Nueva Zelanda.
- R es un software open source que es mantenido por muchos contribuyentes. El *R Core Team* está conformado por unos 17 programadores que son responsables de modificar el código fuente del R.
- El sitio web oficial del R es: `http://www.R-project.org`

# Instalación de R I

- R puede ser instado en Windows, Mac o Linux.
- Para instalar el sistema base, visitar el sitio web de R y seguir las instrucciones de instalación.
- Adicionalmente al sistema base existen una serie de paquetes adicionales de contribuyentes.
- Un paquete es una colección de funciones, ejemplos y documentación que usualmente están enfocados en realizar una tarea específica.
- El sistema base contiene solamente algunos paquetes. Para instalar un paquete adicional, por ejemplo `agricolae`, se debe estar conectado en la internet y escribir:

  ```
  > install.packages("agricolae")
  ```

  Si no ha sido configurado antes, aparecerá una ventana para seleccionar el mirror más cercano, luego todo será automático.

# Instalación de R II

- Antes de usar los contenidos de un paquete es necesario cargarlo,

  ```
  > library(agricolae)
  ```

- Ver el sitio web de R website para una lista completa de los paquetes de contribuyentes.

# La consola del R I

- En la consola del R es dónde se realizan los cálculos.
- Cuando una expresión se introduce en la consola, es posteriormente evaluada. Dependiendo de la expresión, el sistema puede responder mediante la salida de resultados a la consola o la creación de un gráfico en una ventana nueva. Luego otra expresión es ingresada y evaluada.
- Una sesión en R es la interacción entre el usuario y el sistema.
- Para obtener la última expresión ingresada usar la techa de flecha hacia arriba.
- Para obtener el valor de la última expresión evaluada tipear `.Last.value`.
- Presionar Esc para detener la evaluación de la expresión que se está evaluando.

# R cómo calculadora I

- Si se introduce una expresión matemática, el resultado se mostrará en la consola.

| | | | | | | |
|---|---|---|---|---|---|---|
| **Binary Operators** | + | - | * | / | ^ | %% |
| **Math Functions** | abs | sqrt | log | exp | log10 | factorial |
| **Trig Functions** | sin | cos | tan | asin | acos | atan |
| **Rounding** | round | ceiling | floor | trunc | signif | zapsmall |
| **Math Quantities** | Inf | -Inf | NaN | pi | exp(1) | 1i |

%% is for modular arithmetic

```
> 5 %% 4
[1] 1
> log(2)
[1] 0.6931472
> cos(pi)
[1] -1
> ceiling(3.2)
```

# R cómo calculadora II

```
[1] 4
> 0/0
[1] NaN
> 1/Inf
[1] 0
```

# Objetos y Funciones I

- Lo que normalmente hacemos es crear objetos y aplicar funciones a los objetos (las funciones también se consideran objetos).
- Asignar a un objeto un nombre x:

```
x <- objecto (objecto -> x)
x = objecto
```

- Llamar una función:

```
nombre función (lista de argumentos separados por comas)
```

  - Cada función tiene un conjunto formal de argumentos con algunos valores por defecto. Estos pueden ser encontrados en la documentación de la función.
  - El llamado a una función puede incluir cualquier subconjunto de la lista completa de argumentos.
  - Para especificar un argumento en particular usar el nombre del argumento.

# Objetos y Funciones II

- Los argumentos no tienen que ser nombrados si están inscritos en el mismo orden que la lista de argumentos formales de la función. Sin embargo, para que su código sea más fácil de entender por lo general es una buena idea nombrar a sus argumentos.
- R **DISTINGUE EL USO DE MAYÚSCULAS Y MINÚSCULAS**.

# Ejemplo - Asignar Objetos y Llamado de Funciones I

- Supongamos que queremos encontrar la media de un conjunto de números. Primero se asigna el vector de números con un nombre x y luego se llama a la función `mean()`.

```
> x <- c(0,5,7,9,1,2,8)
> x
[1] 0 5 7 9 1 2 8
> mean(x)
[1] 4.571429
> X
Error: object 'X' not found
```

- Ahora supongamos que se quiere ordenar un vector y que los números estén en orden descendente. Por defecto R ordena de modo ascendente, por lo que se tiene que cambiar el argumento decreasing por TRUE (el valor por defecto es FALSE).

# Ejemplo - Asignar Objetos y Llamado de Funciones II

```
> y <- c(4,2,0,9,5,3,10)
> y
[1]  4  2  0  9  5  3 10
> sort(y)
[1]  0  2  3  4  5  9 10
> sort(y, decreasing=TRUE)
[1] 10  9  5  4  3  2  0
```

# Editor de Scripts I

- El editor de scripts es usado para escribir programas en R.
- Para abrir un nuevo script, hacer clic en File>New Script.
- La forma más sencilla para correr un código es usando teclas de atajo.
  - Para ejecutar una parte del programa, seleccionar las lineas que se desean ejecutar y precionar Ctrl+R.
  - Para ejecutar todo el programa, seleccionar todo el código, Ctrl+A y ahí Ctrl+R.
- Cuándo se guarda un archivo de script en R, se debe de incluir la extensión .R, R no lo añade de forma automática.
- No es necesario usar punto y coma al final de una linea, excepto en el caso de ingresar más de una expresión.
- Es posible escribir una expresión larga en varias líneas, pero hay que tener cuidado de que R no evalúe una porción de código no terminado como una expresión completa.
- Para comentar una linea usar #. No hay comentarios por bloque en R.

# Documentación parte 1 I

- Conozco la función media (mean) pero quiero saber mas de ella.

  ```
  > ?mean
  > help(mean)
  ```

- Necesito mas información acerca de la media.

  ```
  > help.search("mean")
  ```

- Necesito un ejemplo que involucre la media.

  ```
  > example(mean)
  ```

# Documentación parte 2 I

- Supongamos que estamos interesados en hacer un diagrama de flujo. Después de navegar por la lista de paquetes aportados en el sitio web de R decidimos utilizar: diagram.
- Primero bajamos, instalamos y extraemos el paquete

```
> install.packages("diagram")
> library(diagram)
```

- Necesito una lista de funciones con breves descripciones del paquete "diagram".

```
> help(package=diagram)
```

# Workspace and Options() I

- Workspace es el lugar donde se almacena todo lo que realizas en una sesión de R.
  - Si quieres listar las variables que has creado: ls()
  - Si quieres ver que librarias están descargadas: search()
  - Si quieres ver que librarias instalaste: library()
  - Si quieres eliminar un objeto:rm(nombre del objeto)
- la función options() determina como R calcula y muestra los resultados.

```
> options(digits=15) # Cambia a 15 digitos
> pi
[1] 3.14159265358979 # R por defecto muestra 7)
> options(defaults) # Regresa a la normalidad
> getOption("digits")
[1] 7
```

# Vectores I

- Un vector es una colección ordenada de objetos del mismo tipo.
- La función `c(...)` concatena argumentos para formar un vector.
- Para crear un vector con un patrón determinado:
  - `:` Secuencia de enteros.
  - `seq()` Secuencia en general.
  - `rep()` Vector de elementos repetidos.

```
> v1 <- c(2.5, 4, 7.3, 0.1)
> v1
[1] 2.5 4.0 7.3 0.1
> v2 <- c("A", "B", "C", "D")
> v2
[1] "A" "B" "C" "D"
> v3 <- -3:3
```

# Vectores II

```
> v3
[1] -3 -2 -1  0  1  2  3
> seq(0, 2, by=0.5)
[1] 0.0 0.5 1.0 1.5 2.0
> seq(0, 2, len=6)
[1] 0.0 0.4 0.8 1.2 1.6 2.0
> rep(1:5, each=2)
 [1] 1 1 2 2 3 3 4 4 5 5
> rep(1:5, times=2)
 [1] 1 2 3 4 5 1 2 3 4 5
```

# Acceso a Elementos de Vectores I

- Usar [ ] con un vector/escalar de posiciones para acceder a los elementos de un vector.
- Incluir un signo menos antes del vector/escalar para remover elementos

```
> x <- c(4, 7, 2, 10, 1, 0)
> x[4]
[1] 10
> x[1:3]
[1] 4 7 2
> x[c(2,5,6)]
[1] 7 1 0
> x[-3]
[1]  4  7 10  1  0
> x[-c(4,5)]
```

# Acceso a Elementos de Vectores II

```
[1] 4 7 2 0
> x[x>4]
[1]  7 10
> x[3] <- 99
> x
[1]  4  7 99 10  1  0
```

# which() y match() I

- Funciones adicionales que retornarán el índice de un vector.
  - `which()` Indices de un vector lógico donde la condición es verdadera.
  - `which.max()` Ubicación del mayor (primer) elemento de un vector numérico.
  - `which.min()` Ubicación del menor (primer) elemento de un vector numérico.
  - `match()` Primera posición de un elemento en un vector.

```
> x <- c(4, 7, 2, 10, 1, 0)
> x>=4
[1]  TRUE  TRUE FALSE  TRUE FALSE FALSE
> which(x>=4)
[1] 1 2 4
> which.max(x)
[1] 4
```

# which() y match() II

```
> x[which.max(x)]
[1] 10
> max(x)
[1] 10
> y <- rep(1:5, times=5:1)
> y
 [1] 1 1 1 1 1 2 2 2 2 3 3 3 4 4 5
> match(1:5, y)
[1]  1  6 10 13 15
> match(unique(y), y)
[1]  1  6 10 13 15
```

# Operaciones con Vectores I

- Cuando los vectores son usados en expresiones matemáticas, las operaciones son realizadas con cada elemento.

```
> x <- c(4,7,2,10,1,0)
> y <- x^2 + 1
> y
[1]  17  50   5 101   2   1
> x*y
[1]   68  350   10 1010    2    0
```

# Funciones con Vectores I

| | | |
|---|---|---|
| `sum(x)` | `prod(x)` | Sum/product of the elements of `x` |
| `cumsum(x)` | `cumprod(x)` | Cumulative sum/product of the elements of `x` |
| `min(x)` | `max(x)` | Minimum/Maximum element of `x` |
| `mean(x)` | `median(x)` | Mean/median of `x` |
| `var(x)` | `sd(x)` | Variance/standard deviation of `x` |
| `cov(x,y)` | `cor(x,y)` | Covariance/correlation of `x` and `y` |
| `range(x)` | | Range of `x` |
| `quantile(x)` | | Quantiles of `x` for the given probabilities |
| `fivenum(x)` | | Five number summary of `x` |
| `length(x)` | | Number of elements in `x` |
| `unique(x)` | | Unique elements of `x` |
| `rev(x)` | | Reverse the elements of `x` |
| `sort(x)` | | Sort the elements of `x` |
| `which()` | | Indices of `TRUE`s in a logical vector |
| `which.max(x)` | `which.min(x)` | Index of the max/min element of `x` |
| `match()` | | First position of an element in a vector |
| `union(x, y)` | | Union of `x` and `y` |
| `intersect(x, y)` | | Intersection of `x` and `y` |
| `setdiff(x, y)` | | Elements of `x` that are not in `y` |
| `setequal(x, y)` | | Do `x` and `y` contain the same elements? |

# La función `vector()` I

- La función vector, que tiene dos argumentos `mode` y `length`, crea un vector cuyos elementos pueden ser de tipo numérico, lógico o caracter dependiendo del argumento especificado en mode (0, FALSE o " " respectivamente).
- Las siguientes funciones tienen exactamente el mismo efecto y tienen un solo argumento (la longitud del vector): `numeric()`,`logical()`, y `character()`.

```
> vector(mode = "list",length = 2)
[[1]]
NULL

[[2]]
NULL
```

# Matrices I

- Una matriz es una generalización, en dos dimensiones, de un vector.
- Para crear una matriz:

```
matrix(data=NA, nrow=1, ncol=1, byrow = FALSE,
       dimnames = NULL)
```

`data`: Un vector con los datos para llenar la matriz; si los datos no contienen elementos suficientes para llenar la matriz, entonces estos son reciclados.
`nrow`: Número deseado de filas.
`ncol`: Número deseado de columnas.
`byrow`: Si es FALSE (defecto) la matriz es llenada por columnas, caso contrario por filas.
`dimnames`: (opcional) lista de longitud 2 dando los nombres de filas y columnas respectivamente, los nombres de la lista serán usados como nombres de las dimensiones.

# Matrices II

```
x <- matrix(c(5,0,6,1,3,5,9,5,7,1,5,3), nrow=3, ncol=4, byrow=TRUE,
+             dimnames=list(rows=c("r.1", "r.2", "r.3"),
+                           cols=c("c.1", "c.2", "c.3", "c.4")))
> x
     cols
rows  c.1 c.2 c.3 c.4
  r.1   5   0   6   1
  r.2   3   5   9   5
  r.3   7   1   5   3
> matrix(data=5, nr=2, nc=2)
     [,1] [,2]
[1,]    5    5
[2,]    5    5
> matrix(1:6, 2, 3)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> matrix(1:6, 2, 3, byrow=TRUE)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

# Matrices III

- Otra manera de crear una matriz es dando los valores apropiados al atributo `dim` (que inicialmente tiene valor NULL):

```
> x <- 1:15
> x
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
> dim(x)
NULL
> dim(x) <- c(5, 3)
> x
     [,1] [,2] [,3]
[1,]    1    6   11
[2,]    2    7   12
[3,]    3    8   13
[4,]    4    9   14
[5,]    5   10   15
```

# Acceso a Elementos de una Matriz I

- Acceder a los elementos de una matriz usando [ ] similar a los vectores, pero considerando 2 dimensiones.

```
> x <- matrix(c(5,0,6,1,3,5,9,5,7,1,5,3), nrow=3, ncol=4, byrow=TRUE)
> x
     [,1] [,2] [,3] [,4]
[1,]    5    0    6    1
[2,]    3    5    9    5
[3,]    7    1    5    3
> x[2,3] # Fila 2, Columna 3
[1] 9
> x[1,] # Fila 1
[1] 5 0 6 1
> x[,2] # Columna 2
[1] 0 5 1
> x[c(1,3),] # Filas 1 y 3, todas las columnas
     [,1] [,2] [,3] [,4]
[1,]    5    0    6    1
[2,]    7    1    5    3
```

# Acceso a Elementos de una Matriz II

- También podemos acceder a partes de una matriz mediante el uso de los nombres de fila o columna.
- A veces es mejor hacer referencia a una fila/columna por su nombre en lugar del índice numérico. Por ejemplo, si un programa agrega o permuta las columnas de una matriz entonces el índice numérico de las columnas puede cambiar. Como resultado,se podría hacer referencia a la columna equivocada de la nueva matriz si se utiliza el número de índice antigüo. Sin embargo, el nombre de cada columna no cambiará.
- Por ello es posible acceder a los elementos de una matriz con [ ] pero ahora usando el nombre de la fila o columna, con comillas, en lugar del índice numério.
- No se tiene que especificar los nombres cuando se crea una matriz. Para obtener o establecer los nombres de columna, fila o ambos para A:

```
colnames(A)
rownames(A)
dimnames(A)
```

- Es posible nombrar a los elementos del vector, `c("name.1"=1, "name.2"=2)`. Usar la función `names()` para obtener o establecer los nombres de los elementos del vector

# Acceso a Elementos de una Matriz III

```
> N <- matrix(c(5,8,3,0,4,1), nrow=2, ncol=3, byrow=TRUE)
> colnames(N) <- c("c.1", "c.2", "c.3")
> N
     c.1 c.2 c.3
[1,]   5   8   3
[2,]   0   4   1
> N[,"c.2"] # Columna llamada "c.2"
[1] 8 4
> colnames(N)
[1] "c.1" "c.2" "c.3"
> M <- diag(2)
> (MN <- cbind(M, N)) # Colocando la expresión en paréntesis
                      # mostrará los resultados
     c.1 c.2 c.3
[1,] 1 0   5   8   3
[2,] 0 1   0   4   1
> MN[,2] # Columna 2
[1] 0 1
> MN[,"c.2"] # Columna llamada "c.2"
[1] 8 4
```

# Operaciones con Matrices I

- Cuando se usan matrices en expresiones matemáticas se realizan las operaciones elemento por elemento.
- Para multiplicación matricial usar el operador %*%.
- Si se utiliza un vector en la multiplicación de matrices, será forzado a que sea una matriz fila o columna para hacer que los argumentos sean conformables. Usando %*% en dos vectores devolverá el producto interno ( %o %para el producto externo) como una matriz y no un escalar. Utilice c() o as.vector() para convertir a un escalar.

# Operaciones con Matrices II

```
> A <- matrix(1:4, nrow=2)
> B <- matrix(1, nrow=2, ncol=2)
> A*B
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> A%*%B
     [,1] [,2]
[1,]    4    4
[2,]    6    6
> y <- 1:3
> y%*%y
     [,1]
[1,]   14
> A/(y%*%y)
Error in A/(y %*% y) : non-conformable arrays
> A/c(y%*%y)
           [,1]      [,2]
[1,] 0.07142857 0.2142857
[2,] 0.14285714 0.2857143
```

# Funciones con Matrices I

| | |
|---|---|
| `t(A)` | Transpose of `A` |
| `det(A)` | Determinate of `A` |
| `solve(A, b)` | Solves the equation `Ax=b` for `x` |
| `solve(A)` | Matrix inverse of `A` |
| `MASS::ginv(A)` | Generalized inverse of `A` (MASS package) |
| `eigen(A)` | Eigenvalues and eigenvectors of `A` |
| `chol(A)` | Choleski factorization of `A` |
| `diag(n)` | Create a `n`×`n` identity matrix |
| `diag(A)` | Returns the diagonal elements of a matrix `A` |
| `diag(x)` | Create a diagonal matrix from a vector `x` |
| `lower.tri(A),upper.tri(A)` | Matrix of logicals indicating lower/upper triangular matrix |
| `apply()` | Apply a function to the margins of a matrix |
| `rbind(...)` | Combines arguments by rows |
| `cbind(...)` | Combines arguments by columns and |
| `dim(A)` | Dimensions of `A` |
| `nrow(A), ncol(A)` | Number of rows/columns of `A` |
| `colnames(A), rownames(A)` | Get or set the column/row names of `A` |
| `dimnames(A)` | Get or set the dimension names of `A` |

# Funciones con Matrices II

```
> m1 <- matrix(1, nr = 2, nc = 2)
> m2 <- matrix(2, nr = 2, nc = 2)
> rbind(m1, m2)
     [,1] [,2]
[1,]    1    1
[2,]    1    1
[3,]    2    2
[4,]    2    2
> cbind(m1, m2)
     [,1] [,2] [,3] [,4]
[1,]    1    1    2    2
[2,]    1    1    2    2
> rbind(m1, m2) %*% cbind(m1, m2)
     [,1] [,2] [,3] [,4]
[1,]    2    2    4    4
[2,]    2    2    4    4
[3,]    4    4    8    8
[4,]    4    4    8    8
> cbind(m1, m2) %*% rbind(m1, m2)
     [,1] [,2]
```

# Funciones con Matrices III

```
[1,]   10   10
[2,]   10   10
> diag(m1)
[1] 1 1
> diag(rbind(m1, m2) %*% cbind(m1, m2))
[1] 2 2 8 8
> diag(m1) <- 10
> m1
     [,1] [,2]
[1,]   10    1
[2,]    1   10
> diag(3)
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
> v <- c(10, 20, 30)
> diag(v)
     [,1] [,2] [,3]
[1,]   10    0    0
```

# Funciones con Matrices IV

```
[2,]    0   20    0
[3,]    0    0   30
> diag(2.1, nr = 3, nc = 5)
     [,1] [,2] [,3] [,4] [,5]
[1,]  2.1  0.0  0.0    0    0
[2,]  0.0  2.1  0.0    0    0
[3,]  0.0  0.0  2.1    0    0
```

# La función `apply()` I

- La función apply() es usada para aplicar funciones a los márgenes (filas o columnas) de matrices, arrays, o marcos de datos.

  `apply(X, MARGIN, FUN, ...)`

`X`: Una matrix, vector o marco de datos.
`MARGIN`: Vector de sub-índices indicando a que margen aplicar la función: 1=filas, 2=columnas, c(1,2)=filas y columnas
`FUN`: Función a ser aplicada.
`...`: Argumentos adicionales de FUN.

- Es posible usar funciones propias definidas por el usuario.

# La función apply() II

```
> (x <- matrix(1:12, nrow=3, ncol=4))
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> apply(x, 1, sum) # Total de filas
[1] 22 26 30
> apply(x, 2, mean) # Total de columnas
[1]  2  5  8 11
```

# Ejemplo: Simulando Datos de Sobrevivencia I

- La función apply() es muy útil para simular datos de sobrevivencia. Suponga que queremos simular $n = 10$ observaciones, dónde el tiempo de los eventos $T$ siguen una distribución exponencial con media $\lambda = 0.25$ y los tiempos de censura C están distribuidos uniformemente entre $0$ y $1$. Entonces los datos observados serán, $X = \text{mín}\,(T, C)$ y $\delta = I\,(T < C)$

```
> # Tamaño de muestra
> n = 10
> # Semilla
> set.seed(666)
> # Generar los tiempos para los eventos y censura (ver la ayuda para
> # entender como R parametriza la función exponencial)
> event <- rexp(n, 4)
> censor <- runif(n)
> # Seleccionar el tiempo mínimo y crear la variable indicadora
> time <- apply(cbind(censor, event), 1, min)
```

# Ejemplo: Simulando Datos de Sobrevivencia II

```
> index <- apply(cbind(censor, event), 1, which.min)-1
> cbind(event, censor, time, index) # Verificar
             event     censor        time index
 [1,] 0.137184245 0.03654720 0.036547196     0
 [2,] 0.491021973 0.89163741 0.491021973     1
 [3,] 0.034887371 0.48323641 0.034887371     1
 [4,] 0.002307458 0.46666453 0.002307458     1
 [5,] 0.137946540 0.98422408 0.137946540     1
 [6,] 0.878498816 0.60134555 0.601345547     0
 [7,] 0.652839522 0.03834435 0.038344347     0
 [8,] 0.380900670 0.14149569 0.141495691     0
 [9,] 0.518826066 0.80638553 0.518826066     1
[10,] 0.155628222 0.26668568 0.155628222     1
> data <- cbind(time, index) # Datos Simulados
```

# Arrays I

- Un array es una generalización multi-dimensional de un vector.
- Para crear un array:
  `array(data = NA, dim = length(data), dimnames = NULL)`

`data`: Un vector con los datos para llenar el array; si los datos no contienen elementos suficientes para llenar el arreglo, entonces los elementos son reciclados.
`dim`: Dimensión del array, un vector de longitud uno o mayor dadndo el índice máximo en cada dimensión.
`dimnames`: (opcional) Nombre de las dimensiones, lista con un componente por cada dimensión, ya sea NULL o un vector de caracteres de longitud dada por dim. La lista puede tener nombres, y la lista de nombres será usada para nombrar a las dimensiones.

- Los valores son ingresados por columnas.

# Arrays II

- Del mismo modo que con vectores y matrices, cuando los arrays son usados en expresiones matemáticas las operaciones son realizadas elemento por elemento.
- Del mismo modo que con los vectores y matrices, los elementos de un array deben ser todos del mismo tipo (numérico, caracter, lógico, etc.)
- Ejemplo de un array $2 \times 3 \times 2$

```
> w <- array(1:12, dim=c(2,3,2),
+     dimnames=list(c("A","B"), c("X","Y","Z"), c("N","M")))
> w
, , N

  X Y Z
A 1 3 5
B 2 4 6

, , M
  X  Y  Z
A 7  9 11
B 8 10 12
```

# Acceso a elementos de un Array I

- Se accede a los elementos de un array usando [ ], del mismo modo que con vectores y matrices, sólo que con más dimensiones.

```
> w <- array(1:12, dim=c(2,3,2),
+            dimnames=list(c("A","B"), c("X","Y","Z"), c("N","M")))
> w[2,3,1] # Fila 2, Columna 3, Matriz 1
[1] 6
> w[,"Y",] # Columna con encabezado "Y"
  N  M
A 3  9
B 4 10
> w[1,,] # Fila 1
  N  M
X 1  7
Y 3  9
Z 5 11
> w[1:2,,"M"] # Fila 1 y 2, Matriz "M"
   X  Y  Z
A  7  9 11
B  8 10 12
```

# Funciones con Arrays I

| | |
|---|---|
| `apply()` | Apply a function to the margins of an array |
| `aperm()` | Transpose an array by permuting its dimensions |
| `dim(x)` | Dimensions of `x` |
| `dimnames(x)` | Get or set the dimension names of `x` |

# apply() I

- Se puede usar la función `apply()` para más de una dimensión.
- Para un array de 3 dimensiones existen tres márgenes para aplicar la función: 1=filas, 2=columnas, y 3=matrices.

```
> # Suma de Columnas
> apply(w, 2, sum)
 X  Y  Z
18 26 34
> # Suma de Filas por Matrices
> apply(w, c(1,3), sum)
```

# Listas I

- Una lista es una forma general de un vector, donde los elementos no precisan ser del mismo tipo o dimensión.
- La función `list(...)` crea una lista de argumentos.
- Los argumentos tienen la forma de $nombre = valor$. Los argumentos pueden ser especificados con o sin nombres.

```
> x <- list(num=c(1,2,3), "Nick", identity=diag(2))
> x
$num
[1] 1 2 3

[[2]]
[1] "Nick"

$identity
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

# Acceso a Elementos de una Lista I

- Se accede a los elementos de una lista usando [ ] del mismo modo que [[ ]] o $.

```
> x <- list(num=c(1,2,3), "Nick", identity=diag(2))
> x[[2]] # Segundo elemento de x
[1] "Nick"
> x[["num"]] # Elemento llamado "num"
[1] 1 2 3
> x$identity # Elemento llamado "identity"
     [,1] [,2]
[1,]    1    0
[2,]    0    1
> x[[3]][1,] # Primera fila del tercer elemento
[1] 1 0
> x[1:2] # Crear una sublista con los primeros dos elementos
$num
[1] 1 2 3

[[2]]
[1] "Nick"
```

# Funciones con Listas I

| | |
|---|---|
| `lapply()` | Apply a function to each element of a list, returns a list |
| `sapply()` | Same as `lapply()`, but returns a vector or matrix by default |
| `vapply()` | Similiar to `sapply()`, but has a pre-specified type of return value |
| `replicate()` | Repeated evaluation of an expression, useful for replicating lists |
| `unlist(x)` | Produce a vector of all the components that occur in `x` |
| `length(x)` | Number of objects in `x` |
| `names(x)` | Names of the objects in `x` |

# Ejemplo: `lapply()`, `sapply()`, `vapply()` I

- Primero generamos una lista $L$ de siete vectores distintos. Luego generamos un resumen descriptivo para cada vector de $L$ usando `lapply()`, `sapply()` y `vapply()`.

```
> # Lista con siete vectores distintos
> L <- lapply(3:9, seq)
> L
[[1]]
[1] 1 2 3

[[2]]
[1] 1 2 3 4

[[3]]
[1] 1 2 3 4 5

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

# Ejemplo: lapply(), sapply(), vapply() II

```
[[5]]
[1] 1 2 3 4 5 6 7

[[6]]
[1] 1 2 3 4 5 6 7 8

[[7]]
[1] 1 2 3 4 5 6 7 8 9

> # Calcular las estadísticas de resumen
> lapply(L, fivenum)
[[1]]
[1] 1.0 1.5 2.0 2.5 3.0

[[2]]
[1] 1.0 1.5 2.5 3.5 4.0

[[3]]
[1] 1 2 3 4 5

[[4]]
```

# Ejemplo: lapply(), sapply(), vapply() III

```
[1] 1.0 2.0 3.5 5.0 6.0

[[5]]
[1] 1.0 2.5 4.0 5.5 7.0

[[6]]
[1] 1.0 2.5 4.5 6.5 8.0

[[7]]
[1] 1 3 5 7 9

> sapply(L, fivenum)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]  1.0  1.0    1  1.0  1.0  1.0    1
[2,]  1.5  1.5    2  2.0  2.5  2.5    3
[3,]  2.0  2.5    3  3.5  4.0  4.5    5
[4,]  2.5  3.5    4  5.0  5.5  6.5    7
[5,]  3.0  4.0    5  6.0  7.0  8.0    9
> vapply(L, fivenum, c(Min.=0, "1st Quart"=0,
        Median=0, "3rd Qu"=0, Max.=0))
          [,1] [,2] [,3] [,4] [,5] [,6] [,7]
```

# Ejemplo: `lapply()`, `sapply()`, `vapply()` IV

```
Min.         1.0  1.0    1  1.0  1.0  1.0    1
1st Quart    1.5  1.5    2  2.0  2.5  2.5    3
Median       2.0  2.5    3  3.5  4.0  4.5    5
3rd Qu       2.5  3.5    4  5.0  5.5  6.5    7
Max.         3.0  4.0    5  6.0  7.0  8.0    9
```

- Dado que 3:9 no es una lista, R llama a as.list(3:9) el cual convierte el vector 3:9 a una lista de longitud $7$ donde cada número es un elemento de $L$. Recordar que seq(n) es similar a 1:n.

# Marco de Datos (Dataframes) I

- R denomina marco de datos (dataframes) a las bases de datos (datasets).
- Un dataframe tiene una estructura similar a una matriz, donde las columnas pueden ser de diversos tipos. Es posible pensar en un dataframe como una lista. Cada columna es un elemento de la lista y cada elemento tiene la misma longitud.
- El dataframe es la estructura de datos fundamental usada en R para realizar análisis estadístico.
- Se crea automaticamente cuando se leen datos desde un archivo.

# Marco de Datos (Dataframes) II

```
> x <- 1:4; n <- 10; M <- c(10, 35); y <- 2:4
> data.frame(x, n)
  x  n
1 1 10
2 2 10
3 3 10
4 4 10
> data.frame(x, M)
  x  M
1 1 10
2 2 35
3 3 10
4 4 35
> data.frame(x, y)
Error in data.frame(x, y) :
  arguments imply differing number of rows: 4, 3
```

# Funciones de Prueba y Conversión I

| Type | Testing | Coercing |
|---|---|---|
| Array | `is.array()` | `as.array()` |
| Character | `is.character()` | `as.character()` |
| Dataframe | `is.data.frame()` | `as.data.frame()` |
| Factor | `is.factor()` | `as.factor()` |
| List | `is.list()` | `as.list()` |
| Logical | `is.logical()` | `as.logical()` |
| Matrix | `is.matrix()` | `as.matrix()` |
| Numeric | `is.numeric()` | `as.numeric()` |
| Vector | `is.vector()` | `as.vector()` |

# Numérico I

- Los datos numéricos en R pueden ser enteros(integer) o reales(double)
- format() es una función genérica que se usa con otros objetos.

```
> # trim - si es FALSE genera anchos comunes
> format(c(1,10,100,1000), trim = FALSE)
[1] " 1" " 10" " 100" "1000"
> format(c(1,10,100,1000), trim = TRUE)
[1] "1" "10" "100" "1000"
> # nsmall - número de dígitos después del punto decimal
> format(13.7, nsmall = 3)
[1] "13.700"
> # scientific - genera notación científica
> format(2^16, scientific = TRUE)
[1] "6.5536e+04"
```

# Operadores Lógicos I

# Operadores Lógicos II

| | |
|---|---|
| `!x` | NOT `x` |
| `x & y` | `x` AND `y` elementwise, returns a vector |
| `x && y` | `x` AND `y`, returns a single value |
| `x \| y` | `x` OR `y` elementwise, returns a vector |
| `x \|\| y` | `x` OR `y`, returns a single value |
| `xor(x,y)` | Exclusive OR of `x` and `y`, elementwise |
| `x %in% y` | `x` IN `y` |
| `x < y` | $x < y$ |
| `x > y` | $x > y$ |
| `x <= y` | $x \leq y$ |
| `x >= y` | $x \geq y$ |
| `x == y` | $x = y$ |
| `x != y` | $x \neq y$ |
| `isTRUE(x)` | `TRUE` if `x` is `TRUE` |
| `all(...)` | `TRUE` if all arguments are `TRUE` |
| `any(...)` | `TRUE` if at least one argument is `TRUE` |
| `identical(x,y)` | Safe and reliable way to test two objects for being *exactly* equal |
| `all.equal(x,y)` | Test if two objects are *nearly* equal |

# Lógicos: No es lo mismo I

- En general, los operadores lógicos no pueden producir un solo valor y pueden devolver un NA si un elemento es NA o NaN.
- Si quieres como respuesta VERDADERO o FALSO, no debes utilizar -=.º i=", a menos que esté absolutamente seguro de que nada inusual puede suceder. Debes usar la función identical() en su lugar.

```
> name <- "Nick"
> if(name=="Nick") TRUE else FALSE
[1] TRUE
> if(identical(name, "Nick")) TRUE else FALSE
[1] TRUE
```

# Lógicos: No es lo mismo 2 I

- Con all.equal los objetos son tratados como iguales y la única diferencia es probablemente el resultado de cálculos con decimales. Devuelve TRUE si la diferencia relativa media es menos de la tolerancia especificada.

```
> (x <- sqrt(2))
[1] 1.414214
> x^2==2
[1] FALSE
> all.equal(x^2, 2)
[1] TRUE
> all.equal(x^2, 1)
[1] "Mean relative difference: 0.5"
> isTRUE(all.equal(x^2, 1))
[1] FALSE
```

# Factor I

- Un factor es una variable categórica con un número definido de niveles ordenados o no ordenados. Utilice la función **factor** para crear una variable.

```
> factor(rep(1:2, 4), labels=c("trt.1", "trt.2"))
[1] trt.1 trt.2 trt.1 trt.2 trt.1 trt.2 trt.1 trt.2
Levels: trt.1 trt.2
> factor(rep(1:3, 4), labels=c("low", "med", "high"),
 ordered=TRUE)
[1] low med high low med high low med high low med high
Levels: low < med < high
```

# funciones del Factor I

- levels(x): Recupera o establece los niveles de x
- nlevels(x): Da el número de niveles de x
- relevel(x,ref): Niveles de x se reordenan de manera que el nivel especificado por ref sea primero
- reorder(): Reordena los niveles sobre la base de los valores de una segunda variable
- gl(): Genera factores especificando el patrón de sus niveles
- cut(x, breaks): Divide el rango de x en intervalos (factores) determinado por lo que se especifique en breaks

# Datos Perdidos I

- R utiliza NaN para valores no definidos
- R utiliza Null para objetos nulos o no definidos
- R utiliza NA para datos perdidos (not available)
- Una función puede manejar los datos que faltan. Por ejemplo la funcion **mean** descarta valores perdidos si se escribe na.rm = TRUE)

```
> x <- c(4, 7, 2, 0, 1, NA)
> mean(x)
[1] NA
> mean(x, na.rm=TRUE)
[1] 2.8
> which(x>4)
[1] 2
```

# Observación I

- Hay que tener cuidado cual función usar.

```
> x <- c(4, 7, 2, 0, 1, NA)
> x==NA
[1] NA NA NA NA NA NA
%#note que transformó todos los datos a NA
```

# Detectando Datos Perdidos I

- is.na(x):detecta datos NA
- is.nan(x):detecta datos NaN
- is.null(x): verifica si x es nulo.

```
> is.na(x)
[1] FALSE FALSE FALSE FALSE FALSE TRUE
> any(is.na(x))
[1] TRUE
> (y <- x/0)
[1] Inf Inf Inf NaN Inf NA
> is.nan(y)
[1] FALSE FALSE FALSE TRUE FALSE FALSE
> is.na(y)
[1] FALSE FALSE FALSE TRUE FALSE TRUE
```