

# Informática para Economistas

## Introducción a Rstudio

M.Sc. Ciro Ivan Machacuay Meza

Universidad Nacional del Centro del Perú

2025



# Contenido

- 1 Introducción
- 2 Modo de trabajo
- 3 Tipos de objetos
- 4 Operadores y funciones
- 5 Listas y data
- 6 Subsetting and Element Extraction (indexación)
- 7 Guardar resultados
- 8 Missing value
- 9 Vectorización de funciones\*
- 10 Referencias

## Section 1

### Introducción

# Introducción

- “La programación es una habilidad vital para todos los que trabajan intensamente con datos”(Grolemund, 2014).
- R es un lenguaje de programación de distribución libre creado en 1993 por **Ross Ihaka** y **Robert Gentleman**.
- R funciona como un ambiente en el que se aplican técnicas estadísticas en lugar de una acumulación gradual de herramientas muy específicas y poco flexibles (Santana y Farfán, 2014).

# Introducción

## ¿Por qué R?

- ① **Es gratis.**
- ② Curva de aprendizaje plana, pero con alta potencia.
- ③ Es flexible y de fácil acceso.
- ④ Se actualiza constantemente (esto puede traer problemas\*), contrario a los programas comerciales.
- ⑤ Incorporación anticipada de técnicas estadísticas.
- ⑥ Una colección amplia, coherente e integrada de herramientas para el análisis de datos (Venables y Smith, 2021).
- ⑦ **Reproducible**, adios cajas negras (Castro, 2021).

# Introducción

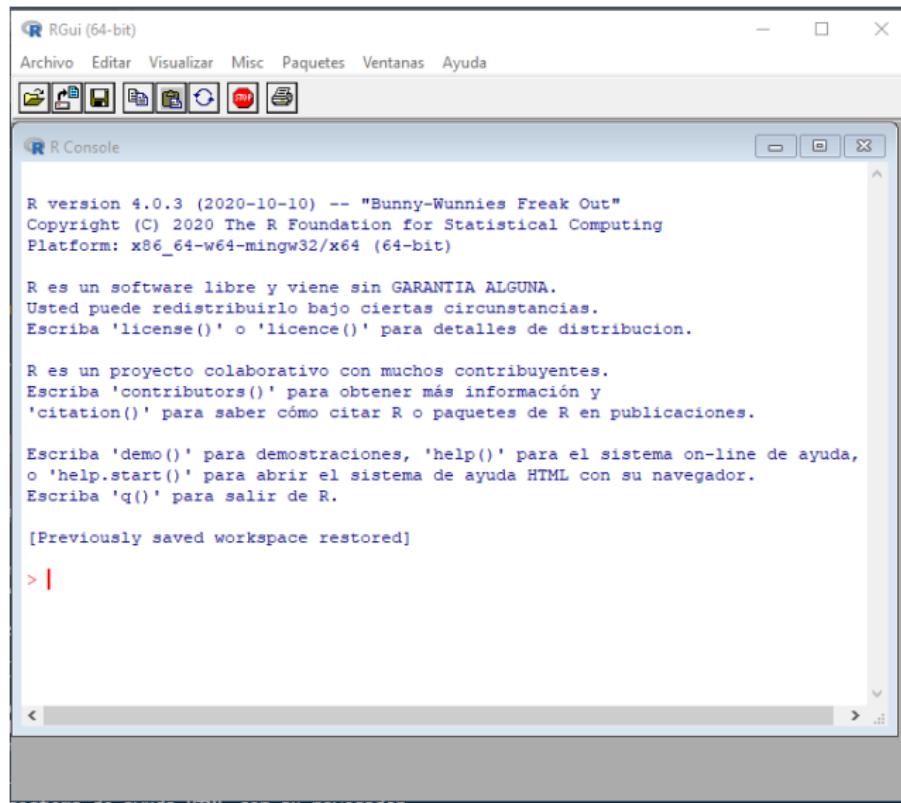
R se divide en dos partes:

- ① **El sistema base**, que puede ser descargado de la página oficial del proyecto <https://www.r-project.org/> -en su ultima versión o cualquiera de las anteriores-; y,
  - ② **Paquetes modulares (packages)**, cuya viabilidad es posible gracias a la facultades de R para desarrollar nuevas herramientas.
- 
- Nota: Las **actualizaciones** incluyen cambios y mejoras, pero pueden volver disfuncionales los programas ya realizados.

## Section 2

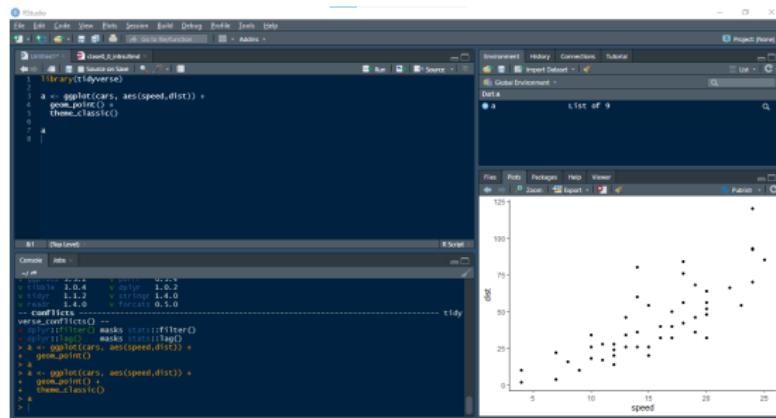
### Modo de trabajo

# Ambiente R



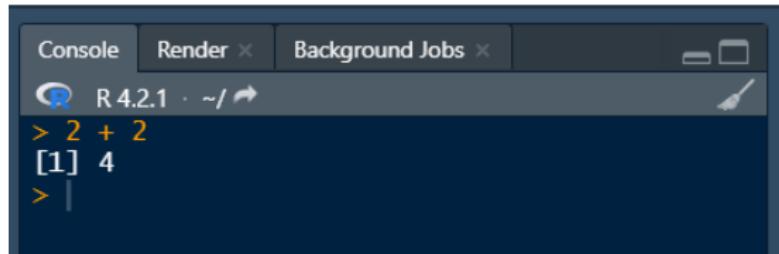
# Ambiente Rstudio

- Elementos claves de R (Davies, 2016):
  - Console: la consola es donde insertamos ordenes.
  - Script: copilación de sentencias o códigos de trabajo.
  - Workspaces: ambiente donde se encuentran todos los objetos creados.
  - Working Directory: ruta o carpeta donde R esta guardando o buscando información.



# Modo iterativo

- Instalado el programa, el modo iterativo representa la forma de uso más sencillo.
- En la consola, el “prompt”, representado por el símbolo `>` (mayor que), indica el lugar donde se escriben las órdenes (o comandos). Cuando introduces una **expresión** en la línea de comandos y das ENTER.



The screenshot shows a dark-themed R console window. At the top, there are tabs for "Console", "Render", and "Background Jobs". The main area displays the R prompt (`>`) followed by the expression `2 + 2`, which is evaluated and displayed as `[1] 4`. A cursor is visible at the bottom of the input field.

# Modo iterativo

- R puede ser usado como una calculadora (modo de trabajo iterativo):

2+2

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

2.3\*2

```
## [1] 4.6
```

2-3; 2^3; 8/2

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

```
## [1] 8
```

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

# Operador de asignación

- R es un lenguaje de programación, normalmente la razón de su uso es automatizar los procesos y evitar la repetición de tareas. Por tanto, podemos guardar los resultados en objetos.
- La manera de asignar un valor a un **objeto** (variable) en R, es usar el operador de asignación (`<-`, equivalente al signo de igual).

```
result<-5*6    # assign("result", 5*6)
#imprimir resultados
result
```

```
## [1] 30
```

```
#usarlo en otras operaciones
result+4
```

```
## [1] 34
```

# Operador de asignación

- Esta forma de asignar objeto en lugar de “=” (Respuesta de chat gpt5.0):
  - Evita la **confusión entre la asignación “=” e igualdad “==”**.
  - **Facilidad de lectura:** El uso de “<-” puede hacer que el código sea más legible.
  - Posibles **conflictos con funciones:** El uso de = como operador de asignación podría crear conflictos con el uso de = en argumentos de funciones.

# Comentarios

- El signo `#` se utiliza para **introducir comentarios**.
- R es **case sensitive**.

# Operador de asignación

- Por tanto, las órdenes introducidas en R pueden ser **expresiones** (se imprime y pierde su valor) o **asignaciones** (no se imprime y guarda su valor).
- Podemos usar operaciones o funciones alimentadas por argumentos (obligatorios u opcionales).

```
x <- c(7,8,15)
a <- sum(x)
a

## [1] 30
```

# Combinar operaciones

- Podemos utilizar las propiedades de vectorización para recuperar valores agregados aplicando operaciones sobre cada uno de los valores del vector.
- Ejemplo para el calculo de la varianza muestral:

```
sum(x^2)
```

```
## [1] 338
```

```
sum((x-mean(x))^2)/(length(x)-1)
```

```
## [1] 19
```

```
var(x)
```

```
## [1] 19
```

# Uso universal

- Pensemos siempre en utilizar sentencias que no necesiten la modificación humana ante cambios en los datos:
- En lugar de:

```
edad <- c(21, 16, 28, 8)  
sum(edad)/4
```

```
## [1] 18.25
```

- Usemos:

```
edad2 <- c(21, 16, 28, 8, 22)  
sum(edad2)/length(edad2)
```

```
## [1] 19
```

# Continuidad de órdenes

- Las órdenes se separan en líneas o mediante punto y coma (;). En caso de no estar completa, se muestra el signo + en la consola, hasta que sea completada.

```
max(7,8,15)
```

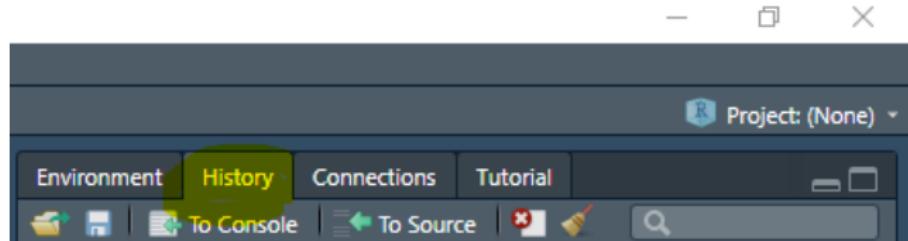
```
## [1] 15
```

```
max(7,8,  
     15)
```

```
## [1] 15
```

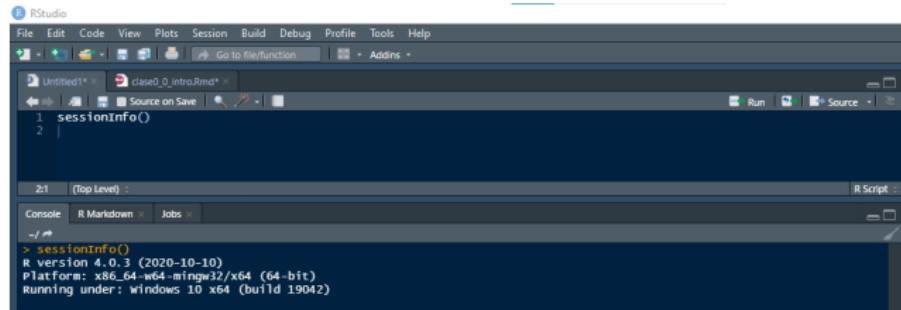
# Historial de comandos

- La flecha arriba, Ctrl + Up o el comando `history()` (últimos 25 comandos), permiten recuperar las órdenes ejecutadas.



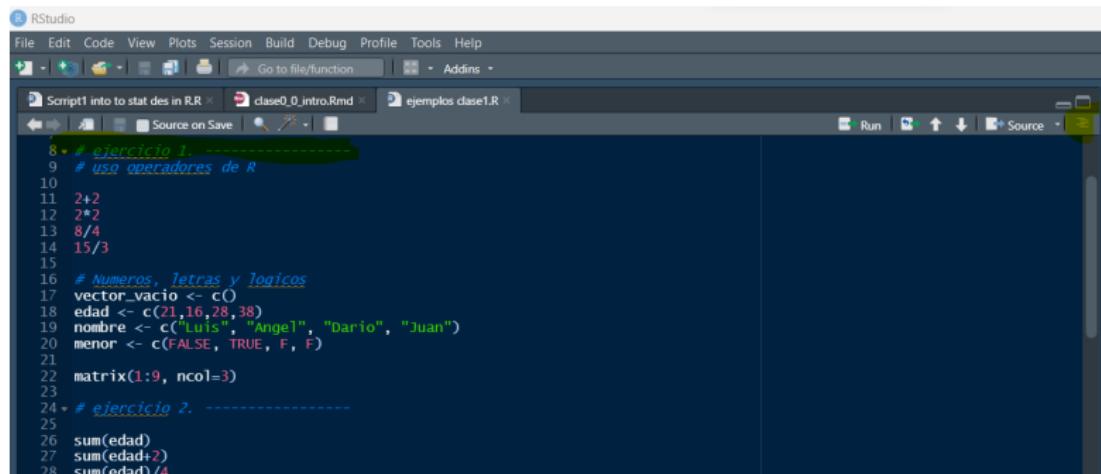
# Modo programador

- Para evitar tener que guardar cada sentencia, podemos acceder al modo programador trabajando sobre un script (`sessionInfo()`).
- SIEMPRE trabajar modo programador.



# Modo programador

- Divide tu **script** en secciones (agregando - seguidos al final del comentario —) permitirá ir organizando mejor sus trabajos.



The screenshot shows the RStudio interface with a dark theme. The top menu bar includes File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, and Help. Below the menu is a toolbar with various icons. The main workspace shows a script file with the following content:

```
8 * # ejercicio 1.
9 # uso operadores de R
10
11 2+2
12 2*2
13 8/4
14 15/3
15
16 # Numeros, letras y logicos
17 vector_vacio <- c()
18 edad <- c(21,16,28,38)
19 nombre <- c("luis", "Angel", "Dario", "Juan")
20 menor <- c(FALSE, TRUE, F, F)
21
22 matrix(1:9, ncol=3)
23
24 * # ejercicio 2. -----
25
26 sum(edad)
27 sum(edad+2)
28 sum(edad)/4
```

The code is color-coded: comments start with '#', arithmetic operators like '+', '\*', '/', and '/' are in blue, and variable names like 'vector\_vacio' and 'edad' are in green. The RStudio interface also shows tabs for 'Script1 into to stat des in R.R' and 'clase0\_0\_intro.Rmd'.

# Modo programador

- Escribe **scripts** para que el resto te entienda:

Ejemplo 1:

```
x <- 2 + 3
```

Ejemplo 2: el nombre de los objetos deben ser explicativos y las sentencias comentadas.

```
# Suma las edades de los individuos más jóvenes
edad_sum <- 2 + 3
```

## Section 3

# Tipos de objetos

# Objetos

Los objetos se pueden clasificar en:

- ① **Atómicos**, donde todos los elementos [texto, números (enteros, doble), lógicos] son del mismo tipo: (vectores y matrices).

```
nombre<-"Juanga"  
hombre <-TRUE  
edad <-23
```

- ② **Recursivos**, pueden combinar una colección de objetos diferentes (dataframe, listas).

```
data.frame(nombre,hombre,edad)
```

```
##   nombre hombre edad  
## 1 Juanga     TRUE    23
```

# Tipos de vectores: números

- Los vectores numérico. Es el tipo de datos con mayor importancia o principal uso, agrupados en numeric, integer y complex. El mismo, permite almacenar vectores y matrices.
- Estos son objetos comunes, permiten representar magnitudes. la función **concatenar** [c()] elementos del mismo tipo.

```
v1 <- 3
v2 <- 2+4
v3 <- c(1L,2L,3L)
v4 <- c(1:3)
ing <- c(17.3,21.1252,23.356)

assign("edad1", c(17,21,23))
```

# Objetos

- Cuando creamos un vector, R crea un vector con los valores (1, 2, 3), y lo vincula a un objeto llamado v3, por tanto, *puede pensar en un nombre como una referencia a un valor* (Hadley, 2020).

```
v3 <- c(1,2,3)
```

```
v4 <- v3
```

- El nombre asignado no puene comenzar con digitos o simbolos, tampoco se pueden usar palabas reservadas: TRUE, NULL, if... Otros nombres como o C, mean, se pueden usar, pero no son recomendable.

# Tipos de vectores: texto

- Caracteres (letras). Estas expresiones se denotan entre comillas simples o dobles.

```
x <- c("a", "b", "c")
class(x)
```

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

```
nombres<-c("Maria", "Jesus")
nombres
```

```
## [1] "Maria" "Jesus"
```

# Tipos de vectores: lógicos

- Los objetos lógicos representan valores binarios (falso (0) y verdadero (1)).
- En R, el valor falso se representan por FALSE (=0) o únicamente por la letra F mayúscula, el valor verdadero es representado por TRUE (=1) o por la letra T.

```
x<-c(T,T,F,F,T)  
x
```

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

```
# representación en números  
as.numeric(x)
```

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

```
x*1
```

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

# Asignando nombres

- Inserte un vector llamado clientes que indique la cantidad de clientes atendidos por una peluquero durante la semana:

```
cliente <- c(8,15,13,22,24)
cliente
```

```
## [1] 8 15 13 22 24
```

```
names(cliente) <- c("Lun", "Mar", "Mie", "Jue", "Vie")
cliente
```

```
## Lun Mar Mie Jue Vie
## 8 15 13 22 24
```

```
names(cliente)
```

```
## [1] "Lun" "Mar" "Mie" "Jue" "Vie"
```

# Factores nominales

```
sexo <- c("m", "f", "m", "f", "m", "m")
sexofactor <- factor(sexo, levels = c("m", "f"))

table(sexo)

## sexo
## f m
## 2 4

table(sexofactor)

## sexofactor
## m f
## 4 2

sexofactor <- factor(sexo, levels=c("m", "f"), labels=c("Hombre", "Mujer"))
table(sexofactor)

## sexofactor
## Hombre Mujer
##      4      2
```

# Vectores de secuencias y repeticiones

```
1:10 # del 1 al 10
```

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

```
10:1 # del 10 al 1
```

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

```
seq(1,10) #vector que vaya desde 1 a 10
```

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

```
seq(1,10,by=2) # fijar intervalos de la secuencia
```

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

```
seq(1,10,length=3) # fijar longitud de la secuencia
```

```
## [1] 1.0 5.5 10.0
```

# Vectores de secuencias y repeticiones

```
rep(c(1:3), time=2)
```

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

```
rep(c(1:3), each=2)
```

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

# Matrices

- Desde el punto de vista del lenguaje, una matriz es un vector con un atributo adicional.
- Los datos en R se van rellenando columna a columna de izquierda da derecha. Aunque también se puede especificar el orden deseado.

```
matrix(1:12, nrow=4, ncol=3)
```

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

```
(m1<-matrix(1:12, nrow=4, ncol=3, byrow=TRUE))
```

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

# Matrices

- A las matrices creadas podemos asignar nombres y acceder a sus elementos a partir de estos, las funciones son colnames y rownames.

```
colnames(m1)<-c("edad", "peso", "altura")
rownames(m1)<-c("Perla", "Juan", "Eva", "Alex")
m1
```

```
##          edad peso altura
## Perla      1    2     3
## Juan       4    5     6
## Eva        7    8     9
## Alex       10   11    12
```

# Modo programador

- Un modo de trabajo alternativo consiste en crear un **Script** que permita guardar instrucciones, y nos hace libre de realizar instrucción por instrucción. Para esto, escribimos los códigos en un **Script**.
  - Ctrl + Shift + N / file + new file + Script.
  - Ctrl + Shift + R (insertar secciones en un Script).
  - Ctrl + Shift + C (insertar comentarios (nos me gusta)).
  - Ctrl + Shift + N (new Script).
  - Ctrl + Shift + H (cambiar directorio de trabajo).
  - Ctrl + Shift + K (Compile Notebook).
  - Ctrl + Shift + Alt + M (renombrar en conjunto. OJO).
  - q() cerrar el programa.

# Paquetes

- Muchas herramientas de R no están disponible en el paquete base, no obstante, podemos contar con ella mediante librerías. Una forma de adherirlas:

```
# install.packages(stargazer)
library(stargazer)

## Lista de todos los paquetes instalados:
list_pac <- rownames(installed.packages())
```

- Otra alternativa para utilizar las funciones de un paquete consiste en utilizar el operador :: para adjuntar la función a R sin necesidad de cargar el paquete:

```
# stargazer::stargazer()
```

## Section 4

# Operadores y funciones

# Operaciones

- Sobre los objetos de R, utilizando diversos operadores, podemos obtener diferentes resultados. Los operadores más frecuentemente usados son:

Aritméticos	Relacionales	Lógicos	Especiales	Propios
+ Suma	== igualdad	& Y lógico	%% residuo	
- Resta	!= Diferente	! No lógico	%in% contenido	%!in%
* Multiplicación	< Menor que	O lógico	grep cuasi igualdad	
/ División	> Mayor que	all	<--> assignment	
$\wedge$ Potencia (**)	$\leq$ Menor o igual	any	[] \$ indexación	
sqrt()	$\geq$ Mayor o igual	&&,	> pipe	% > %
log(), exp()		xor		

Fuente: elaborado a partir de “The R Book”.

# Operaciones aritméticas con vectores

```
v<-c(6,4,8)
```

```
v+3
```

```
## [1] 9 7 11
```

```
v/2
```

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

```
v+c(1,2,3)
```

```
## [1] 7 6 11
```

```
v*c(1,2,3)
```

```
## [1] 6 8 24
```

```
v^(2:4)
```

```
## [1] 36 64 4096
```

# Funciones

- Las siguientes funciones proporcionan medidas de resumen de los estadísticos de un vector (Narayananchar, et al, 2016).

```
x <- c(2,5,8,6,7,9,10,15,-1,8)
```

```
sum(x); mean(x); max(x)
```

```
## [1] 69
```

```
## [1] 6.9
```

```
## [1] 15
```

# Funciones

- Los operadores de R son una función:

```
5+5
```

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

```
`+` (5,5)
```

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

# Funciones propias

- Podemos crear funciones a partir de nuestras necesidades:

```
coefvar <- function(x) sd(x)/mean(x)
coefvar(x)
```

```
## [1] 0.6352244
```

```
MyVar <- function(x){sum((x-mean(x))^2)/(length(x)-1)}
MyVar(x)
```

```
## [1] 19.21111
```

```
var(x)
```

```
## [1] 19.21111
```

# Funciones estadísticas

- También, podemos acceder a una importante batería de funciones estadísticas:

Posición	Dispersión	Forma	Asociación	Resumen
<code>mean(x)</code>	<code>var(x)</code>	<code>skewness(x)</code>	<code>cov(x)</code>	<code>summary</code>
<code>quantile(x,perc)</code>	<code>sd(x)</code>	<code>kurtosis(x)</code>	<code>cor(x)</code>	
<code>mean(x, trim=0.1)</code>	<code>range(x)</code>			
<code>median(x)</code>	<code>IQR(x)</code>			

- Fuente: elaborado a partir de (Harrison, 2020)

# funciones para strings

```
substr('tragabala', start = 1, stop = 4)
```

```
## [1] "trag"
```

```
nchar('tragabala') #cuenta caracteres, !=length
```

```
## [1] 9
```

```
# Última tres letra
```

```
substr('tragabala', nchar('tragabala')-3+1, nchar('tragabala'))
```

```
## [1] "ala"
```

```
# Sustituir elementos
```

```
chartr("a","A", "tragabala")
```

```
## [1] "trAgAbAlA"
```

# Operaciones aritméticos en funciones

- Los operadores pueden aparecer en formas de funciones:

```
log(2)
```

```
## [1] 0.6931472
```

```
sqrt(9)
```

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

```
abs(-7)
```

```
## [1] 7
```

```
round(12.3252,2)
```

```
## [1] 12.33
```

```
8 %% 2
```

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

# Operadores relacionales

- Los operadores relacionales permiten comparar elementos de un objeto, comparar objetos, indexar mediante operadores para generar vectores lógicos y establecer condicionales.

```
edad<-c(8,4,2,6,10,9)
```

```
edad>6
```

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

# Operadores relacionales

- Podemos usar los vectores lógicos para contabilizar la cantidad de observaciones que cumplen con una condición:

```
sum(edad<=6)
```

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

- Recuperar los valores de un vector que cumplen determinada condición:

```
(edad<=6)*edad #coloca cero a los valores que no cumplen la condición
```

```
## [1] 0 4 2 6 0 0
```

```
sum((edad<=6)*edad)
```

```
## [1] 12
```

# Lógicos

Estos operadores operan bajo los criterios del **álgebra de sucesos**.

- **Unión**  $\cup$ . La unión de sucesos  $A \cup B$  que ocurre si solo si, al menos uno de los dos sucesos ocurre (se lee como suma de A y B; A o B; A unido a B.).
- **Intersección**  $\cap$ . Producto de sucesos. Es el suceso que ocurre, solo si ocurre tanto A como B.

```
x<-c(T,T,F,F); y<-c(T,F,T,F)
```

```
y0x <- y | x
```

```
yYx <- y & x
```

```
cbind(x,y,y0x,yYx)
```

```
##          x      y    y0x    yYx
## [1,]  TRUE  TRUE  TRUE  TRUE
## [2,]  TRUE FALSE  TRUE FALSE
## [3,] FALSE  TRUE  TRUE FALSE
## [4,] FALSE FALSE FALSE FALSE
```

# Lógicos

- Usando una base de datos ilustrativa, podemos observar el uso de estos operadores:

```

edaM <- edad>5
edaMh <- edad>5 & sexofactor=="Hombre"
edaMhm <- edad>5 | sexofactor=="Hombre"
xorEdadMh <- xor(edad>5, sexofactor=="Hombre")

(Mydatos <- data.frame(sexofactor,edad, edaM, edaMh, !edaMh, edaMhm, xorEdadMh))

```

	sexofactor	edad	edaM	edaMh	X.edaMh	edaMhm	xorEdadMh
## 1	Hombre	8	TRUE	TRUE	FALSE	TRUE	FALSE
## 2	Mujer	4	FALSE	FALSE	TRUE	FALSE	FALSE
## 3	Hombre	2	FALSE	FALSE	TRUE	TRUE	TRUE
## 4	Mujer	6	TRUE	FALSE	TRUE	TRUE	TRUE
## 5	Hombre	10	TRUE	TRUE	FALSE	TRUE	FALSE
## 6	Hombre	9	TRUE	TRUE	FALSE	TRUE	FALSE

# Lógicos

- Anexo los resultados de distintas combinaciones cuando aparecen valores NA:

```
x<-c(T,T,F,F,NA,NA,NA); y<-c(T,F,T,F,T,F,NA)

y0x <- y | x
yYx <- y & x
xorxy <- xor(x,y) #tiene una de los dos, pero no las dos

cbind(x,y,y0x,yYx,xorxy, !y0x,!yYx)
```

```
##          x      y     y0x    yYx xorxy
## [1,]  TRUE  TRUE  TRUE  TRUE FALSE FALSE
## [2,]  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE
## [3,] FALSE  TRUE  TRUE FALSE  TRUE FALSE  TRUE
## [4,] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
## [5,]    NA  TRUE  TRUE    NA     NA FALSE    NA
## [6,]    NA FALSE    NA FALSE    NA     NA  TRUE
## [7,]    NA    NA    NA    NA     NA     NA    NA
```

# Funciones lógicas

- Los operadores también nos permiten verificar si alguna condición se cumple en todas nuestras observaciones:

```
#al menos 1 es hombre?  
any(sexofactor=="Hombre")
```

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

```
#son todos hombres?  
all(sexofactor=="Hombre")
```

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

# Práctica

- Suponga que a su institución financiera se acercan cinco individuos a solicitar créditos, y del formulario de solicitud ha completado la siguiente información:

```
# id=1; nombre=Juan; edad=21; ingreso=15000; calific=a.  
# id=2; nombre=Jose; edad=26; ingreso=22500; calific=b.  
# id=3; nombre=Ara; edad=40; ingreso=38520; calific=c.  
# id=4; nombre=Noel; edad=42; ingreso=32500; calific=b.  
# id=5; nombre=luisa; edad=17; ingreso=32500; calific=a.
```

- La política de préstamo de la empresa, indica que se asignan préstamos aquellos individuos mayores de edad, con ingresos superiores a los 30 mil pesos y una calificación crediticia de a o b.
  - Inserte los vectores y cree la base correspondiente en un data.frame.
  - Indique cuales individuos calificarían para el crédito, creando un vector lógico que asuma el valor de 1 para aquellos individuos que califican para crédito.

# Pertenece a (in)

- En caso que deseamos verificar la pertenencia a un conjunto de valores, en lugar de una solo:

```
edad %in% c(8,1,10)
```

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

```
identical(edad %in% c(8,1), edad==8 | edad==1)
```

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

```
sum(edad %in% c(8,1,10))
```

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

# No pertenece a (in)

- R permite crear operadores lógicos en función de nuestras necesidades: suponga desea negar contenido en, identificando ahora las edades que no corresponden a determinados años.
- Luego, este operador puede crearse o manejarse igual que cualquier otro operador.

```
`%!in%` = Negate(`%in%`)
```

```
edad
```

```
## [1] 8 4 2 6 10 9
```

```
edad %!in% c(8,1,10)
```

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

# Igualdad parcial (grep)

- Hasta ahora, hemos estado interesados en la igualdad total de los valores de los objetos, sin embargo, podemos estar interesados en identificar aspectos parciales de los valores de una celda.
- Iniciales del teléfono de una zona geográfica.
- Palabras claves incluidas en un texto.
- En el ejemplo siguiente se identifican los textos que contienen A o a ( $A|a+$  ver expresiones regulares para más detalles):

```
(nombres<-c("Maria", "Jesus", "Juan", "Antonio", "Ana"))
```

```
## [1] "Maria"    "Jesus"     "Juan"      "Antonio"   "Ana"
```

```
grep("A|a+", nombres)
```

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

# Sorting, Ranking and Ordering

```
ranks<-rank(edad)
sorted<-sort(edad)
ordered<-order(edad)

data.frame(edad, ranks, sorted, ordered)
```

```
##   edad ranks sorted ordered
## 1     8      4      2      3
## 2     4      2      4      2
## 3     2      1      6      4
## 4     6      3      8      1
## 5    10      6      9      6
## 6     9      5     10      5
```

## Section 5

### Listas y data

# Data frame

- Un data frame es un marco de datos (Base de datos p). Los
- **nombres de las variables** deben ser claros y seguir leyes.

Vectores:

```
nota <- c(91,76,68,78)
nombre <- c("Luis", "Angel", "Dario", "Juan")
menor <- c(FALSE, TRUE, F, F)
```

Data frame:

```
base_datos<-data.frame(nota,nombre,menor)
base_datos
```

```
##   nota nombre menor
## 1    91    Luis FALSE
## 2    76   Angel  TRUE
## 3    68   Dario FALSE
## 4    78   Juan FALSE
```

# Cargar base de datos desde Excel

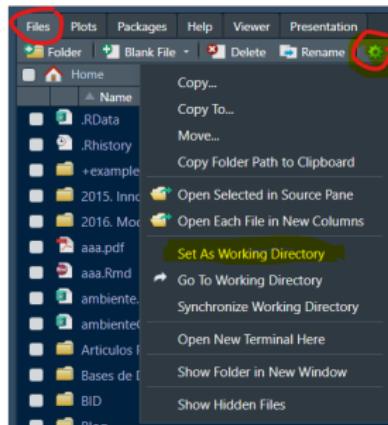
- La mayor parte del tiempo nuestros datos deben ser **importados** de alguna plataforma externa. R contiene una amplia cantidad de herramientas para esta tarea.
  - `read.csv("data.csv")`

# Cargar base de datos desde Excel

- Es importante aquí entender el **directorio de trabajo**:

```
getwd() #setwd()
```

```
## [1] "C:/Users/Nerys Ramirez/Dropbox/Tidyverse in R/Clase 0 Introduccion al ambiente"
```



# Cargar base de datos desde Excel

- la función **read\_excel** del paquete **readxl**, permite **importar datos desde Excel**.

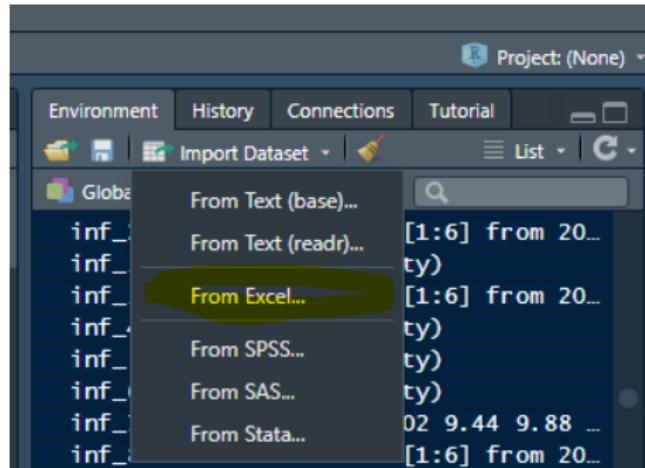
```
library(tidyverse)
wage1_excel <- readxl::read_excel("wage1.xlsx")
head(wage1_excel, 2)
```

```
## # A tibble: 2 x 24
##   wage   educ exper tenure nonwhite female married numdep smsa northcen south
##   <dbl> <dbl> <dbl>   <dbl>    <dbl>    <dbl>    <dbl> <dbl>    <dbl>    <dbl>
## 1  3.1     11     2      0      0       1       0       2       1       0       0
## 2  3.24    12     22      2      0       1       1       3       1       0       0
## # ... with 13 more variables: west <dbl>, construc <dbl>, ndurman <dbl>,
## #   trcommpu <dbl>, trade <dbl>, services <dbl>, profserv <dbl>, profocc <dbl>,
## #   clerocc <dbl>, servocc <dbl>, lwage <dbl>, expersq <dbl>, tenursq <dbl>
```

- Nota. siempre activo tidyverse por el formato tidy.

# Cargar base de datos desde Excel

- Mediante ventanas, podemos importar datos desde Excel:



# Cargar base de datos: pc

- La función `getwd()` nos devuelve el directorio de trabajo (carpeta donde R guardará o buscará los archivos que indiquemos) y el comando `setwd()` nos permite modificarlo.
- Es preferible especificar el directorio de trabajo para poder acortar las direcciones de objetos.

```
# setwd("C:/.../Datos")  
  
datawage <- read.delim("wage1.txt")  
head(datawage[,1:7], n=5)
```

```
##   wage educ exper tenure nonwhite female married  
## 1 3.10    11     2      0       0     1     0  
## 2 3.24    12    22      2       0     1     1  
## 3 3.00    11     2      0       0     0     0  
## 4 6.00     8    44     28       0     0     1  
## 5 5.30    12     7      2       0     0     1
```

# Utilidades con data.frame

- Las funciones `cbind()` y `rbind()` permiten concatenar atributos respecto a `dim` (combinar filas o columnas).
- Extraer el nombre de las columnas:

```
names(base_datos)
```

```
## [1] "nota"    "nombre"   "menor"
```

```
dput(names(base_datos))
```

```
## c("nota", "nombre", "menor")
```

# Cargar base de datos: paquetes y sistema

```
#data()  
  
#install.packages("wooldridge")  
library(wooldridge)  
head(wage1[,1:7], n=5)  
  
##   wage educ exper tenure nonwhite female married  
## 1 3.10    11     2      0       0      1      0  
## 2 3.24    12    22      2       0      1      1  
## 3 3.00    11     2      0       0      0      0  
## 4 6.00     8    44     28       0      0      1  
## 5 5.30    12     7      2       0      0      1
```

# Listas

- La lista es una colección de objetos de distintas magnitudes (el data frame solo acepta vectores de igual tamaño).

```
lista_a <- list(nombre = "USA",
                 datos = base_datos)
```

```
lista_a
```

```
## $nombre
## [1] "USA"
##
## $datos
##   nota nombre menor
## 1    91 Luis FALSE
## 2    76 Angel TRUE
## 3    68 Dario FALSE
## 4    78 Juan FALSE
```

# Listas

- Anexo un ejemplo de lista de base de datos:

```
list(list(nombre = "USA",
          datos = base_datos),  
  
      list(nombre = "Dom",
            datos = c()))
```

```
## [[1]]  
## [[1]]$nombre  
## [1] "USA"  
##  
## [[1]]$datos  
##   nota nombre menor  
## 1   91 Luis FALSE  
## 2   76 Angel TRUE  
## 3   68 Dario FALSE  
## 4   78 Juan FALSE  
##  
##  
## [[2]]  
## [[2]]$nombre  
## [1] "Dom"
```

# Listas

```
mylist <- list(edad,nombres,c=c(1,3))
mylist
```

```
## [[1]]
## [1] 8 4 2 6 10 9
##
## [[2]]
## [1] "Maria"    "Jesus"    "Juan"     "Antonio"  "Ana"
##
## $c
## [1] 1 3
```

# Listas

```
matriz1 <- matrix(1:6, nrow = 2)
matriz2 <- matrix(7:12, nrow = 2)
matriz3 <- matrix(13:18, nrow = 2)

# Crear una lista de matrices
lista_de_matrices <- list(matriz1, matriz2, matriz3)
lista_de_matrices

## [[1]]
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## [[2]]
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
##
## [[3]]
##      [,1] [,2] [,3]
## [1,]   13   15   17
## [2,]   14   16   18
```

## Section 6

# Subsetting and Element Extraction (indexación)

# Indexar vectores

- R permite acceder a un elemento de un vector añadiendo el nombre de un vector en un índice entre corchetes [].
- El vector índice, puede indicar parte de otro vector y ser de tres tipos:
  - Vector de *números enteros*: en caso de ser positivos son concatenados y en caso de ser negativos son omitidos. No hace falta sea de igual longitud al vector seleccionado.
  - Vector lógico: los valores correspondientes a TRUE son seleccionados.
  - Vector de nombres: solo se aplica cuando el vector tiene el atributo names.

# Indexar vectores

```
edad
```

```
## [1] 8 4 2 6 10 9
```

```
edad[5] # El quinto elemento
```

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

```
edad[c(1,3,6)] # Posiciones específicas
```

```
## [1] 8 2 9
```

```
edad[length(edad)]
```

```
## [1] 9
```

```
edad[1:5] # Del primer al quinto elemento
```

```
## [1] 8 4 2 6 10
```

# Omitir o sobreescribir

- Sobreescrivir:

```
edad
```

```
## [1] 8 4 2 6 10 9
```

```
edad[5] <- 8
```

```
edad
```

```
## [1] 8 4 2 6 8 9
```

- Omitir:

```
edad[-5] # Todos los elementos, menos el quinto
```

```
## [1] 8 4 2 6 9
```

# Indexación por números

- El acceso a los elementos individuales de un vector no solamente es para consulta o lectura, sino también para su modificación o escritura.

```
# Operaciones con elementos  
edad[4]+edad[2]
```

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

```
# Guardar los resultados en una posición  
edad[1] <- edad[2] + edad[5] - 7  
edad
```

```
## [1] 5 4 2 6 8 9
```

# Indexación lógica

- El uso de operadores lógicos también permite el uso de **indexación lógica**. El mismo, funciona como un filtro condicional para acceder a subconjuntos de datos que cumplen con determinadas condiciones.

```
edad[edad>2] # Un vector solo con las edades mayores a 2
```

```
## [1] 5 4 6 8 9
```

```
edad[3 <= edad & edad <= 7] # Un vector solo con las edades comprendidos entre 3 y
```

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

```
sum(3 <= edad & edad <= 7) #número de elementos contenidos en un rango determinado
```

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

# Indexación lógica

```
ingresos <- rnorm(10,mean=20,sd=5)
ingresos[ingresos>10]
```

```
## [1] 24.80439 24.88727 27.67358 29.48933 28.09376 29.17022 21.28451 24.70070
## [9] 12.84230 18.94330
```

```
ingresos[ingresos>10]<-0
```

# Indexación mediante which

- La función `which` suele ser útil dado que indica la posición del vector donde se encuentra una numero determinado [ver `which.min()` `which.max()`].

```
edadx <- c(15,22,15,19,17)
vec_log <- edadx <= 17
vec_log
```

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

```
which(vec_log)
```

```
## [1] 1 3 5
```

# Indexación mediante which

- En el caso de operadores lógicos podemos:
  - Posiciones de personas con más de 5 años.
  - Posiciones de personas con edad menor al promedio.
  - Posiciones de la persona con mayor edad.

```
which(edad>5)
```

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

```
which(edad<mean(edad))
```

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

```
which(edad==max(edad))
```

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

# Indexación de matrices

- En caso de una matriz bidimensional se debe especificar [*nfila, ncol*] tanto fila como columna. Luego, aplican las mismas reglas vistas hasta aquí.

```
m1 <- matrix(1:12, nrow=3)
```

```
m1
```

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

```
m1[3,2] #Obtener el elemento de la fila 2, columna 3
```

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

```
m1[2,] #Llamar todos los elementos de la fila 2
```

```
## [1] 2 5 8 11
```

```
m1[2,1:2]
```

```
## [1] 2 5
```

# Indexación de matrices

```
m1
```

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

```
m1[nrow(m1),ncol(m1)]
```

```
## [1] 12
```

```
"["(m1,1:2,2)
```

```
## [1] 4 5
```

# Indexación de data.frame

- Cuando se trabaja con data.frame, se puede acceder a sus subconjuntos, usando el simbolo de \$, a su izquierda se coloca el nombre de la lista y a la derecha (o directamente luego de utilizar el comando attach), o utilizando [].
- Dentro del corchete debemos indicar las indicaciones tanto para filas como para columnas.

[fila,columna]

- No colocar una de estas indicaciones R asume deseas todas las filas o columnas.

# Indexación de data.frame

- Seleccionando columnas:

```
Mydatos$edad
```

```
## [1] 8 4 2 6 10 9
```

```
Mydatos[,2]
```

```
## [1] 8 4 2 6 10 9
```

```
Mydatos[, 'edad']
```

```
## [1] 8 4 2 6 10 9
```

# Indexación de data.frame

- La opción de doble paréntesis ([[ ]]) permite acceder a una columna determinada.

```
Mydatos[[2]]
```

```
## [1] 8 4 2 6 10 9
```

```
Mydatos[2]
```

```
##     edad
## 1      8
## 2      4
## 3      2
## 4      6
## 5     10
## 6      9
```

# Indexación de data.frame

- Acceder a elementos de una fila, equivalente a realizar filtros entre usuarios de Excel.

```
Mydatos[edad>4,] #Establece la base solo con mayores de 4 años
```

```
##   sexofactor edad edaM edaMh X.edaMh edaMhm xorEdadMh
## 1     Hombre    8  TRUE  TRUE  FALSE  TRUE  FALSE
## 4     Mujer    6  TRUE FALSE  TRUE  TRUE  TRUE
## 5     Hombre   10  TRUE  TRUE  FALSE  TRUE  FALSE
## 6     Hombre    9  TRUE  TRUE  FALSE  TRUE  FALSE
```

```
Mydatos[sexofactor=="Mujer",] #Establece la base solo con mujeres
```

```
##   sexofactor edad edaM edaMh X.edaMh edaMhm xorEdadMh
## 2     Mujer    4 FALSE FALSE  TRUE FALSE  FALSE
## 4     Mujer    6  TRUE FALSE  TRUE  TRUE  TRUE
```

# Indexación de data.frame

- Combinar filas y columnas:

- ‘ En este caso, las columnas representan variables, mientras que las filas corresponden observaciones. Por tanto, condicional a filas es similar a realizar filtros o análisis condicional en Excel.

```
Mydatos[sexofactor=="Hombre",c(1:3)]
```

```
##   sexofactor edad edaM
## 1      Hombre    8  TRUE
## 3      Hombre    2 FALSE
## 5      Hombre   10  TRUE
## 6      Hombre    9  TRUE
```

- **Ejercicios.** Dado el siguiente data.frame, obtenga:

- Notas y el número de estudiantes que aprobaron el curso ( $> 70$ ).
- Nota media de hombres y mujeres (por sexo).
- Nota promedio de las mujeres que aprobaron.
- Nota promedio de las mujeres que aprobaron y eran de la provincia a.
- Nota promedio de las mujeres que aprobaron y eran de la provincia a o c.
- Use indexación por posición para indicar de que provincia era la persona con la nota más alta.
- Obtenga un data.frame omitiendo los reprobados.

```
##   nota mujer prov
## 1   75     1     a
## 2   80     1     b
## 3   60     0     a
## 4   85     1     c
## 5   70     0     a
## 6   65     0     b
```

# Indexación de lista

- Un ejemplo de la utilizadas de las listas se obtienen de la salidas de funciones como la usada para estimar modelos de regresión lineal:

```
model <- lm(speed~dist, data=cars)

is.list(model)

## [1] TRUE

names(model)

##  [1] "coefficients"   "residuals"        "effects"         "rank"
## [5] "fitted.values"  "assign"           "qr"              "df.residual"
## [9] "xlevels"         "call"             "terms"           "model"
```

# Indexación de lista

```
mylist[1] #primer vector
```

```
## [[1]]  
## [1] 8 4 2 6 10 9
```

```
mylist[[1]]
```

```
## [1] 8 4 2 6 10 9
```

```
mylist[["c"]]
```

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

```
mylist[[2]][1]
```

```
## [1] "Maria"
```

```
mylist$edad
```

```
## NULL
```

## Section 7

### Guardar resultados

# Exportar resultados en pantalla

- Cada vez que ejecutamos una orden en R se imprimen los resultados en pantallas, el comando sink permite guardar estas impresiones en un archivo palno txt.

```
sink("rfinal.txt")  
2+2
```

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

```
sink()
```

- Adicionalmente, es posible salvar el ambiente de trabajo.

```
save.image("ambienteClase1.RData")
```

# Exportar a pdf

- R permite guardar el script, resultado de consola, historico de comandos, gráficos, listas u objetos.

```
# guardar resultados impresos en consola
pdf("graph.pdf")

x<-rnorm(100)
hist(x)
dev.off()

## pdf
## 2
```

# Guardar objetos

- Adicionalmente, podemos guardar lista/objetos de nuestro ambiente de trabajo en formato RData.

```
mi_lista <- list(  
  nombre = "Juan",  
  edad = 30,  
  ciudad = "Ocoa"  
)  
  
saveRDS(mi_lista, file = "mi_lista.RData")
```

- Para cargar la lista desde el archivo RData:

```
mi_lista_cargada <- readRDS("mi_lista.RData")
```

## Section 8

# Missing value

# Missing value

- Los valores perdidos en R se representan mediante NA (Not Available), estos se utilizan para representar valores faltantes o ausentes en un vector o en un conjunto de datos. .

```
edad3 <- c(21, 16, NA, 28, 38, 17)
mean(edad3, na.rm = TRUE)
```

```
## [1] 24
```

# Missing value

- R contiene una bateria de funciones para trabajar con NA.

```
(x <- c(NA, 1:5))
```

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

```
is.na(x)
```

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

```
x[!is.na(x)]
```

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

```
(x[is.na(x)] <- mean(x, na.rm = T))
```

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

## Section 9

# Vectorización de funciones\*

# Vectorización de funciones\*

- Aquí solo se presentan algunos ejemplos como forma de motivar contenido futuro.

```
tapply(edad, sexofactor, mean)
```

```
## Hombre Mujer  
##      6     5
```

```
by(edad, sexofactor,mean)
```

```
## sexofactor: Hombre  
## [1] 6  
## -----  
## sexofactor: Mujer  
## [1] 5
```

# Vectorización de funciones

- La función split nos permite segmentar un vector según las clases incluidas en un factor, generando una lista.
- La función vectorizada sapply permite realizar una orden sobre cada uno de los elementos de la lista (genera un vector, lapply una lista).

```
split(edad, sexo)
```

```
## $f  
## [1] 4 6  
##  
## $m  
## [1] 5 2 8 9
```

```
sapply(split(edad, sexo), mean)
```

```
## f m  
## 5 6
```

# Vectorización de funciones

```
aggregate(edad,list(sexo),mean)
```

```
##      Group.1 x
## 1          f 5
## 2          m 6
```

# Vectorización de funciones

- Seleccionar todas las columnas que cumplen con determinada condición:

```
Mydatos[,sapply(Mydatos,is.logical)]
```

```
##      edaM edaMh X.edaMh edaMhm xorEdadMh
## 1    TRUE  TRUE    FALSE   TRUE    FALSE
## 2   FALSE FALSE    TRUE  FALSE    FALSE
## 3   FALSE FALSE    TRUE   TRUE    TRUE
## 4    TRUE FALSE    TRUE   TRUE    TRUE
## 5    TRUE  TRUE   FALSE   TRUE    FALSE
## 6    TRUE  TRUE   FALSE   TRUE    FALSE
```

# Indexación vectorizada

- De una lista de matrices, deseamos recuperar la primera columna de cada una de las matrices:

```
lista_de_matrices[[1]]
```

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

```
lapply(lista_de_matrices, function(x) x[, 1])
```

```
## [[1]]
## [1] 1 2
##
## [[2]]
## [1] 7 8
##
## [[3]]
## [1] 13 14
```

# Indexación vectorizada

- Podemos usar vectorización para crear un lista con distintos operadores:

```
sexo <- c("f","f","m","f","m")
lapply(c("f","m"), function(x) which(sexo==x))
```

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

```
split(1:length(sexo), sexo)
```

```
## $f
## [1] 1 2 4
##
## $m
## [1] 3 5
```

## Section 10

### Referencias

# Referencias

- ① Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988). The New S Language. Wadsworth and Brooks/Cole.
- ② Castro, E. (2020). Diseño experimental y análisis de datos. Universidad Andrés Bello.
- ③ Conesa, D. (2013). "Curso introducción R". Universidad de Valencia, Dept. de estadística e investigación operativa. Valencia, España.
- ④ Contreras J.M., Molina E. y Arteaga P. "Introducción a la programación estadística con R".
- ⑤ Crawley, M. The R Book. Imperial College London.
- ⑥ Grolemund, G. (2014). Hands-On Programming with R.
- ⑦ Harrison, G. (2020). "Getting Started with R for Measurement". Universidad de Hawái.
- ⑧ James, Gareth; Witten, Daniela; Hastie, Trevor and Tibshirani, Robert. An Introduction to Statistical Learning. Springer Texts in Statistics. USA.
- ⑨ Matloff, N. (2009). "The Art of R Programming"
- ⑩ Narayanachar, P.; Ramaiah, S and Manjunath, B.G. (). "A course in Statistics with R". Wiley. This edition first published 2016.
- ⑪ Pruim, R. (2011). Computational Statistics Using R and R Studio An Introduction for Scientists. SC 11 Education Program.

# Referencias

- ① Romero, F. (nd). Matrices, marcos de datos y lectura de datos en R. Máster en Lógica, Computación e Inteligencia Artificial. Universidad de Sevilla.
- ② Santana (2013). "El Arte de Programar en R".
- ③ Teator, P. (2011). R Cookbook. United States of America.
- ④ R Core Team (2018). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria.
- ⑤ Venable, W.; Smith, D. (2021). An Introduction to R. R Core Team.
- ⑥ Verzani, J. (2002). simpleR - Using R for Introductory Statistics.
- ⑦ Wickham, H. (2015). Advanced R. Taylor and Francis, LLC.