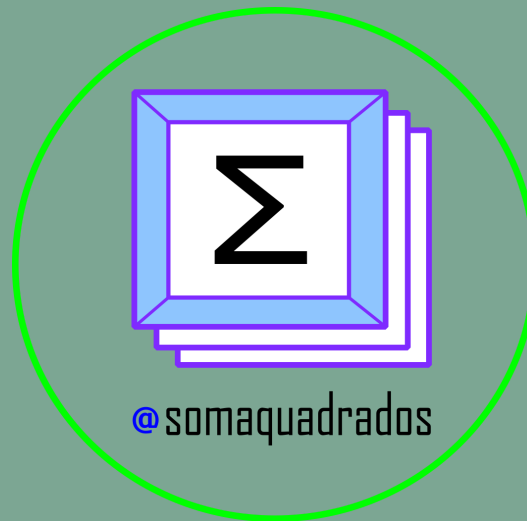
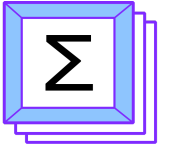


# Programación con R

## Clase 2



Marília Melo Favalesso



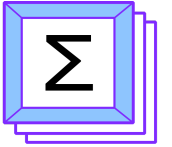
@somaquadrados

# Archivos

- clase\_2.R
- datos.xlsx
- datos.txt
- datos.csv
- tidy\_ej.xlsx

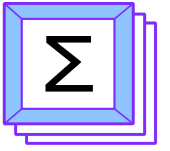
# Contenido de hoy

- Manejo de objetos en R
- ¿Cómo armar mi tabla de datos?
- Tidyverse
- Importar datos a R
- Operador pipe (%>%)
- tidyr
- dplyr



@somaquadrados

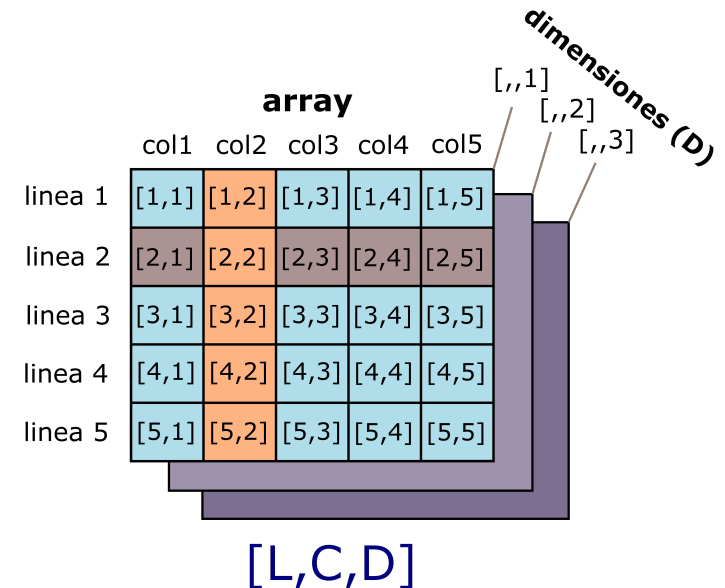
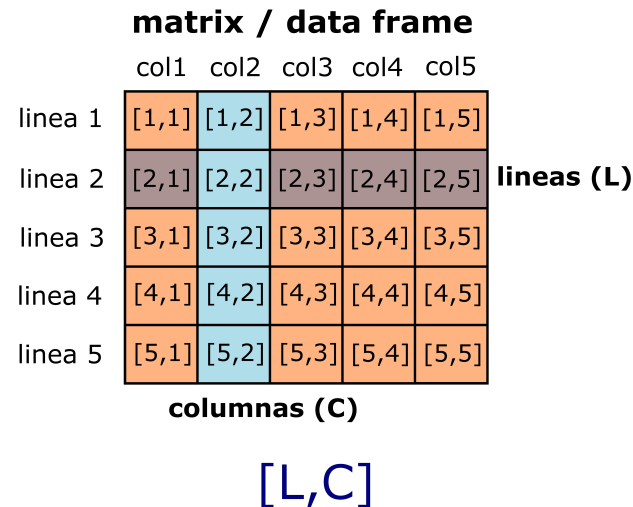
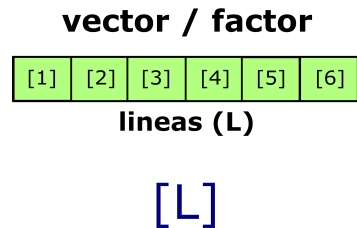
# Manejo de datos

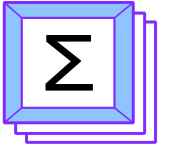


# Manejo de datos

## Indexación

- Los objetos son conjuntos *indexados* - Esto nos permite acceder a cada valor de manera individual.
- Comprender la indexación es fundamental para manipular datos en **R**.
- Usamos corchetes (`[]`) para acceder a la posición de los elementos de un objeto.





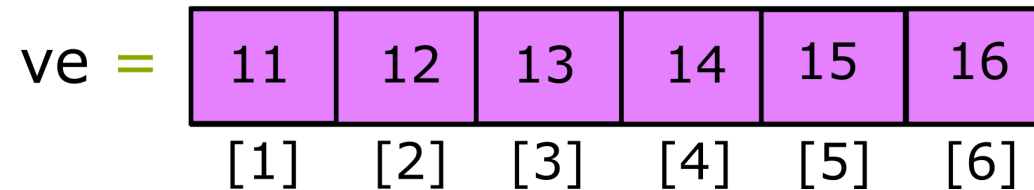
@somaquadrados

# Manejo de datos

## Indexación

### Gestión de datos unidimensionales [L]

- Seleccionar elementos



- No **R**:

```
ve = c(11, 12, 13, 14, 15, 16)
ve
```

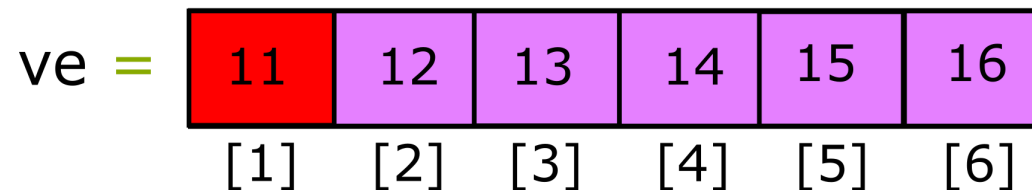
```
## [1] 11 12 13 14 15 16
```

# Manejo de datos

## Indexación

### Gestión de datos unidimensionales [L]

- Seleccionar elementos



- No **R**:

```
ve <- c(11, 12, 13, 14, 15, 16)
ve
```

```
## [1] 11 12 13 14 15 16
```

```
ve[1]
```

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

# Manejo de datos



## Indexación

### Gestión de datos unidimensionales [L]

- Seleccionar elementos

```
ve[2] # selecciona solo el segundo elemento
```

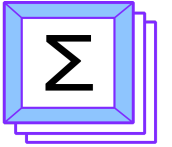
```
## [1] 12
```

```
ve[c(2, 4)] # selecciona los elementos en la segunda y cuarta posición
```

```
## [1] 12 14
```

```
ve[4:6] # selecciona los entre la cuarta y sexta posición
```

```
## [1] 14 15 16
```



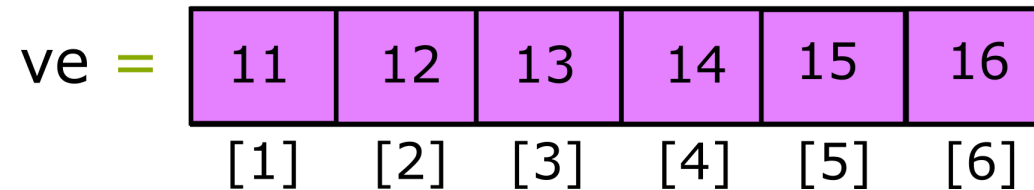
@somaquadrados

# Manejo de datos

## Indexación

### Gestión de datos unidimensionales [L]

- Quitar elementos

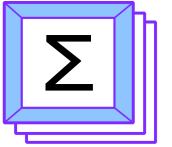


```
ve = c(11, 12, 13, 14, 15, 16)
```

```
ve
```

```
## [1] 11 12 13 14 15 16
```



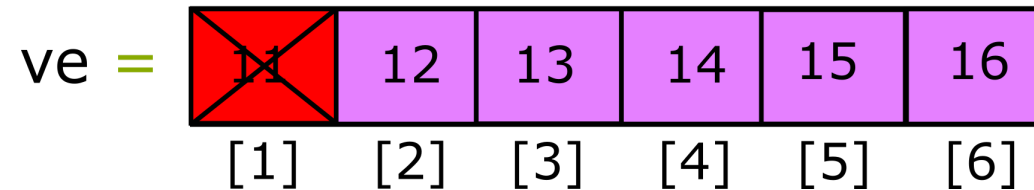


# Manejo de datos

## Indexación

### Gestión de datos unidimensionales [L]

- Quitar elementos



```
ve = c(11, 12, 13, 14, 15, 16)  
ve
```

```
## [1] 11 12 13 14 15 16
```

```
ve[-1]
```

```
## [1] 12 13 14 15 16
```

# Manejo de datos

## Indexación

### Gestión de datos unidimensionales [L]

- Quitar elementos

```
ve[-2] # quitar solo el segundo elemento
```

```
## [1] 11 13 14 15 16
```

```
ve[-c(2, 4)] # quitar los elementos en la segunda y cuarta posición
```

```
## [1] 11 13 15 16
```

```
ve[-c(4:6)] # quitar los entre la cuarta y sexta posición
```

```
## [1] 11 12 13
```

# Manejo de datos

## Indexación

### Gestión de datos bidimensionales [L,C]

- Seleccionar elementos

	col1	col2	col3	col4	col5	
linea 1	[1,1]	[1,2]	[1,3]	[1,4]	[1,5]	
linea 2	[2,1]	[2,2]	[2,3]	[2,4]	[2,5]	<b>lineas (L)</b>
linea 3	[3,1]	[3,2]	[3,3]	[3,4]	[3,5]	
linea 4	[4,1]	[4,2]	[4,3]	[4,4]	[4,5]	
linea 5	[5,1]	[5,2]	[5,3]	[5,4]	[5,5]	
	<b>columnas (C)</b>					

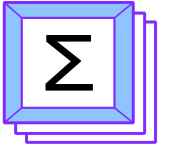
# Manejo de datos

## Indexación

### Gestión de datos bidimensionales [L,C]

- Seleccionar elementos

	col1	col2	col3	col4	col5	
linea 1	[1,1]	[1,2]	[1,3]	[1,4]	[1,5]	
linea 2	[2,1]	[2,2]	[2,3]	[2,4]	[2,5]	<b>lineas (L)</b>
linea 3	[3,1]	[3,2]	[3,3]	[3,4]	[3,5]	
linea 4	[4,1]	[4,2]	[4,3]	[4,4]	[4,5]	
linea 5	[5,1]	[5,2]	[5,3]	[5,4]	[5,5]	
	<b>columnas (C)</b>					



# Manejo de datos

## Indexación

### Gestión de datos bidimensionales [L,C]

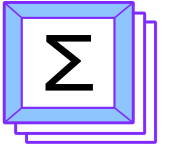
- Seleccionar elementos

```
ma <- matrix(data = (c(1:25)), nrow = 5, ncol = 5, byrow = TRUE) # crear una matriz
ma
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   1   2   3   4   5
## [2,]   6   7   8   9  10
## [3,]  11  12  13  14  15
## [4,]  16  17  18  19  20
## [5,]  21  22  23  24  25
```

```
ma[2,2] # seleccionar el valor de la segunda línea y la segunda columna.
```

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



@somaquadrados

# Manejo de datos

## Indexación

### Gestión de datos bidimensionales [L,C]

- Seleccionar elementos

```
ma[3:4, c(3, 5)] # seleccione las líneas 3 y 4 y las columnas 3 y 5
```

```
##      [,1] [,2]  
## [1,]   13   15  
## [2,]   18   20
```

```
ma[c(1,5), 3:5] # seleccione las líneas 1 y 5 y las columnas entre 3 - 5
```

```
##      [,1] [,2] [,3]  
## [1,]    3    4    5  
## [2,]   23   24   25
```

Es posible seleccionar más de una fila y columna al mismo tiempo.

# Manejo de datos

## Indexación

### Gestión de datos bidimensionales [L,C]

- Quitar elementos

	col1	col2	col3	col4	col5	
linea 1	[1,1]	[1,2]	[1,3]	[1,4]	[1,5]	
linea 2	[2,1]	[2,2]	[2,3]	[2,4]	[2,5]	<b>lineas (L)</b>
linea 3	[3,1]	[3,2]	[3,3]	[3,4]	[3,5]	
linea 4	[4,1]	[4,2]	[4,3]	[4,4]	[4,5]	
linea 5	[5,1]	[5,2]	[5,3]	[5,4]	[5,5]	
	<b>columnas (C)</b>					

# Manejo de datos

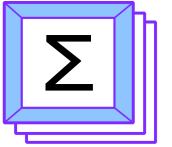
## Indexación

### Gestión de datos bidimensionales [L,C]

- Quitar elementos

	col1	col2	col3	col4	col5	
linea 1	[1,1]	[1,2]	[1,3]	[1,4]	[1,5]	
linea 2	[2,1]	<del>[2,2]</del>	[2,3]	[2,4]	[2,5]	<b>lineas (L)</b>
linea 3	[3,1]	[3,2]	[3,3]	[3,4]	[3,5]	
linea 4	[4,1]	[4,2]	[4,3]	[4,4]	[4,5]	
linea 5	[5,1]	[5,2]	[5,3]	[5,4]	[5,5]	
	<b>columnas (C)</b>					





@somaquadrados

# Manejo de datos

## Indexación

### Gestión de datos bidimensionales [L,C]

- Quitar elementos

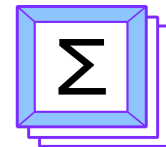
```
ma[-2, -2] # menos la fila dos y la columna dos
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    4    5
## [2,]   11   13   14   15
## [3,]   16   18   19   20
## [4,]   21   23   24   25
```

```
ma[-c(1, 4), -2:-3] # menos las filas 1 y 4 y las columnas 2 y 3
```

```
##      [,1] [,2] [,3]
## [1,]    6    9   10
## [2,]   11   14   15
## [3,]   21   24   25
```

# Manejo de datos



@somaquadrados

## Indexación

### Gestión de datos bidimensionales [L,C]

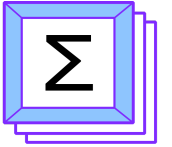
- También podemos usar el nombre de las filas y columnas para manejar los datos: `[nombre_linea, nombre_columna]`.

```
rownames(ma) = paste("nomlin", 1:5, sep = "_") # nombre en las lineas
colnames(ma) = paste("nomcol", 1:5, sep = "_") # nombre en las columnas
ma # la tabla
```

```
##           nomcol_1 nomcol_2 nomcol_3 nomcol_4 nomcol_5
## nomlin_1         1         2         3         4         5
## nomlin_2         6         7         8         9        10
## nomlin_3        11        12        13        14        15
## nomlin_4        16        17        18        19        20
## nomlin_5        21        22        23        24        25
```

```
ma["nomlin_2", "nomcol_3"] # selecciona linea 2 y columna 3
```

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



@somaquadrados

# Manejo de datos

## Indexación

### Gestión de *data frame* \$:

El operador `$` se utiliza para extraer elementos con nombre de un **data frame**.

```
# vectores
grupo <- rep(c("CT", "EXP"), 5)
valor <- sample(1:50, 10)
genero <- sample(c("H", "M"), 10, replace = T)
```

```
# data-frame
dt <- data.frame(grupo, valor, genero)
```

```
# resultado
dt
```

##	grupo	valor	genero
## 1	CT	37	M
## 2	EXP	40	H
## 3	CT	16	H
## 4	EXP	49	H
## 5	CT	24	M
## 6	EXP	3	H
## 7	CT	1	M
## 8	EXP	6	M
## 9	CT	21	H
## 10	EXP	38	H

# Manejo de datos



## Indexación

### Gestión de *data frame* con `$`:

- El operador `$` se utiliza para extraer elementos con nombre de un *data frame*.

```
dt$grupo
```

```
## [1] "CT" "EXP" "CT" "EXP" "CT" "EXP" "CT" "EXP" "CT" "EXP"
```

```
dt$valor
```

```
## [1] 37 40 16 49 24 3 1 6 21 38
```

```
dt$genero
```

```
## [1] "M" "H" "H" "H" "M" "H" "M" "M" "H" "H"
```

# Manejo de datos



## Indexación

### Gestión de *data frame* con `$`:

- Para cambiar la clase de elementos de una columna:

```
class(dt$grupo)
```

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

```
dt$grupo <- as.factor(dt$grupo)
dt$grupo
```

```
## [1] CT EXP CT EXP CT EXP CT EXP
## Levels: CT EXP
```

```
class(dt$grupo)
```

```
## [1] "factor"
```

# Manejo de datos



## Indexación

### Gestión de *data frame* con `$`:

- Para cambiar la clase de elementos de una columna:

```
class(dt$valor)
```

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

```
dt$valor <- as.numeric(dt$valor)  
dt$valor
```

```
## [1] 37 40 16 49 24 3 1 6 21 38
```

```
class(dt$valor)
```

```
## [1] "numeric"
```

# Manejo de datos



## Indexación

### Gestión de *data frame* con `$`:

- agregar una nueva columna:

```
# antes  
dt
```

```
##      grupo valor genero  
## 1      CT    37      M  
## 2     EXP    40      H  
## 3      CT    16      H  
## 4     EXP    49      H  
## 5      CT    24      M  
## 6     EXP     3      H  
## 7      CT     1      M  
## 8     EXP     6      M  
## 9      CT    21      H  
## 10    EXP    38      H
```

```
# después  
dt$ID <- 1:10; dt
```

```
##      grupo valor genero ID  
## 1      CT    37      M   1  
## 2     EXP    40      H   2  
## 3      CT    16      H   3  
## 4     EXP    49      H   4  
## 5      CT    24      M   5  
## 6     EXP     3      H   6  
## 7      CT     1      M   7  
## 8     EXP     6      M   8  
## 9      CT    21      H   9  
## 10    EXP    38      H  10
```

# Manejo de datos



## Indexación

### Gestión de *data frame* con `$`:

- Usamos el `$` para separar una variable de un **data.frame**.
- Esta variable ahora se puede manejar como un *objeto unidimensional*.

```
# selecciona el primero elemento  
dt$valor
```

```
## [1] 37 40 16 49 24 3 1 6 21 38
```

```
dt$valor[1]
```

```
## [1] 37
```

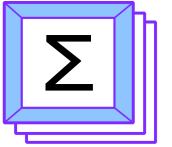
```
# elimina los valores en las posiciones 3 y 4  
dt$genero
```

```
## [1] "M" "H" "H" "H" "M" "H" "M" "M" "H" "H"
```

```
dt$genero[-c(3:4)]
```

```
## [1] "M" "H" "M" "H" "M" "M" "H" "H"
```



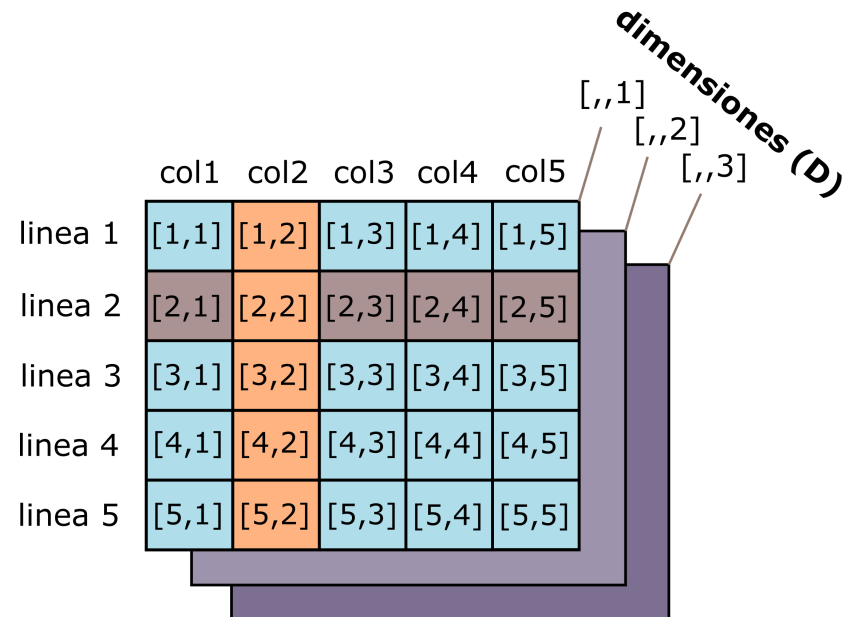


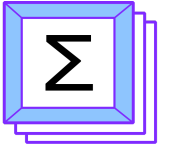
# Manejo de datos

## Indexación

### Gestión de datos $n$ dimensionales [L,C,D]

- Seleccionar elementos



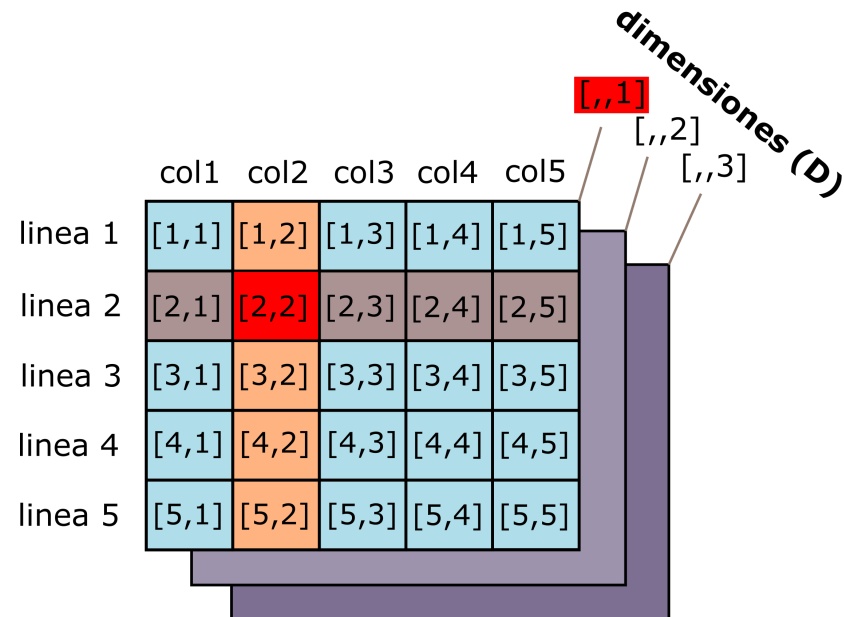


# Manejo de datos

## Indexación

### Gestión de datos $n$ dimensionales [L,C,D]

- Seleccionar elementos



# Manejo de datos



## Indexación

### Gestión de datos $n$ dimensionales [L,C,D]

- Seleccionar elementos

```
ar <- array(data = c(1:8), dim = c(2, 2, 2)); a
```

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

```
ar[,,1] # dimensión 1
```

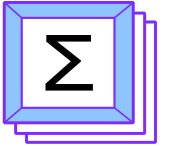
```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
ar[,2,1] # columna 2 y dimensión 1
```

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

```
ar[2,2,1] # línea 2, columna 2 y dimensión 1
```

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



# Manejo de datos

## Indexación

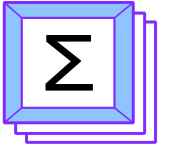
### Gestión de datos $n$ dimensionales [L,C,D]

- Quitar elementos.

The diagram shows a 5x5 grid of data points. The rows are labeled 'linea 1' through 'linea 5' and the columns are labeled 'col1' through 'col5'. The grid contains the following values:

	col1	col2	col3	col4	col5
linea 1	[1,1]	[1,2]	[1,3]	[1,4]	[1,5]
linea 2	[2,1]	[2,2]	[2,3]	[2,4]	[2,5]
linea 3	[3,1]	[3,2]	[3,3]	[3,4]	[3,5]
linea 4	[4,1]	[4,2]	[4,3]	[4,4]	[4,5]
linea 5	[5,1]	[5,2]	[5,3]	[5,4]	[5,5]

Coordinate labels are shown to the right of the grid:  $[,,1]$  points to the first column,  $[,,2]$  points to the second column, and  $[,,3]$  points to the third column. A label 'dimensiones (D)' is positioned above these coordinate labels.

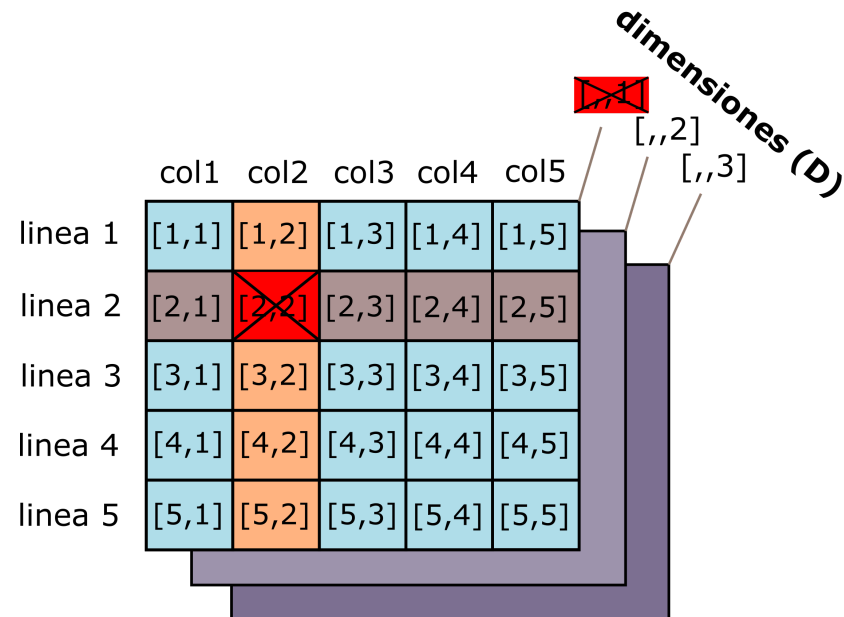


# Manejo de datos

## Indexación

### Gestión de datos $n$ dimensionales [L,C,D]

- Quitar elementos.



# Manejo de datos



## Indexación

### Gestión de datos $n$ dimensionales [L,C,D]

- Quitar elementos.

```
ar
```

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

```
ar[,,-1] # dimensión
```

```
##      [,1] [,2]
## [1,]    5    7
## [2,]    6    8
```

```
ar[,-2,-1] # columna y dimensión
```

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

```
ar[-2,-2,-1] # fila, columna y dimensión
```

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

# Manejo de datos



## Indexación

### Ejercicios

Trabjará con una tabla de datos que no es suya (son datos secundarios). Esta tabla corresponde a datos de pacientes tratados en un hospital de su ciudad.

Pacientes	Barrio	Genéro	Edad	Año	Médico
Paciente_1	12	F	57	2020	A
Paciente_2	28	M	60	2021	B
Paciente_3	7	F	39	2021	C
Paciente_4	20	M	56	2020	A
Paciente_5	19	F	67	2018	B
Paciente_6	8	M	33	2018	C

# Manejo de datos



## Indexación

### Ejercicios

0 - Cree la tabla como un `data.frame` en **R**.

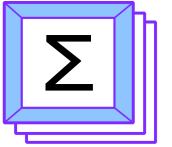
1 - Trabajarás con las variables "género", "edad", "barrio" y "médico". Descarte las otras variables de la tabla.

2 - En su estudio, solo trabajará con personas que se identifiquen con el género femenino. Seleccione solo los datos correspondientes a estas personas.

3 - Agregue una columna que contenga el motivo de la búsqueda de atención médica: Paciente 1 = "tratamiento hormonal", Paciente 3 = "Problemas gástricos", Paciente 5 = "Sospecha de dengue".

4 - Cambie la clase de variable "edad" para `numeric`.





@somaquadrados

# Manejo de datos

## Seleccionar elementos por condición (**vector**)

¿Algún valor en el vector es igual a 11?

11	12	13	11	15	16	== 11?
[1]	[2]	[3]	[4]	[5]	[6]	

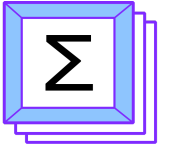
respuesta



T	F	F	T	F	F
---	---	---	---	---	---

T = verdadero  
F = falso

Operadores relacionales con salidas **booleanas** (VERDADERO o FALSO).



@somaquadrados

# Manejo de datos

## Seleccionar elementos por condición (**vector**)

$A == B$  igual

$A != B$  no igual

$A < B$  menor que

$A <= B$  menor o igual que

$A > B$  mayor que

$A >= B$  mayor o igual que

$A | B$  o

$A ! B$  no

$A \%in\% B$  en el conjunto

Comparación de objetos: **A** con **B**.

Operadores relacionales con salidas **booleanas** (VERDADERO o FALSO).

# Manejo de datos



## Seleccionar elementos por condición (**vector**)

- Operadores relacionales con salidas booleanas (VERDADERO o FALSO).

```
A <- 6; B <- 28
```

```
A == B # A es igual a B?
```

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

```
A != B # A es distinto de B?
```

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

```
A > B # A es mayor que B?
```

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

```
A <= B # A menor o igual que?
```

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

```
A >= B # A es mayor o igual que B?
```

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

```
A < B # A es menor que B?
```

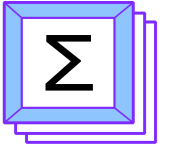
```
## [1] TRUE
```

```
A %in% B # A en B?
```

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

```
A != B # A no es igual B?
```

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



@somaquadrados

# Manejo de datos

## Seleccionar elementos por condición (**vector**)

- Operadores relacionales con salidas **booleanas** (VERDADERO o FALSO).

```
ve <- c(10, 15, 30, 32, 50, 68, 70)
ve
```

```
## [1] 10 15 30 32 50 68 70
```

```
# ¿Qué elementos tienen el valor = 30?
ve == 30
```

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

```
# ¿Qué elementos tienen un valor superior a 30?
ve > 30
```

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

```
# ¿Qué elementos tienen el valor inferior a 50?
ve < 50
```

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

```
# ¿Qué elementos tienen valores mayores o iguales a 45?
ve >= 45
```

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

# Manejo de datos

## Seleccionar elementos por condición (**vector**)

- Elementos del vector
  - ¿Qué pasa si, en lugar de querer saber cuál valor coincide con la condición y cuál no, quisiera seleccionar los valores relacionados con esa condición?

(nombre del objeto)                      (numérico o carácter)

↑                      ↑                      ↑

**vector**[**vector** **condición** **valor**]

↓

(==, !=, >, <, <=, >=)

```
# Antes:  
ve < 30
```

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

```
# Después:  
ve[ve < 30]
```

```
## [1] 10 15
```

# Manejo de datos



## Seleccionar elementos por condición (**vector**)

- Elementos del vector

```
# ¿Qué elementos tienen el valor igual a 30?  
ve[ve == 30]
```

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

```
# ¿Qué elementos valen menos de 50?  
ve[ve < 50]
```

```
## [1] 10 15 30 32
```

```
# ¿Está el ve insertado en el conjunto '32'?  
ve[ve %in% 32]
```

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

```
# ¿Qué elementos tienen un valor superior a 30?  
ve[ve > 30]
```

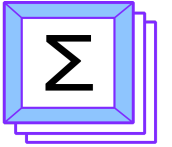
```
## [1] 32 50 68 70
```

```
# ¿Qué elementos tienen valores mayores o iguales a 45?  
ve[ve >= 45]
```

```
## [1] 50 68 70
```

```
# ¿Qué elementos son distintos de 10?  
ve[ve != 10]
```

```
## [1] 15 30 32 50 68 70
```



@somaquadrados

# Manejo de datos

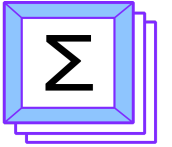
## Seleccionar elementos por condición (**vector**)

### Ejercicios 🐭

Tomó muestras de ratas en tres áreas de Puerto Iguazú. En estos tres anotó la abundancia mensual de ratas durante dos años (24 meses).

```
loc1 <- sample(1:100, 24)
loc2 <- sample(1:100, 24)
loc3 <- sample(1:100, 24)
```

- 1 - En los dos años de recolección, ¿algunos de los sitios presentaron una abundancia superior a 50 ratas en un solo mes? ¿Y abundancia de menos de 20 ratas en un solo mes?
- 3 - ¿Alguna de las ubicaciones tiene los valores `[10, 20, 22]`?
- 4 - Seleccione solo las abundancias que sean mayores que 0 y cree un nuevo vector para cada local.



# Manejo de datos

## Seleccionar elementos por condición (`matrix/data.frame`)

¿Algún valor en la `matrix/data.frame` es igual a 11?

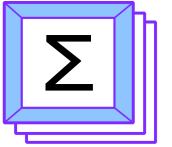
1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

== 11? — respuesta →

F	F	F	F	F
F	F	F	F	F
T	F	F	F	F
F	F	F	F	F
F	F	F	F	F

Operadores relacionales con salidas **booleanas** (VERDADERO o FALSO).





@somaquadrados

# Manejo de datos

## Seleccionar elementos por condición (**matrix/data.frame**)

- Operadores relacionales con salidas **booleanas** (VERDADERO o FALSO).

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

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

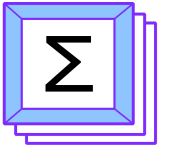
```
# ¿Qué elementos tienen los valores
# = 5 o 6?
ma == c(5, 6)
```

```
##      [,1] [,2] [,3] [,4]
## [1,] FALSE FALSE FALSE FALSE
## [2,] FALSE  TRUE FALSE FALSE
## [3,] FALSE  TRUE FALSE FALSE
```

```
# ¿Qué elementos tienen el valor
# inferior a 8?
ma < 8
```

```
##      [,1] [,2] [,3] [,4]
## [1,] TRUE TRUE  TRUE FALSE
## [2,] TRUE TRUE  FALSE FALSE
## [3,] TRUE TRUE  FALSE FALSE
```

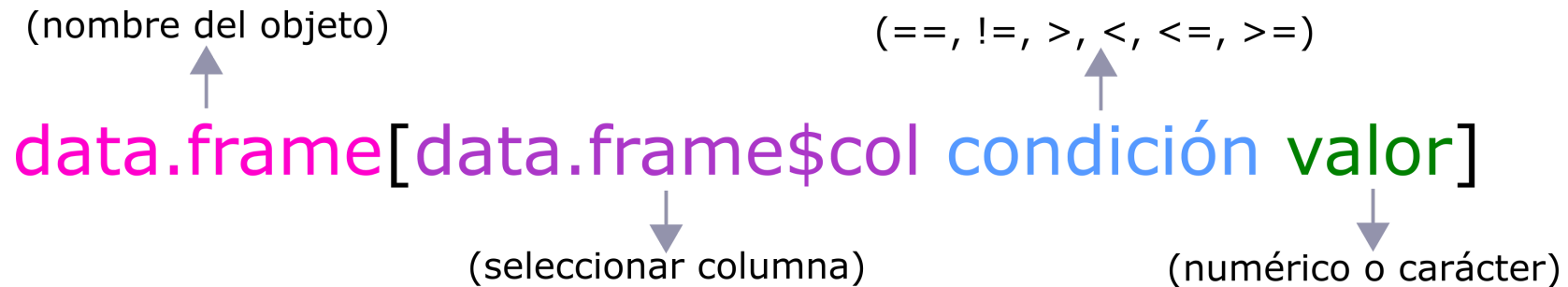
# Manejo de datos



@somaquadrados

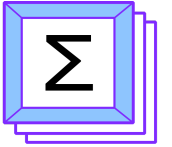
## Seleccionar elementos por condición (data frame)

- Elementos del `data.frame`.
  - ¿Qué pasa si, en lugar de querer saber cuál valor coincide con la condición y cuál no, quisiera seleccionar los valores relacionados con esa condición?



```
ma[ma == 8]
```

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



@somaquadrados

# Manejo de datos

## Seleccionar elementos por condición (**matrix/data.frame**)

- Operadores relacionales con salidas **booleanas** (VERDADERO o FALSO).

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

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

```
# ¿Qué elementos tienen los valores
# = 5 o 6?
ma[ma == c(5, 6)]
```

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

```
# ¿Qué elementos tienen el valor
# inferior a 8?
ma[ma < 8]
```

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

# Manejo de datos



## Seleccionar elementos por condición (**data.frame**)

- En un **data.frame**, podemos separar la columna que nos interesa con el operador `$` y luego aplicar la selección por condición.

```
# Nombra las columnas y transforma la matriz en data.frame
colnames(ma) = c("A", "B", "C", "D")
mb <- data.frame(ma)
mb
```

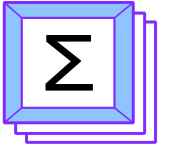
```
##   A B C  D
##  1 1 4 7 10
##  2 2 5 8 11
##  3 3 6 9 12
```

```
mb$A == 1 # Para la columna "A"
```

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

```
mb$B > 5 # Para la columna "B"
```

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



# Manejo de datos

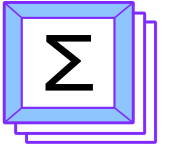
## Seleccionar elementos por condición (`matrix/data.frame`)

### Ejercicios

Tiene una tabla con valores de temperatura, lluvia y abundancia de flebotomos que muestreó en los últimos 5 meses:

temperatura	lluvia	abundancia
19	70	61
30	8	81
22	55	62
18	2	24
26	52	29

- 1 - En alguno de los meses, ¿la temperatura bajó de los 20°C?
- 2 - En alguno de los meses, ¿tomó muestras de más de 20 flebotomos?
- 3 - Seleccione valores de precipitación superiores a 50.
- 4 - ¿Hay un mes con abundancia de flebotomos igual a 10, 20 o 31?
- 5 - En toda la tabla, ¿hay algún valor cero?



@somaquadrados

# Manejo de datos

## Seleccionar elementos por condición (**character**)

- Todo lo que hemos hecho hasta ahora se puede hacer con characters.

```
l <- c("A", "B", "C", "d")
```

```
l < "C"
```

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

```
l == "B"
```

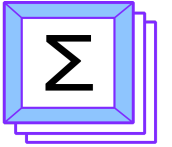
```
## [1] FALSE TRUE FALSE FALSE
```

```
l[l > "B"]
```

```
## [1] "C" "d"
```

```
l[l != "d"]
```

```
## [1] "A" "B" "C"
```

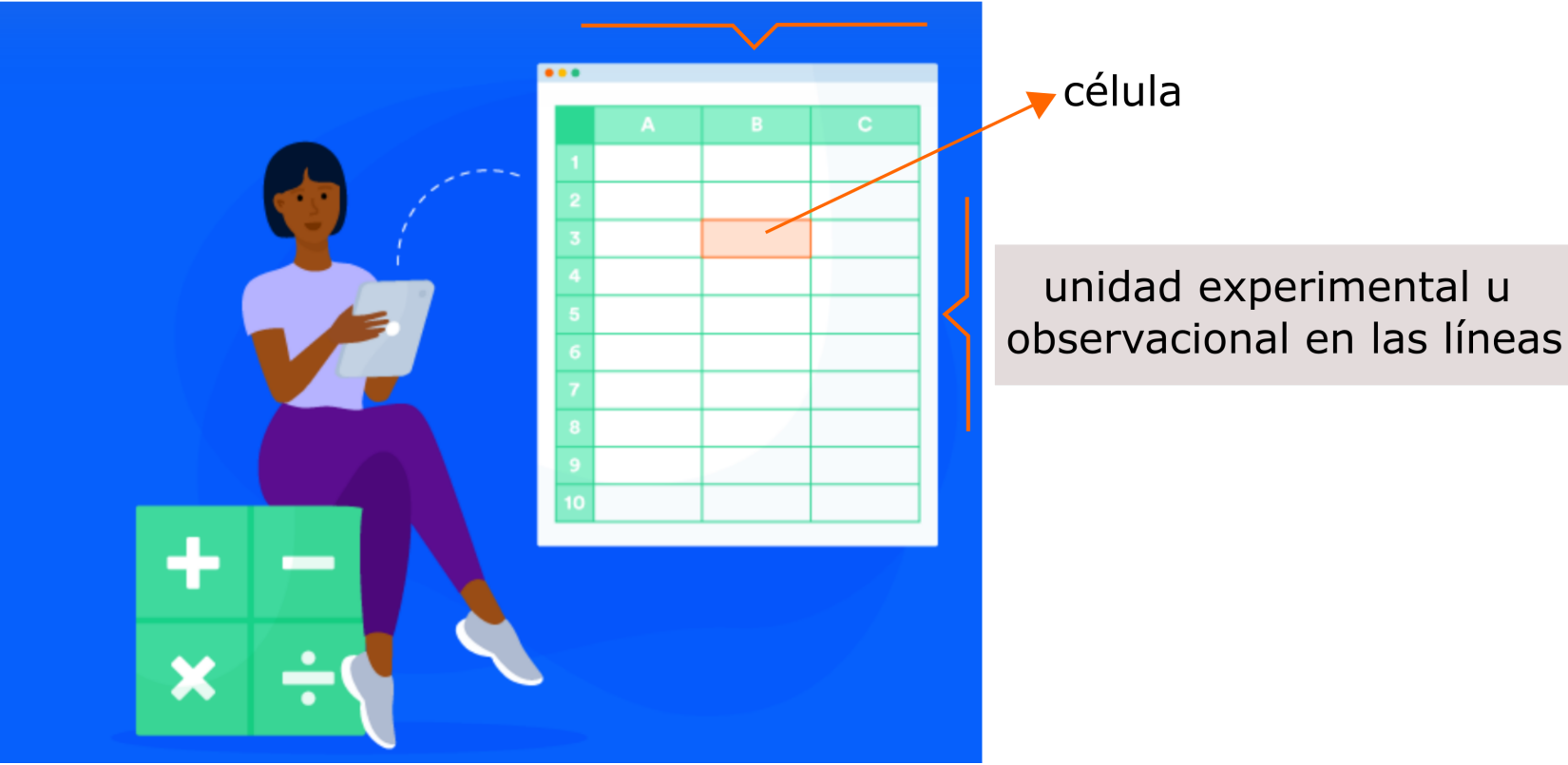


@somaquadrados

# ¿Cómo armar mi tabla de datos?

# ¿Cómo armar mi tabla de datos?

Variables en columnas

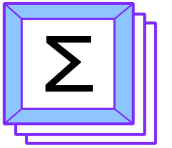


célula

unidad experimental u observacional en las líneas



# ¿Cómo armar mi tabla de datos?



@somaquadrados

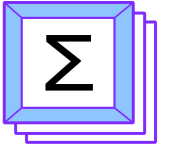
Variables en columnas

	UE	Ano	Var_A
1	A	2001	1
2	A	2002	2
3	A	2003	3
4	B	2001	2
5	B	2002	3
6	B	2003	4
7	C	2001	2
8	C	2002	3
9	C	2003	1
10	D	2001	0

célula

unidad experimental u observacional en las líneas

# ¿Cómo armar mi tabla de datos?



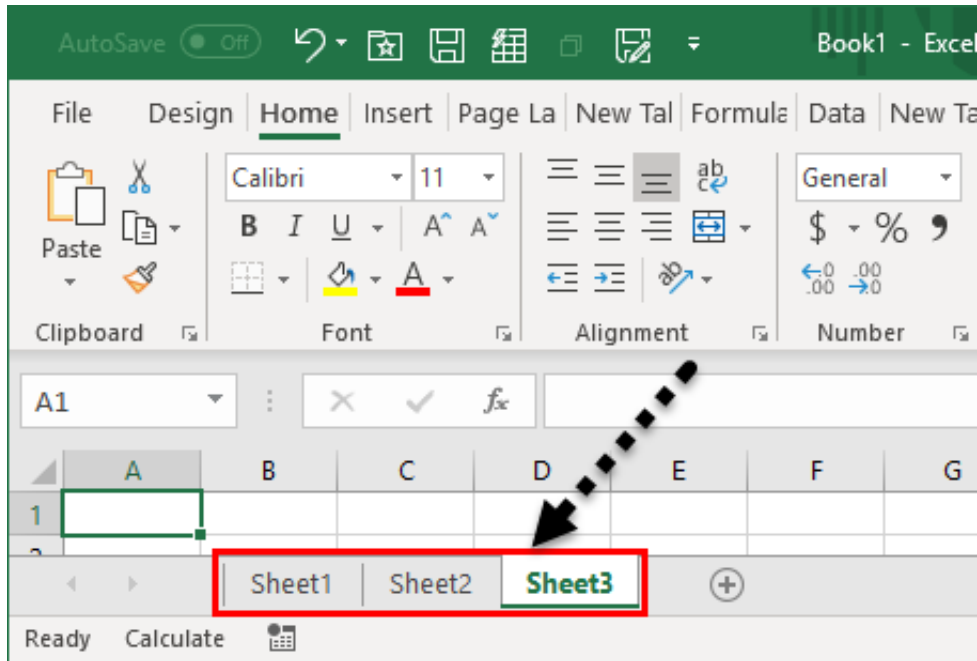
@somaquadrados

## Resumen

Características principales de un conjunto de datos ordenado:

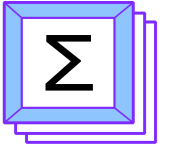
- cada variable es una columna
- cada observación es una línea
- cada valor está en una celda diferente

# ¿Cómo armar mi tabla de datos?



## Descripción de datos:

- Utilice las hojas de datos de su editor (ex. excel) para almacenar información sobre su tabla.
- En la primera hoja (hoja 1) dejamos nuestra tabla y en las demás (normalmente la hoja 2, pero si es necesario usamos otras) incluimos información sobre nuestra tabla, como a qué se refieren los datos, descripción de cada variable, alguna observación importante y alguna fecha de edición de esta tabla.

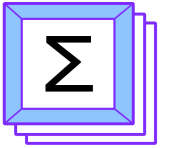


@somaquadrados


# Tidyverse

<https://www.tidyverse.org/>

# Tidyverse



@somaquadrados

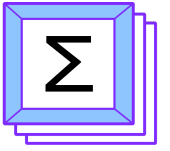
El tidyverse es una colección obstinada de  R diseñados para la ciencia de datos. Todos los paquetes comparten una filosofía de diseño, una gramática y estructuras de datos subyacentes.



- Instale el tidyverse completo con:

```
install.packages("tidyverse")
```

# Tidyverse



@somaquadrados



# Tidyverse



Tidyverse es una colección de  R

- [readr](#) - importación de datos
- [tibble](#) - formato de `data.frame` mejorado
- [tidyr](#), [dplyr](#) - manipulación de datos
- [ggplot2](#) - visualizando de datos
- [purrr](#) - programación avanzada
- [forcats](#) - trabajando con factores
- [stringr](#) - trabajando con cadena de caracteres

# Tidyverse



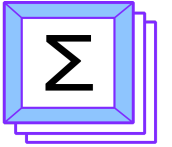
Tidyverse es una colección de  R

- `readr` - importación de datos
- `tibble` - formato de `data.frame` mejorado
- `tidyr`, `dplyr` - manipulación de datos
- `ggplot2` - visualizando de datos
- `purrr` - programación avanzada
- `forcats` - trabajando con factores
- `stringr` - trabajando con cadena de caracteres



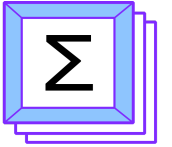
# Tidyverse

Flujo de trabajo en ciencia de datos, con **Tidyverse**



@somaquadrados

# Tidyverse



@somaquadrados

El creador de **Tidyverse** es Hadley Wickham y hoy en día muchas personas han contribuido a su expansión.



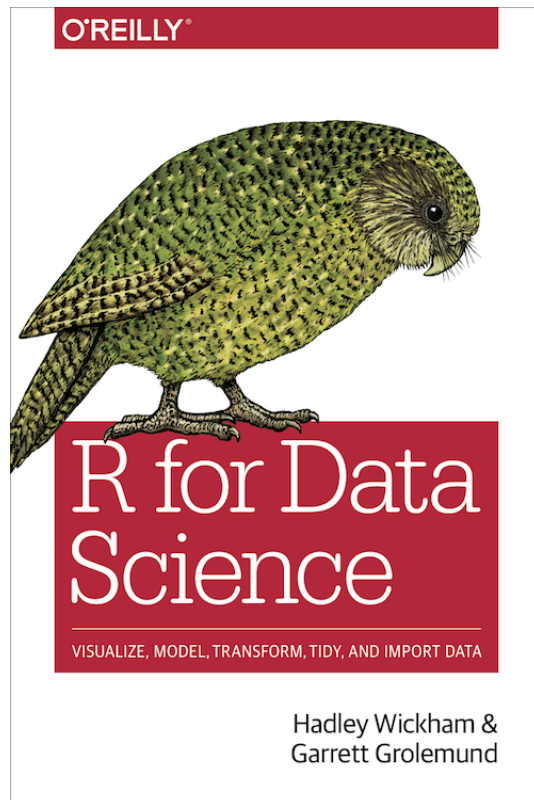
 <http://hadley.nz/>

 [@hadleywickham](https://twitter.com/hadleywickham)

# Tidyverse



## R for Data Science



"This book will teach you how to do data science with **R**: You'll learn how to get your data into **R**, get it into the most useful structure, transform it, visualise it and model it. In this book, you will find a practicum of skills for data science. Just as a chemist learns how to clean test tubes and stock a lab, you'll learn how to clean data and draw plots—and many other things besides. (...)"

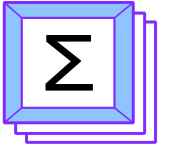
<https://r4ds.had.co.nz/>

# Tidyverse

## Ejercicio

Instale y cargue el paquete **Tidyverse** en su computadora.

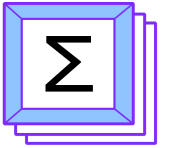




@somaquadrados

# Importar datos a R

# Importar datos a R

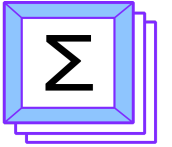


@somaquadrados

Para su alivio, **no es necesario producir su tabla en R** (como lo hemos hecho hasta ahora). Es posible construir la tabla en Excel y luego importar los datos (de HD a nuestra memoria RAM).




La función de importación dependerá del formato en el que se guardó nuestra tabla (.txt, .csv, .xls, .xlsx).





@somaquadrados

# Importar datos a R

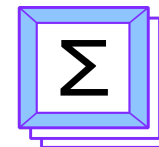
## Working directory

- Recuerde que el directorio de trabajo es una  donde **R** lee y guarda archivos.
- Deberá decirle a R dónde están los archivos en los que va a trabajar.

```
# Aquí incluirá la dirección donde están sus archivos en su computadora.  
setwd("C:/Users/mmfav/introductoryR/clase_2/data")
```


- Deje todos los archivos guardados en esta misma carpeta, esto facilitará su trabajo.
- Tenga en cuenta que la dirección aquí se indica con barras invertidas (/), a diferencia de lo que usan algunos sistemas operativos (\). Por ejemplo:
  - : C:\Users\mmfav\introductoryR\clase\_2\data
  - : C:/Users/mmfav/introductoryR/clase\_2/data

# Importar datos a R



@somaquadrados



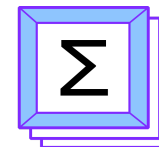
El  **tidyverse** `readr` se usa para importar archivos de texto, como `.txt` o `.csv` a **R**.

`reader` transforma archivos de texto en **tibbles**.

- `read_csv()`; `read_csv2()`: para archivos separados por comas.
- `read_tsv()`: para archivos separados por tabulaciones.
- `read_delim()`: para archivos separados por un delimitador genérico. El argumento `delim =` indica qué carácter separa cada columna del archivo de texto.
- `read_table()`: para archivos de texto tabulares con columnas separadas por espacios.
- `read_fwf()`: para archivos compactos que deben tener el ancho de cada columna especificado.
- `read_log()`: para archivos de registro estándar.




# Importar datos a R



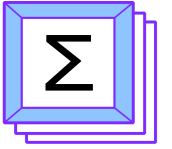
@somaquadrados



El  **tidyverse** `readr` se usa para importar archivos de texto, como `.txt` o `.csv` a **R**.

`readr` transforma archivos de texto en **tibbles**.

- `read_csv()`; `read_csv2()`: para archivos separados por comas.
- `read_tsv()`: para archivos separados por tabulaciones.
- `read_delim()`: para archivos separados por un delimitador genérico. El argumento `delim` indica qué carácter separa cada columna del archivo de texto.
- `read_table()`: para archivos de texto tabulares con columnas separadas por espacios.
- `read_fwf()`: para archivos compactos que deben tener el ancho de cada columna especificado.
- `read_log()`: para archivos de registro estándar.



@somaquadrados

# Importar datos a R

## readr: **.csv**

- Como ejemplo, usaremos la base de datos que proporcionamos en el repositorio ([datos.csv](#)).
- La función para leer los datos es: `read_csv2(file = "archivo.csv")`.

```
datos_csv <- read_csv2(file = "datos.csv")
```

```
## i Using "','" as decimal and "'.'" as grouping mark. Use `read_delim()` for more control.
```

```
##
```

```
## -- Column specification -----
```

```
## cols(
```

```
##   ID = col_double(),
```

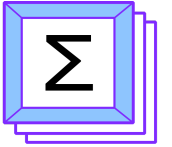
```
##   localidad = col_character(),
```

```
##   `2001` = col_double(),
```

```
##   `2002` = col_double(),
```

```
##   `2003` = col_double()
```

```
## )
```



@somaquadrados

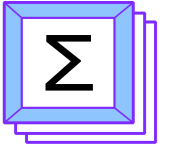
# Importar datos a R

## readr: **.txt**

- Como ejemplo, usaremos la base de datos que proporcionamos en el repositorio ([datos.txt](#)).
- La función para leer los datos es: `read_delim(file = "archivo.txt", delim = "\t")`.

```
datos_txt <- read_delim(file = "datos.txt", delim = "\t")
```

```
##  
## -- Column specification -----  
## cols(  
##   ID = col_double(),  
##   localidad = col_character(),  
##   variable = col_character(),  
##   value = col_double()  
## )
```



@somaquadrados

# Importar datos a R

## Exportar datos (**write\_**)


- Para la mayoría de las funciones `read_`, existe una función `write_` correspondiente.
- Estas funciones sirven para guardar bases en un formato de archivo específico.
- Debe especificar el objeto a exportar y el nombre del archivo con la extensión.

```
# archivo .csv  
write.csv2(x = objeto, path = "nombre_tabla.csv")  
  
# como un .txt  
write_delim(x = objeto, path = "nombre_tabla.txt", delim = "\t")
```

# Importar datos a R

¿Qué pasa si mis datos se guardan como un archivo **excel**?



El  `readxl` se usa para importar archivos de excel, como `.xlsx` o `.xls` a **R**.

- Instalar:

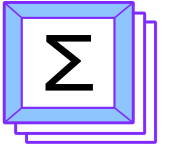
```
install.packages("readxl")
```

- Cargar el paquete:

```
library(readxl)
```

- Para abrir nuestro **archivo de repositorio** (`datos.xlsx`): `read_xlsx("archivo.xlsx")`
- `readxl` transforma archivos de excel en **tibbles**.

- 
- **¡¡No es parte del tidyverse !!**
-



@somaquadrados

# Importar datos a R

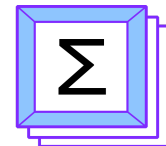
## readxl: .xlsx

### Ejemplo:

```
datos_xlsx <- read_xlsx("datos.xlsx")
datos_xlsx
```

```
## # A tibble: 24 x 6
##   ID localidad      ano zona  var_respuesta temperatura
##   <dbl> <chr>          <dbl> <chr>          <dbl>          <dbl>
## 1     1 1 Puerto Iguazú - Misiones 2001 agrícola      NA           25
## 2     2 2 Puerto Libertad - Misiones 2001 agrícola      2           26.7
## 3     3 3 San Pedro - Misiones      2001 agrícola      5           24.2
## 4     4 4 Tareiri - Misiones        2001 agrícola      4           26.2
## 5     5 5 Puerto Iguazú - Misiones 2005 agrícola      6           27
## 6     6 6 Puerto Libertad - Misiones 2005 agrícola     NA           25.5
## 7     7 7 San Pedro - Misiones      2005 agrícola      7           26
## 8     8 8 Tareiri - Misiones        2005 agrícola      9           26.5
## 9     9 9 Puerto Iguazú - Misiones 2001 bosque       30           20
## 10    10 10 Puerto Libertad - Misiones 2001 bosque       52           21
## # ... with 14 more rows
```

# Importar datos a R



@somaquadrados



Un **tibble**, o `tbl_df`, es una reinención moderna del `data.frame`, manteniendo el tiempo que ha demostrado ser efectivo y descartando lo que no lo es.

- 
- Es un formato requerido para usar funciones **tidyverse**.
  - Las variables pueden ser de tipo *numérico* (`int`, `dbl`), *carácter* (`chr`), *lógicas* (`lgl`) y *factor* (`fctr`)
- 

- Convertir:

| `data.frame` en `tibble`:

```
as_tibble(data)
```

---

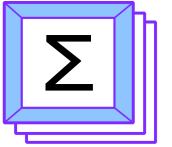
# Importar datos a R



## Ejercicio:

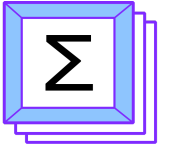
- 1 - Importe una tabla relacionada con una de sus investigaciones para el R; puede tener cualquier format y numero de variables. El formato de salida debe ser `tibbler`.
- 2 - ¿Cuál es la diferencia entre las tablas `datos_csv`, `datos_txt` y `datos_xlsx`?





@somaquadrados


# Operador pipe (%>%)



@somaquadrados

# Operador pipe (%>%)



El  **magrittr** ofrece un operador que hace que su código sea más legible: el pipe (%>%).

La idea del operador pipe (%>%) es bastante simple: use el valor resultante de la expresión de la izquierda como primer argumento de la función de la derecha.

## Por ejemplo:

```
# suma el vector y luego obtén la √ (sqrt):  
x <- 1:10
```

```
# Sin el pipe:  
sqrt(sum(x))
```

```
## [1] 7.416198
```

```
# Con el pipe:  
x %>% sum() %>% sqrt()
```

```
## [1] 7.416198
```

# Operador pipe (%>%)



## Para hacerlo un poco más intuitivo:

- Hacer empanadas SIN el pipe:

```
enfriarse(  
  hornear(  
    cerrar_la_masa(  
      agregar_relleno(  
        abre_la_massa(  
          mezclar(  
            hacer_la_masa(rep(farina, 5), mante  
              hasta = 'masa homogénea'),  
              tipo = "carne picada"),  
          ),  
        ),  
      temperatura = 180, tiempo = 20),  
    )  
  )  
)
```

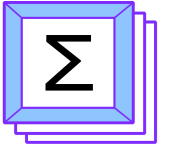
- Hacer empanadas CON el pipe:

```
hacer_la_masa(rep(farina, 5), manteca, sal, agu  
mezclar(hasta = 'masa homogénea') %>%  
abre_la_massa() %>%  
agregar_relleno(tipo = "carne picada") %>%  
cerrar_la_masa() %>%  
hornear(temperatura = 180, tiempo = 20) %>%  
enfriarse()
```

---

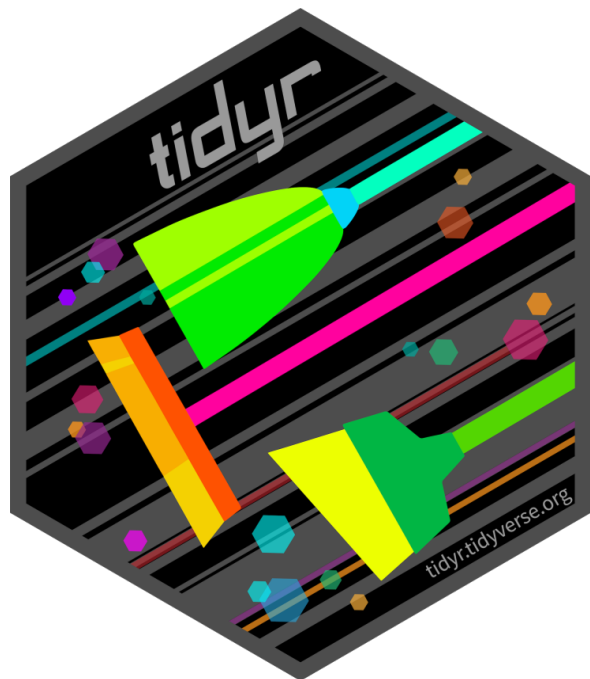
**!!** El código no solo es más pequeño, es más intuitivo, la lectura se vuelve mucho más fácil **!!**

---



@somaquadrados

# tidyr



El objetivo del  tidy es ayudarte a crear datos ordenados.

Los datos ordenados son datos donde:

- Cada columna es variable.
- Cada fila es una observación.
- Cada celda es un valor único.

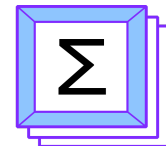
---

Tidy data describe una forma estándar de almacenar datos que se utiliza siempre que sea posible en **tidyverse**.

Si se asegura de que sus datos estén ordenados, pasará menos tiempo luchando con las herramientas y más tiempo trabajando en su análisis.

---

# tidyr



@somaquadrados

- Estas son sus principales funciones:
  - `separate()`: separar los caracteres en varias columnas
  - `unite()`: unir datos de varias columnas en una
  - `drop_na()`: eliminar líneas con NA
  - `replace_na()`: reemplazar valores NA
  - `pivot_wider()`: pasa valores de filas a columnas
  - `pivot_longer()`: pasa valores de columnas a filas

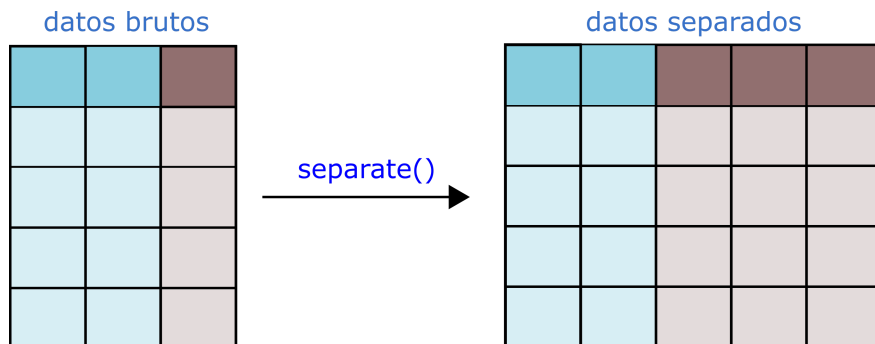
---

Para ver todas las funciones del paquete, consulte la *cheatsheets*:  
<https://github.com/rstudio/cheatsheets/blob/master/data-import.pdf>

---

## separate()

- Muchas veces, una sola variable de columna capturará múltiples variables, o incluso partes de una variable que simplemente no le importa.

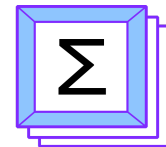


- La función `separate()` separa dos o más variables que están concatenadas en la misma columna.

- La sintaxis de la función es:

```
datos %>%
  separate(
    col = columna_vieja,
    into = c("nueva_columna_1", "nueva_columna_2"),
    sep = c("_") # cómo se separan las variables
  )
```

# tidyr



@somaquadrados

## separate()

Por ejemplo, dividamos la columna "localidad" de la tabla "datos\_xlsx" en "ciudad" y "provincia".

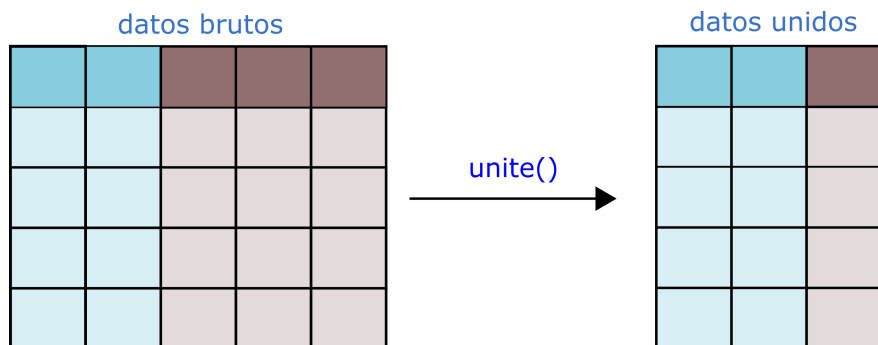
```
datos_xlsx %>%  
  separate(  
    col = localidad, # la columna vieja  
    into = c("ciudad", "provincia"), # las nuevas columnas  
    sep = c(" - ") # modo de separación  
  )
```

ID	ciudad	provincia	ano	zona	var_respuesta	temperatura
1	Puerto Iguazú	Misiones	2001	agrícola	NA	25.0
2	Puerto Libertad	Misiones	2001	agrícola	2	26.7
3	San Pedro	Misiones	2001	agrícola	5	24.2
4	Tareiri	Misiones	2001	agrícola	4	26.2
5	Puerto Iguazú	Misiones	2005	agrícola	6	27.0



## unite()

- La operación `unite()` es lo opuesto a `separate()`.



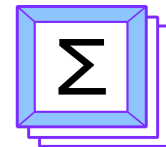
- La función `unite()` une dos variables que están en columnas diferentes.

- Toma dos columnas (variables) y las convierte en una. Se usa ampliamente para ensamblar informes finales o tablas para análisis visual.

- La sintaxis de la función es:

```
datos %>%  
  unite(  
    col = nueva_columna, columnas_para_juntar,  
    sep = c("_") # cómo se separan las variables  
  )
```

# tidyr



@somaquadrados

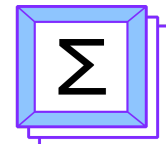
## unite()

Por ejemplo, unamos las columnas "zona" y "año".

```
datos_xlsx %>%  
  unite(  
    col = "zona_ano", "zona", "ano",  
    sep = "_"  
  )
```

ID	localidad	zona_ano	var_respuesta	temperatura
1	Puerto Iguazú - Misiones	agrícola_2001	NA	25.0
2	Puerto Libertad - Misiones	agrícola_2001	2	26.7
3	San Pedro - Misiones	agrícola_2001	5	24.2
4	Tareiri - Misiones	agrícola_2001	4	26.2
5	Puerto Iguazú - Misiones	agrícola_2005	6	27.0
6	Puerto Libertad - Misiones	agrícola_2005	NA	25.5

# tidyr



@somaquadrados

## replace\_na() y drop\_na()

Cuando tenemos datos faltantes en nuestra tabla (NA), podemos reemplazar NA con nuevos valores con la función `replace_na()`,...

```
replace_na(  
  list(columna_X = valor)  
)
```

...o podemos eliminar las filas con valores faltantes con `drop_na()`.

```
drop_na(  
  columna  
)
```

valores faltantes (NA)

		NA
	NA	

replace\_na()

drop\_na()

datos sustitutos

		1
	1	


sin líneas NA

## replace\_na()

**Por ejemplo**, podemos reemplazar las filas con el valor faltante en la columna "var\_respuesta" por cero (lineas 1, 6 y 20)...

```
datos_xlsx %>%  
  replace_na(list(var_respuesta = 0))
```

ID	localidad	ano	zona	var_respuesta	temperatura
<b>1</b>	<b>Puerto Iguazú - Misiones</b>	<b>2001</b>	<b>agrícola</b>	<b>0</b>	<b>25.0</b>
2	Puerto Libertad - Misiones	2001	agrícola	2	26.7
3	San Pedro - Misiones	2001	agrícola	5	24.2
4	Tareiri - Misiones	2001	agrícola	4	26.2
5	Puerto Iguazú - Misiones	2005	agrícola	6	27.0
<b>6</b>	<b>Puerto Libertad - Misiones</b>	<b>2005</b>	<b>agrícola</b>	<b>0</b>	<b>25.5</b>
7	San Pedro - Misiones	2005	agrícola	7	26.0

## drop\_na()

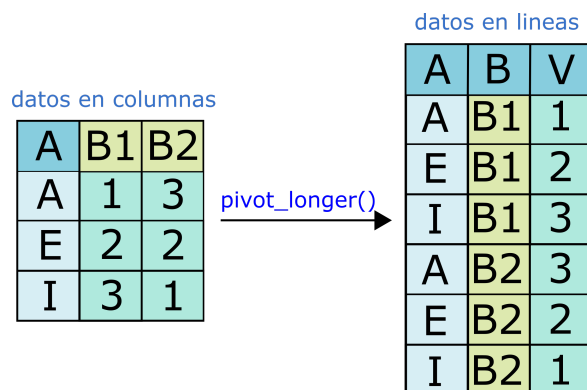
...o podemos eliminar las líneas que tiene valores NA.

```
datos_xlsx %>%  
  drop_na(var_respuesta)
```

ID	localidad	ano	zona	var_respuesta	temperatura
2	Puerto Libertad - Misiones	2001	agrícola	2	26.7
3	San Pedro - Misiones	2001	agrícola	5	24.2
4	Tareiri - Misiones	2001	agrícola	4	26.2
5	Puerto Iguazú - Misiones	2005	agrícola	6	27.0
7	San Pedro - Misiones	2005	agrícola	7	26.0
8	Tareiri - Misiones	2005	agrícola	9	26.5
9	Puerto Iguazú - Misiones	2001	bosque	30	20.0

## pivot\_longer()

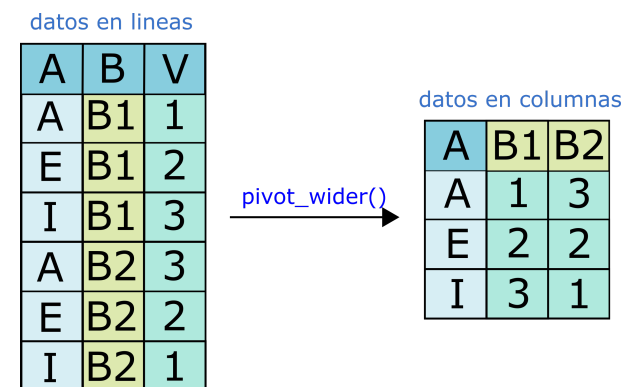
- "Alarga" los datos, aumentando el número de filas y disminuyendo el número de columnas.



```
pivot_longer(  
  cols = Columnas_para_pivotar,  
  names_to = "nombre_nova_columna",  
  values_to = "nombre_col_valores"  
)
```

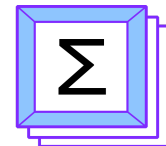
## pivot\_wider()

- Pasa los datos de columnas para filas, aumentando el número de columnas.



```
pivot_wider(  
  names_from = columna_nombres,  
  values_from = columna_valores  
)
```

# tidyr



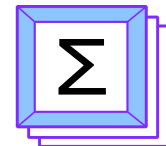
@somaquadrados

## pivot\_longer()

**Por ejemplo**, pasemos los años de la tabla "datos\_csv" de las columnas a las filas. Los valores los almacenaremos en una columna llamada 'value'.

```
datos_csv %>%  
  pivot_longer(  
    cols = c('2001', '2002', '2003'),  
    names_to = "año",  
    values_to = "value"  
  )
```

ID	localidad	año	value
1	Agricola	2001	10
1	Agricola	2002	12
1	Agricola	2003	15
2	PNI	2001	20
2	PNI	2002	22
2	PNI	2003	25
3	Urbano	2001	15
3	Urbano	2002	12
3	Urbano	2003	10



## pivot\_wider()

**Por ejemplo**, pasemos los elementos de la columna "variable" a las columnas y usemos la columna "valor" para los valores (tabla "data\_txt").

```
datos_txt %>%  
  pivot_wider(  
    names_from = variable,  
    values_from = value  
  )
```

ID	localidad	temperatura	pluviosidad
1	Agricola	25	10
2	PNI	22	23
3	Urbano	30	30



## Ejercicios

1 - Abra la tabla "*tidy\_ej.xlsx*" en R.

2 - ¿Faltan datos en la tabla *tidy\_ej*? Si es así, reemplace los valores faltantes con 0.

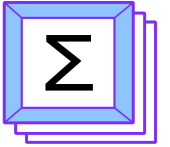
### Tip:

```
tidy_ej %>% is.na() %>% table()
```

3 - Separe la columna "departamento - año" en dos columnas.

4 - Mueva las columnas "PNI", "Urban" y "Rural" a una sola columna y agregue los valores a una nueva columna llamada "número de accidentes".

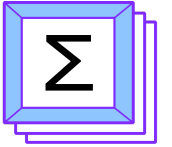
5 - Guarde todo lo que ha hecho en una nueva table **tibble** llamada "tidy\_ej2".



@somaquadrados


# dplyr

# dplyr



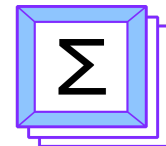
@somaquadrados



El  `dplyr` es lo paquete más útil para realizar la transformación de datos, combinando simplicidad y eficiencia de una manera elegante.

- Los scripts **R** que hacen un uso inteligente de los verbos `dplyr` y las facilidades del operador `pipe` tienden a ser más legibles y organizados sin perder velocidad de ejecución.
- 
- Las principales funciones de `dplyr` son:
    - `select()`: seleccionar columnas
    - `arrange()`: ordenar la base de datos
    - `filter()`: filtrar las líneas
    - `mutate()`: crear/modificar columnas
    - `group_by()`: agrupar la base de datos
    - `summarise()`: resume la base
    - `relocate()`: reordenar columnas
    - `left_join()`, `right_join()` y `full_join()`: juntar ≠ bases de datos.

# dplyr



@somaquadrados

## select()

- Usamos para seleccionar columnas.
- Los argumentos son los nombres de las columnas que desea seleccionar.

```
datos %>%  
  select(nombre_col, nombre_col2)
```

- Para eliminar columnas de la base, agregue un "menos" (-) antes de la selección.

```
datos %>%  
  select(-nombre_col, -nombre_col2)
```

- También disponemos de otras funciones auxiliares:
  - `starts_with()`: para columnas que comienzan con texto estándar
  - `ends_with()`: para columnas que terminan con texto estándar
  - `contiene()`: para columnas que contienen texto estándar

# dplyr



## select()

Seleccionemos las columnas "localidad" y "año" de la tabla "datos\_xlsx".

```
datos_xlsx %>%  
  select(localidad, ano)
```

```
## # A tibble: 24 x 2  
##   localidad      ano  
##   <chr>         <dbl>  
## 1 Puerto Iguazú - Misiones 2001  
## 2 Puerto Libertad - Misiones 2001  
## 3 San Pedro - Misiones 2001  
## 4 Tareiri - Misiones 2001  
## 5 Puerto Iguazú - Misiones 2005  
## 6 Puerto Libertad - Misiones 2005  
## 7 San Pedro - Misiones 2005  
## 8 Tareiri - Misiones 2005  
## 9 Puerto Iguazú - Misiones 2001  
## 10 Puerto Libertad - Misiones 2001  
## # ... with 14 more rows
```

# dplyr



## select()

Seleccione todos los datos excepto "ID" y "temperatura".

```
datos_xlsx %>%  
  select(-ID, -temperatura)
```

```
## # A tibble: 24 x 4  
##   localidad          ano zona    var_respuesta  
##   <chr>              <dbl> <chr>      <dbl>  
## 1 Puerto Iguazú - Misiones 2001 agrícola      NA  
## 2 Puerto Libertad - Misiones 2001 agrícola      2  
## 3 San Pedro - Misiones      2001 agrícola      5  
## 4 Tareiri - Misiones        2001 agrícola      4  
## 5 Puerto Iguazú - Misiones 2005 agrícola      6  
## 6 Puerto Libertad - Misiones 2005 agrícola      NA  
## 7 San Pedro - Misiones      2005 agrícola      7  
## 8 Tareiri - Misiones        2005 agrícola      9  
## 9 Puerto Iguazú - Misiones 2001 bosque       30  
## 10 Puerto Libertad - Misiones 2001 bosque       52  
## # ... with 14 more rows
```

# dplyr



## arrange()

- Para ordenar líneas.
- Los argumentos son las columnas por las que queremos ordenar las filas.

```
datos %>%  
  arrange(columna_x)
```

- También podemos ordenar en orden descendente usando la función `desc()`...

```
datos %>%  
  arrange(desc(columna_x))
```

- ¡Y ordena más de una columna al mismo tiempo!

```
datos %>%  
  arrange(columna_y, desc(columna_x))
```

# dplyr

## arrange()

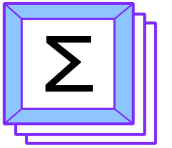
- En el siguiente ejemplo, ordenamos las líneas base en orden ascendente de "response\_var".

```
datos_xlsx %>%  
  arrange(var_respuesta, desc(ano))
```

```
## # A tibble: 24 x 6  
##       ID localidad      ano zona  var_respuesta temperatura  
##   <dbl> <chr>      <dbl> <chr>      <dbl>      <dbl>  
## 1     2 Puerto Libertad - Misiones 2001 agrícola      2        26.7  
## 2     4 Tareiri - Misiones      2001 agrícola      4        26.2  
## 3     3 San Pedro - Misiones     2001 agrícola      5        24.2  
## 4     5 Puerto Iguazú - Misiones 2005 agrícola      6         27  
## 5     7 San Pedro - Misiones     2005 agrícola      7         26  
## 6     8 Tareiri - Misiones     2005 agrícola      9        26.5  
## 7    18 Puerto Libertad - Misiones 2001 urbano     14        29.4  
## 8    23 San Pedro - Misiones     2005 urbano     16         33  
## 9    19 San Pedro - Misiones     2001 urbano     16        32.4  
## 10   24 Tareiri - Misiones     2005 urbano     18        33.6  
## # ... with 14 more rows
```



# dplyr



@somaquadrados

## filter()

- Para filtrar valores de una columna base, usamos la función `filter()`.

```
datos %>%  
  filter(columna < value)
```

- Por ejemplo, podemos seleccionar datos con una "var\_respuesta" superior a 50.

```
datos_xlsx %>%  
  filter(var_respuesta > 50)
```

```
## # A tibble: 4 x 6  
##       ID localidad          ano zona  var_respuesta temperatura  
##   <dbl> <chr>          <dbl> <chr>          <dbl>          <dbl>  
## 1    10 Puerto Libertad - Misiones 2001 bosque          52            21  
## 2    11 San Pedro - Misiones      2001 bosque          63            22  
## 3    15 San Pedro - Misiones      2005 bosque          56            24  
## 4    16 Tareiri - Misiones         2005 bosque          64            26
```

# dplyr



## filter()

- También podemos usar el filtro con caracteres.

```
datos_xlsx %>%  
  filter(zona %in% "urbano")
```

```
## # A tibble: 8 x 6  
##   ID localidad      ano zona  var_respuesta temperatura  
##   <dbl> <chr>          <dbl> <chr>          <dbl>          <dbl>  
## 1  17 Puerto Iguazú - Misiones  2001 urbano          20           28  
## 2  18 Puerto Libertad - Misiones  2001 urbano          14          29.4  
## 3  19 San Pedro - Misiones      2001 urbano          16          32.4  
## 4  20 Tareiri - Misiones         2001 urbano          NA           30  
## 5  21 Puerto Iguazú - Misiones  2005 urbano          29           29  
## 6  22 Puerto Libertad - Misiones  2005 urbano          30          32.5  
## 7  23 San Pedro - Misiones      2005 urbano          16           33  
## 8  24 Tareiri - Misiones         2005 urbano          18          33.6
```

# dplyr



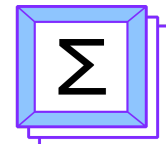
## mutate()

- Para modificar una columna existente o crear una nueva.
- Aplicaremos una función.
- La regla es que el resultado de la operación devuelve un vector con una longitud igual al número de filas en la base.

```
datos_xlsx %>%  
  mutate(columna = columna+función)
```

- También puede crear/modificar tantas columnas como desee dentro de la misma mutación.

```
datos_xlsx %>%  
  mutate(columna = columna+función, nueva_columna = columna/valor)
```



## mutate()

- Por ejemplo, digamos que descubrió un error en su tabla. Agregaste uno individuo más en la columna "var\_respuesta" y ahora necesitas quitar ese valor.

```
datos_xlsx %>%  
  mutate(var_respuesta = var_respuesta - 1)
```

```
## # A tibble: 24 x 6  
##       ID localidad      ano zona  var_respuesta temperatura  
##   <dbl> <chr>      <dbl> <chr>      <dbl>      <dbl>  
## 1     1 1 Puerto Iguazú - Misiones 2001 agrícola      NA         25  
## 2     2 2 Puerto Libertad - Misiones 2001 agrícola      1         26.7  
## 3     3 3 San Pedro - Misiones      2001 agrícola      4         24.2  
## 4     4 4 Tareiri - Misiones         2001 agrícola      3         26.2  
## 5     5 5 Puerto Iguazú - Misiones 2005 agrícola      5         27  
## 6     6 6 Puerto Libertad - Misiones 2005 agrícola     NA         25.5  
## 7     7 7 San Pedro - Misiones      2005 agrícola      6         26  
## 8     8 8 Tareiri - Misiones         2005 agrícola      8         26.5  
## 9     9 9 Puerto Iguazú - Misiones 2001 bosque       29         20  
## 10    10 10 Puerto Libertad - Misiones 2001 bosque       51         21  
## # ... with 14 more rows
```

# dplyr



## summarize()

- Es la técnica de resumir un conjunto de datos utilizando alguna métrica de interés.
- Media, mediana, varianza, frecuencia, proporción, por ejemplo, son tipos de resumen que aportan información diferente sobre una variable.

```
datos %>%  
  summarize(función(columna))
```

- No vamos a explorar esta función aquí, ya que tendremos una clase solo sobre estadística descriptiva en **R**.

```
datos_xlsx %>%  
  summarize(media = mean(temperatura), na.rm = TRUE)
```

```
## # A tibble: 1 x 2  
##   media na.rm  
##   <dbl> <lgl>  
## 1  26.5 TRUE
```

# dplyr



## relocate()

- Para reubicar columnas.
- De forma predeterminada, coloca una o más columnas al comienzo de la base.

```
# Coloque las columnas 5 y 4 al principio de la tabla.  
datos %>%  
  relocate(columna5, columna4)
```

- Podemos usar los argumentos `.after =` y `.before =` para realizar cambios más complejos.

```
# Poner la columna 2 después de la columna 4  
datos %>%  
  relocate(columna2, .after = columna4)
```

```
# Poner la columna 2 antes de la columna 4  
datos %>%  
  relocate(columna2, .before = columna4)
```

# dplyr



## rename()\*

- Cambia los nombres de variables individuales (columnas) usando la sintaxis `nuevo_nombre = viejo_nombre`.

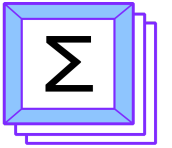
```
datos %>%  
  rename(columna_x = columna.x)
```

- Por ejemplo, vamos cambiar el nombre de la columna "localidad" por "municipalidad".

```
datos_xlsx %>%  
  rename(municipalidad = localidad) %>%  
  names()
```

```
## [1] "ID" "municipalidad" "ano" "zona" "var_respuesta" "temperatura"
```

# dplyr



@somaquadrados

## left\_join(), right\_join() y full\_join()

- Lo usamos para unir dos tablas en una.

```
# Une la tabla 'datos' a la tabla 'datos2' por 'columna_x'  
  
# Mantiene los elementos de la tabla 'datos' y excluye elementos adicionales de 'datos2'.  
datos %>%  
  left_join(datos2,  
            by = "columna_x")  
  
# Mantiene los elementos de la tabla 'datos2' y excluye elementos adicionales de 'datos'.  
datos %>%  
  right_join(datos2,  
             by = "columna_x")  
  
# Mantiene los valores de las dos tablas.  
datos %>%  
  full_join(datos2, by = "columna_x")
```

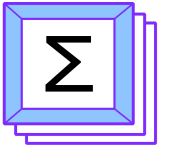


## Ejercicios

Trabajemos con la tabla "tidy\_ej2".

- 1 - Reordene los datos de forma ascendente de acuerdo con la columna "ID".
- 2 - Retire la columna "temperatura".
- 3 - Seleccione solo las líneas con "número de accidentes" > 0.
- 4 - Divida el valor de la columna "número de accidentes" por la columna "población" y multiplique por 100,000; resultando en el número de casos por cada 100.000 habitantes.
- 5 - Salvar la tabla como un ".csv".

¡¡Fin de clase!!







@somaquadrados



¡No olvides tu tarea! 



## Soma dos quadrados

-  [Soma-Dos-Quadrados/introductoryR](#)
-  [/somaquadrados](#)
-  [/somadosquadrados](#)
-  [@somadosquadrados](#)

## Marília Melo Favalesso

-  [mariliabioufpr@gmail.com](mailto:mariliabioufpr@gmail.com)
-  [www.mmfava.com](http://www.mmfava.com)
-  [/mmfava](#)