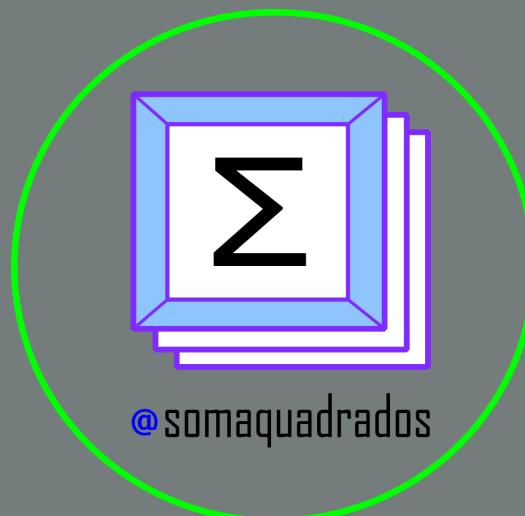
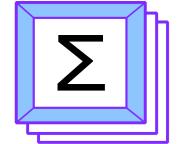


# Programación con R

## Clase 2



Marília Melo Favalesso



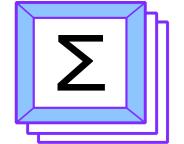
@somaquadrados

# Archivos

- `clase_2.R`
- `datos.csv`
- `datos.txt`
- `datos.xlsx`
- `tidy_ej.xlsx`

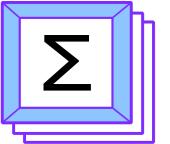
## Contenido de hoy

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



@somaquadrados

# ¿Cómo armar mi tabla de datos?



@somaquadrados

# ¿Cómo armar mi tabla de datos?

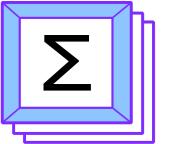
Variables en columnas

célula

unidad experimental u  
observacional en las líneas

The diagram illustrates the concept of variables in columns. It features a person sitting on a large green calculator icon, holding a tablet. A grid table is overlaid on the scene. The table has columns labeled A, B, and C, and rows labeled 1 through 10. An orange arrow points from the word 'célula' to a cell in the B column, row 3. Another orange arrow points from the phrase 'unidad experimental u observacional en las líneas' to the row labels (1-10).

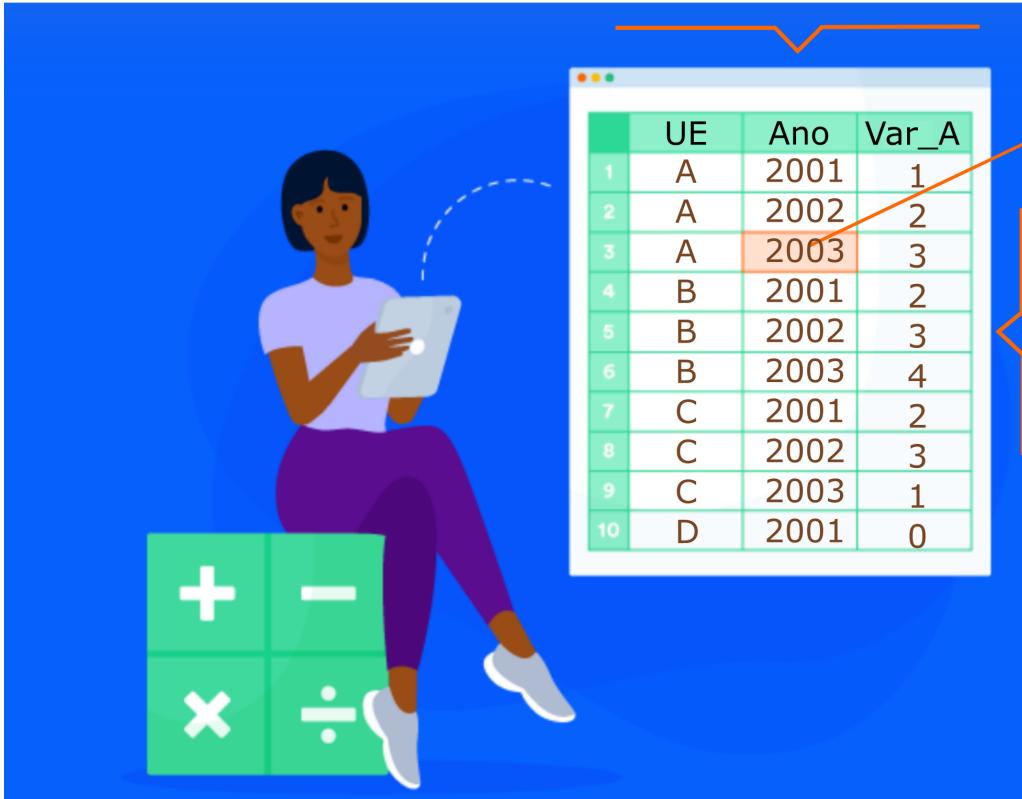
	A	B	C
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			



@somaquadrados

# ¿Cómo armar mi tabla de datos?

Variables en columnas

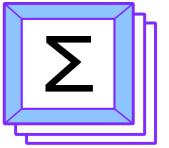


	UE	Ano	Var_A
1	A	2001	1
2	A	2002	2
3	A	2003	3
4	B	2001	2
5	B	2002	3
6	B	2003	4
7	C	2001	2
8	C	2002	3
9	C	2003	1
10	D	2001	0

célula

unidad experimental u  
observacional en las líneas

# ¿Cómo armar mi tabla de datos?

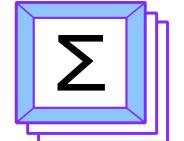


@somaquadrados

## Resumen

Características principales de un conjunto de datos ordenado:

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



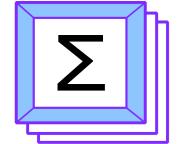
@somaquadrados

# ¿Cómo armar mi tabla de datos?

A screenshot of the Microsoft Excel interface. The ribbon is visible at the top with tabs like File, Design, Home, Insert, Page La, New Tal, Formula, Data, and New Ta. The Home tab is selected. The ribbon has several toolbars: Paste, Font, Alignment, and Number. The status bar at the bottom shows 'Ready' and 'Calculate'. The worksheet area shows a table starting at cell A1, with columns labeled A through E. A red box highlights the tab bar where 'Sheet1' is selected. A green box highlights 'Sheet3', which is the active sheet. A black arrow points from the text 'Descripción de datos:' to the 'Sheet3' tab.

## Descripción de datos:

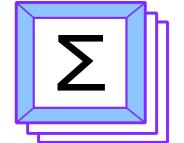
- Utilice las hojas de datos de su editor (ex. excel) para almacenar información sobre su tabla.
- En la primera hoja (hoja 1) dejamos nuestra tabla y en las demás (normalmente la hoja 2, pero si es necesario usamos otras) incluimos información sobre nuestra tabla, como a qué se refieren los datos, descripción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



@somaquadrados

# Tidyverse

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



@somaquadrados

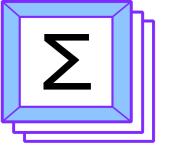
# Tidyverse

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:

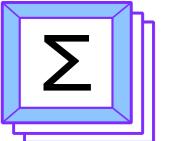
```
install.packages("tidyverse")
```



@somaquadrados

# Tidyverse



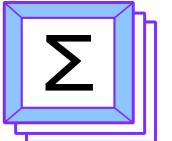


@somaquadrados

# Tidyverse

## Tidyverse es una colección de R

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



@somaquadrados

# Tidyverse

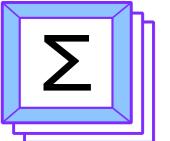
## Tidyverse es una colección de R

- `readr` - importación de datos
- `tibble` - formato de `data.frame` mejorado
- `tidyverse`, `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





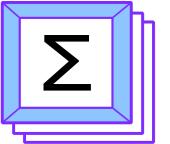
@somaquadrados

# Tidyverse

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



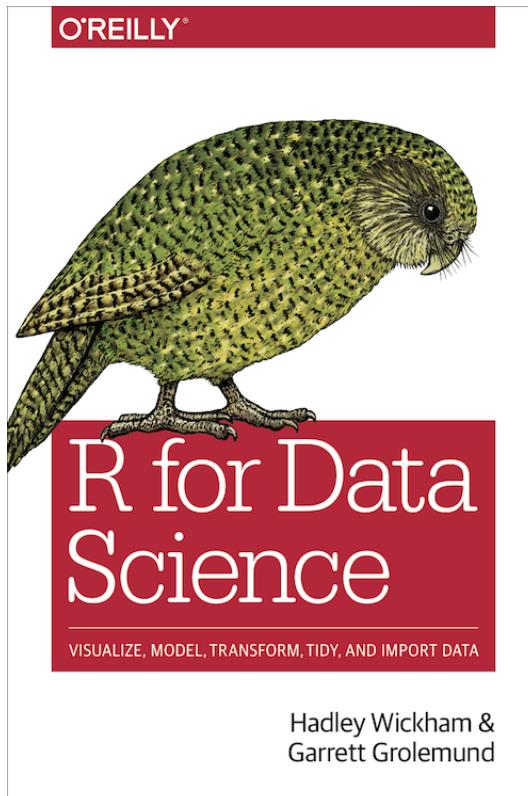
 <http://hadley.nz/>  
 [@hadleywickham](https://twitter.com/hadleywickham)



@somaquadrados

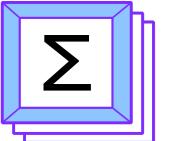
# Tidyverse

## 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/>



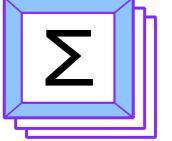
@somaquadrados

# Tidyverse

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

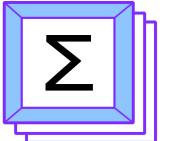
```
install.packages("tidyverse")
```





@somaquadrados

# Importar datos a R



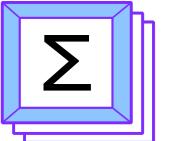
@somaquadrados

# Importar datos a R

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/introductioR/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\introductioR\clase\_2\data
  - : C:/Users/mmfav/introductioR/clase\_2/data

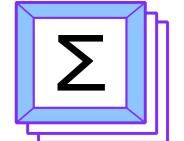
# Importar datos a R



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

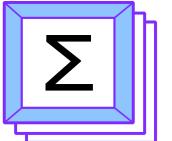
# Importar datos a R



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

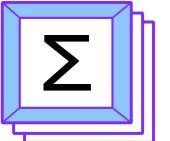
# Importar datos a R

## 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", show_col_types = FALSE)
```

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



@somaquadrados

# Importar datos a R

## readr: .txt

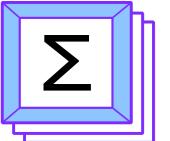
- Como ejemplo, usaremos la base de datos que proporcionamos en el repositorio ([datos.txt](#)).
- La función para leer los datos es: `read_delim(file = "archivo.txt", delim = "\t")`.

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

```
## Rows: 6 Columns: 4

## -- Column specification --
## Delimiter: "\t"
## chr (2): localidad, variable
## dbl (2): ID, value

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```



@somaquadrados

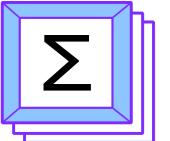
# Importar datos a R

## 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")
```



@somaquadrados

# 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")
```

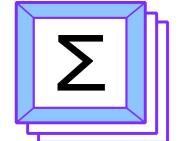
- Cargar el paquete:

```
library(readxl)
```

```
## Warning: package 'readxl' was built under R version 4.0.2
```

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

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



# 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 Puerto Iguazú - Misiones 2001 agrícola      NA         25  
## 2     2 Puerto Libertad - Misiones 2001 agrícola       2         26.7  
## 3     3 San Pedro - Misiones    2001 agrícola       5         24.2  
## 4     4 Tareiri - Misiones     2001 agrícola       4         26.2  
## 5     5 Puerto Iguazú - Misiones 2005 agrícola       6         27  
## 6     6 Puerto Libertad - Misiones 2005 agrícola      NA         25.5  
## 7     7 San Pedro - Misiones    2005 agrícola       7         26  
## 8     8 Tareiri - Misiones     2005 agrícola       9         26.5  
## 9     9 Puerto Iguazú - Misiones 2001 bosque        30         20  
## 10   10 Puerto Libertad - Misiones 2001 bosque        52         21  
## # ... with 14 more rows
```

# Importar datos a R

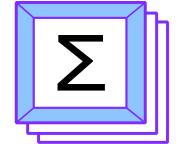


Un **tibble**, o **tbl\_df**, es una reinención moderna del **data.frame**, manteniendo el tiempo que ha demostrado ser efectivo y descartando lo que no lo es.

- 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:

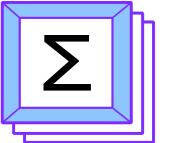
data.frame en **tibble**:

```
as_tibble(data)
```



@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:**

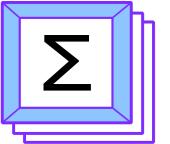
```
# suma el vector y luego obtén la √ (sqrt):  
x <- 1:10
```

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

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

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

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



@somaquadrados

# Operador pipe (%>%)



"Más intuitivo hacerlo un poco"

- **SIN** el pipe:

```
piensas(  
  que(  
    orden(  
      la(  
        en(  
          codar()  
        )  
      )  
    )  
  )  
)
```

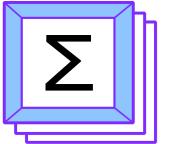
- **CON** el pipe:

```
codar() %>%  
  en() %>%  
  la() %>%  
  orden() %>%  
  que() %>%  
  piensas()
```

---

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

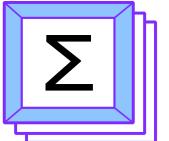
---



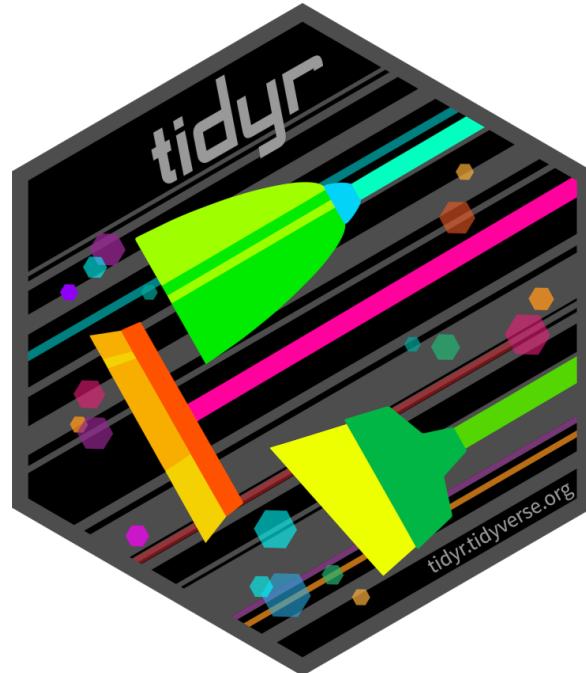
@somaquadrados

# tidy়r

# tidyr



@somaquadrados



El objetivo del  **tidyr** 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.

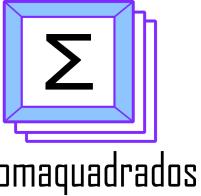
---

Tidyr 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.

---

# tidyverse



- 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>

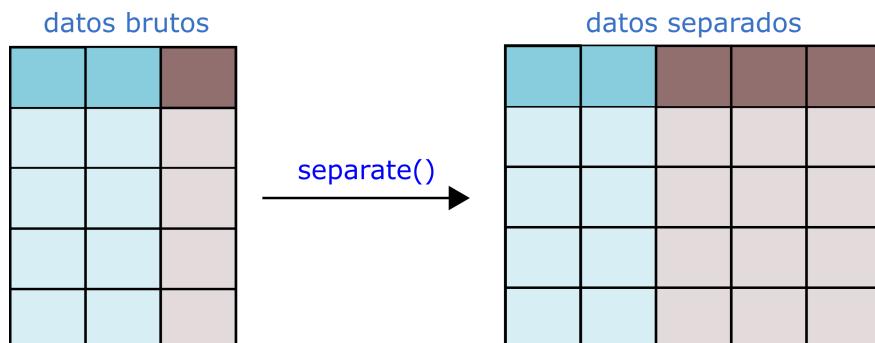
---

# tidyverse



## separate()

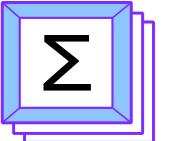
- Muchas veces, una sola variable de columna capturará múltiples variables, o incluso partes de una variable que simplemente no le importa.



- 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
  )
```



# tidyverse

## 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
  ) %>% head()
```

### Antes

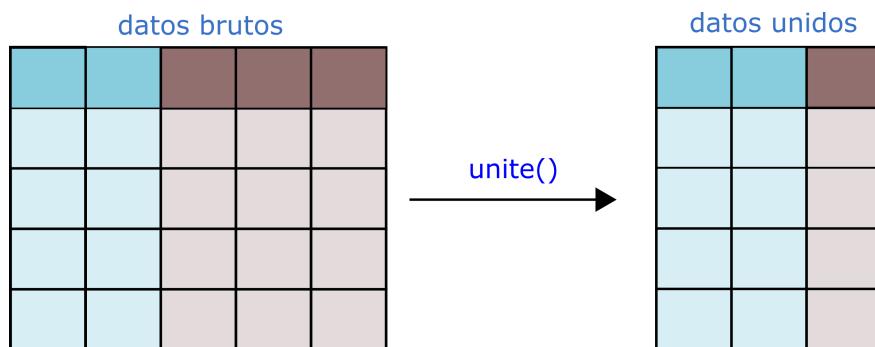
ID	localidad	ano	zona	var_respuesta	temperatura
1	Puerto Iguazú - Misiones	2001	agrícola	NA	25.0

### Después

ID	ciudad	provincia	ano	zona	var_respuesta	tem
1	Puerto Iguazú	Misiones	2001	agrícola	NA	
2	Puerto Liberdad	Misiones	2001	agrícola	2	

## 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
  )
```

# tidyverse



## unite()

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

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

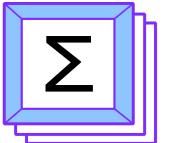
### Antes

ID	localidad	ano	zona	var_respuesta	temperatura
1	Puerto Iguazú - Misiones	2001	agrícola	NA	25.0

### Después

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

# tidyverse



@somaquadrados

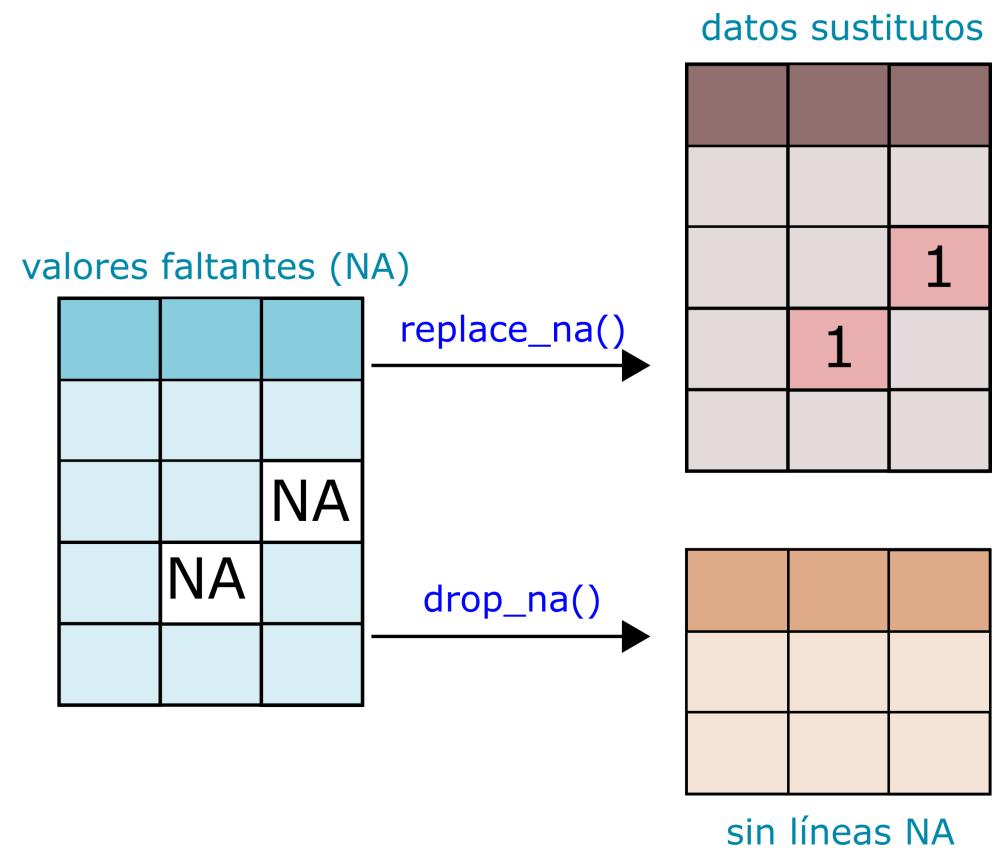
## replace\_na() y drop\_na()

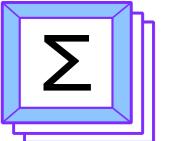
Cuando tenemos datos faltantes en nuestra tabla (NA), podemos reemplazar NA con nuevos valores con la función `replace_na()`...

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

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

```
drop_na(  
  columna  
)
```





@somaquadrados

# tidy়

## replace\_na()

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

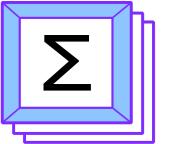
```
datos_xlsx %>%  
  replace_na(list(var_respuesta = 0))
```

### Antes

ID	localidad	ano	zona	var_respuesta	temperatura
1	Puerto Iguazú - Misiones	2001	agrícola	NA	25.0
2	Puerto Libertad - Misiones	2001	agrícola	2	26.7

### Después

ID	localidad	ano	zona	var_respuesta	temperatura
1	Puerto Iguazú - Misiones	2001	agrícola	0	25.0
2	Puerto Libertad - Misiones	2001	agrícola	2	26.7



@somaquadrados

# tidyverse

## drop\_na()

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

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

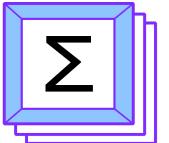
### Antes

ID	localidad	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	2001	agrícola	5	24.2

### Después

ID	localidad	ano	zona	var_respuesta	temperatura
2	<b>Puerto Libertad - Misiones</b>	2001	agrícola	2	26.7
3	San Pedro - Misiones	2001	agrícola	5	24.2

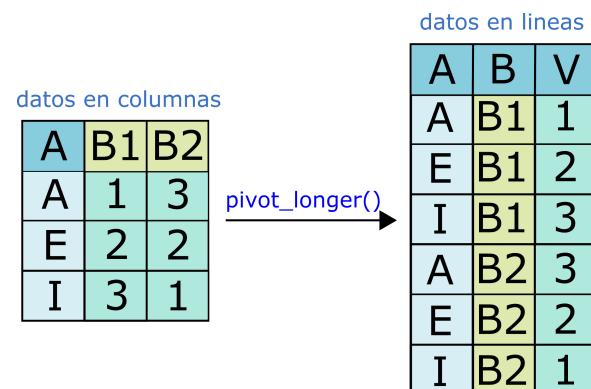
# tidyverse



@somaquadrados

## pivot\_longer()

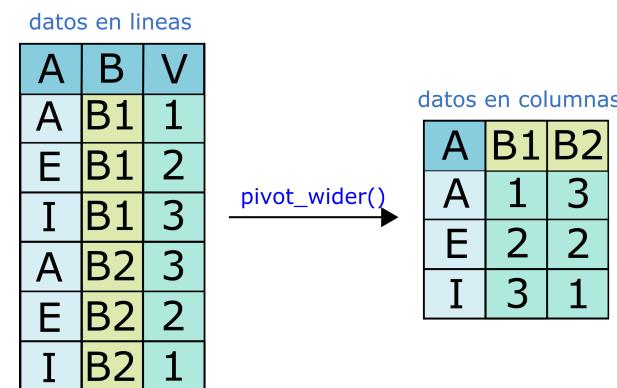
- "Alarga" los datos, aumentando el número de filas y disminuyendo el número de columnas.



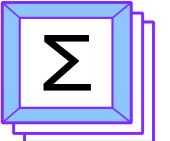
```
pivot_longer(  
  cols = Columnas_para_pivatar,  
  names_to = "nombre_nova_columna",  
  values_to = "nombre_col_values"  
)
```

## pivot\_wider()

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



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



@somaquadrados

# tidyverse

## pivot\_longer()

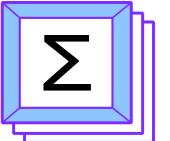
Por ejemplo, pasemos los años de la tabla "datos\_csv" de las columnas a las filas. Los valores los almacenaremos en una columna llamada 'value'.

datos\_csv

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

```
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



@somaquadrados

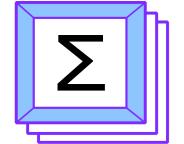
# tidyverse

## pivot\_wider()

¿Qué pasa si quiero revertir la tabla a su formato original?

```
datos_csv2 %>%  
  pivot_wider(  
    names_from = año,  
    values_from = value  
)
```

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



@somaquadrados

# dplyr

# dplyr



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



## 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)
```

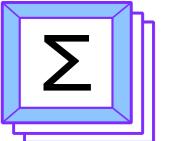
- También disponemos de otras funciones auxiliares:
  - `starts_with()`: para columnas que comienzan con texto estándar
  - `ends_with()`: para columnas que terminan con texto estándar
  - `contains()`: para columnas que contienen texto estándar

## select()

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

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

localidad	año
Puerto Iguazú - Misiones	2001
Puerto Liberdad - Misiones	2001
San Pedro - Misiones	2001
Tareiri - Misiones	2001
Puerto Iguazú - Misiones	2005
Puerto Liberdad - Misiones	2005
San Pedro - Misiones	2005
Tareiri - Misiones	2005



# dplyr

## select()

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

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

localidad	ano	zona	var_respuesta
Puerto Iguazú - Misiones	2001	agrícola	NA
Puerto Liberdad - Misiones	2001	agrícola	2
San Pedro - Misiones	2001	agrícola	5
Tareiri - Misiones	2001	agrícola	4
Puerto Iguazú - Misiones	2005	agrícola	6
Puerto Liberdad - Misiones	2005	agrícola	NA
San Pedro - Misiones	2005	agrícola	7
Tareiri - Misiones	2005	agrícola	9

# 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".

### Antes

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

#### ano var\_respuesta

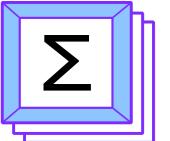
2001	NA
2001	2
2001	5
2001	4
2005	6
2005	NA
2005	7

### Después

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

#### ano var\_respuesta

2001	63
2001	52
2001	41
2001	30
2001	20
2001	16



@somaquadrados

# dplyr

## filter()

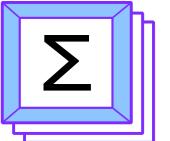
- Para filtrar valores de una columna base, usamos la función `filter()`.

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

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

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

ID	localidad	ano	zona	var_respuesta	temperatura
10	Puerto Libertad - Misiones	2001	bosque	52	21
11	San Pedro - Misiones	2001	bosque	63	22
15	San Pedro - Misiones	2005	bosque	56	24
16	Tareiri - Misiones	2005	bosque	64	26



@somaquadrados

# dplyr

## filter()

- También podemos usar el filtro con caracteres.

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

ID	localidad	ano	zona	var_respuesta	temperatura
17	Puerto Iguazú - Misiones	2001	urbano	20	28.0
18	Puerto Liberdad - Misiones	2001	urbano	14	29.4
19	San Pedro - Misiones	2001	urbano	16	32.4
20	Tareiri - Misiones	2001	urbano	NA	30.0
21	Puerto Iguazú - Misiones	2005	urbano	29	29.0
22	Puerto Liberdad - Misiones	2005	urbano	30	32.5
23	San Pedro - Misiones	2005	urbano	16	33.0
24	Tareiri - Misiones	2005	urbano	18	33.6

# dplyr



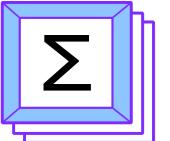
## mutate()

- 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)
```



@somaquadrados

# dplyr

## 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)
```

ID	localidad	ano	zona	var_respuesta	temperatura
1	Puerto Iguazú - Misiones	2001	agrícola	NA	25.0
2	Puerto Liberdad - Misiones	2001	agrícola	1	26.7
3	San Pedro - Misiones	2001	agrícola	4	24.2
4	Tareiri - Misiones	2001	agrícola	3	26.2
5	Puerto Iguazú - Misiones	2005	agrícola	5	27.0
6	Puerto Liberdad - Misiones	2005	agrícola	NA	25.5
7	San Pedro - Misiones	2005	agrícola	6	26.0

# dplyr



## summarize()

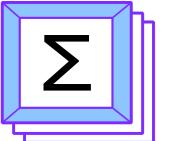
- Es la técnica de resumir un conjunto de datos utilizando alguna métrica de interés.
- Media, mediana, varianza, frecuencia, proporción, por ejemplo, son tipos de resumen que aportan información diferente sobre una variable.

```
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
```



@somaquadrados

# 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)
```

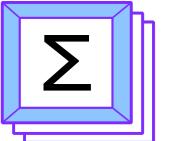
- Por ejemplo, vamos cambiar el nombre de la columna "localidad" por "municipalidad".

### Antes

```
datos_xlsx %>%  
  names()  
  
## [1] "ID"           "localidad"     "ano"
```

### Después

```
datos_xlsx %>%  
  rename(municipalidad = localidad) %>%  
  names()  
  
"zona"           "var_respuesta" "temperatura"  
## [1] "ID"           "municipalidad" "ano"
```



@somaquadrados

# dplyr

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

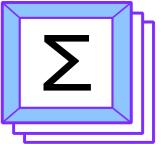
- Lo usamos para unir dos tablas en una.

```
# 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")

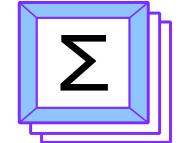
# Mantiene los valores de las dos tablas.
datos %>%
  full_join(datos2, by = "columna_x")
```



@somaquadrados

# ¡¡Fin de clase!!





@somaquadrados

# Soma dos quadrados

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

# Marília Melo Favalesso

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