

COMPUTACIÓN ESTADÍSTICA

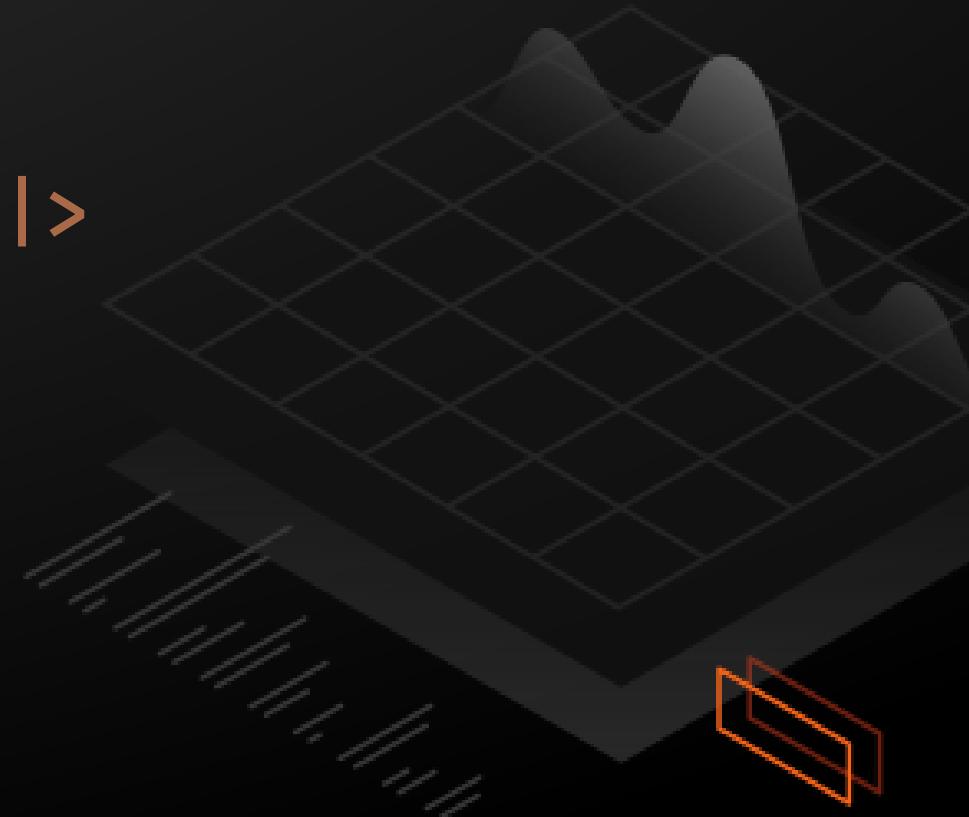
EPG3308

04 TRANSFORMACIÓN DE DATOS **dplyr**, |>

JOSHUA KUNST

@JBKUNST

2023-04-11



PROGRAMAR (Y EL *PIPE*)

PROGRAMAR

- Programar produce código y el código es una herramienta de comunicación
- El código le dice a la computadora qué es lo que quieras que haga, pero también comunica significado a otros seres humanos
- Es importante pensar el código como un medio de comunicación, ya que todo proyecto que realices es esencialmente colaborativo
- Aun cuando no estés trabajando con otras personas, definitivamente lo estarás haciendo con tu *futuro yo*.

FORMAS DE PROGRAMAR

Revisaremos formas (paradigmas) comunes de escribir código revisando sus beneficios/desventajas:

1. Guardar cada paso intermedio como un nuevo objeto.
2. Sobreescribir el objeto original muchas veces.
3. Componer funciones.
4. Usar pipes.

PIPES

```
x                                ## [1] 2 NA 4 6 7 8 NA 10 15 2
x |>
  log() |>
  mean(na.rm = TRUE) |>
  exp() |>
  sqrt()
```

CUAL ES LA MEJOR FORMA?

1. **Guardar paso intermedio**: Se crean objetos irrelevantes con nombres poco importantes. Cuidado al incrementar sufijo en cada linea.
2. **Sobreescibir el objeto original**: Depuración (*debuego*) doloroso, si hay un error hay que correr todo desde el comienzo. La repetición `xnew` hace el código poco transparente.
3. **Componer funciones**: Lectura poco amigable, no tan sencilla de entender. Argumentos terminan separados. *Debugueo* doloroso. Si existe un error no es sencillo reconocer cual de las funciones ocurrió.
4. **Pipes**: Cuidado al *concatenar* varios pasos (> 8?). No sirve en caso de tener muchos inputs/outputs, básicamente trabaja un *objeto*. Son lineales, por lo que existe mucho if/else/depependencia compleja más vale separar expresiones.

EL PIPE (*PA/P*) | >

De forma general veremos que `z |> f()` es equivalente a `f(z)` y en el caso de aplicar parámetros extras tenemos que `z |> g(y)` es `g(x, y)`.

Diferencias entre |> y %>%

En el 99% de los códigos estos *pipes* son intercambiables, i.e., hacen lo mismo.

%>%:

- Primero en aparecer a través del paquete `magrittr` (`dplyr`, lo carga).
- Tiene funcionalidades extras, como otros operadores en el paquete `magrittr`.

| >

- Viene incorporado en R base desde 4.1 (2021-05).

TRANSFORMACIÓN DE DATOS

TRANSFORMACIÓN DE DATOS

La visualización es una herramienta importante para la generación de conocimiento; sin embargo, es raro que obtengas los datos exactamente en la forma en que los necesitas. A menudo tendrás que crear algunas variables nuevas o resúmenes, o tal vez solo quieras cambiar el nombre de las variables o reordenar las observaciones para facilitar el trabajo con los datos.

R4DS.

dplyr Lo BÁSICO

Revisaremos 5 funciones claves de **dplyr** que permiten resolver una gran parte de tus desafíos de manipulación de datos:

- **Filtrar** o elegir las observaciones por sus valores (**filter()** – del inglés filtrar).
- **Reordenar** las filas (**arrange()** – del inglés organizar).
- **Seleccionar** las variables por sus nombres (**select()** – del inglés seleccionar).
- Crear nuevas variables con **transformaciones** de variables existentes (**mutate()** – del inglés mutar o transformar).
- Contraer muchos valores en un solo **resumen** (**summarise()** – del inglés resumir).

Todas estas funciones se pueden usar junto con **group_by()** (del inglés *agrupar por*), que cambia el alcance de cada función para que actúe ya no sobre todo el conjunto de datos sino de grupo en grupo.

dplyr Lo BÁSICO (2)

Todos los verbos funcionan de manera similar:

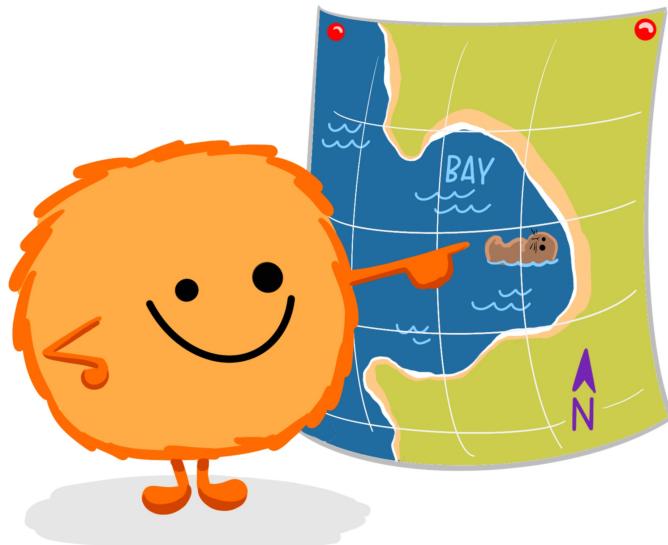
1. El **primer** argumento es un *data frame*.
2. Los argumentos posteriores describen qué hacer con el *data frame* usando los nombres de las variables (sin comillas).
3. El **resultado** es un nuevo *data frame*.

En conjunto, estas propiedades hacen que sea fácil encadenar varios pasos simples para lograr un resultado complejo.

dplyr::filter()

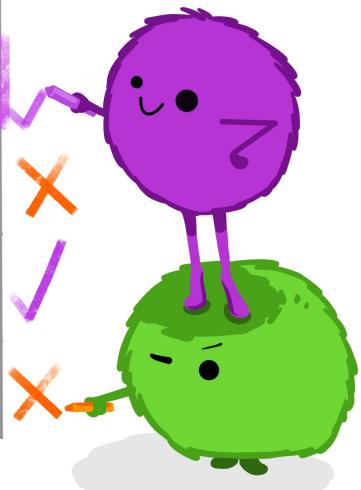
KEEP ROWS THAT
satisfy
your CONDITIONS

keep rows from... this data... ONLY IF... type is "otter" AND site is "bay"
filter(df, type == "otter" & site == "bay")

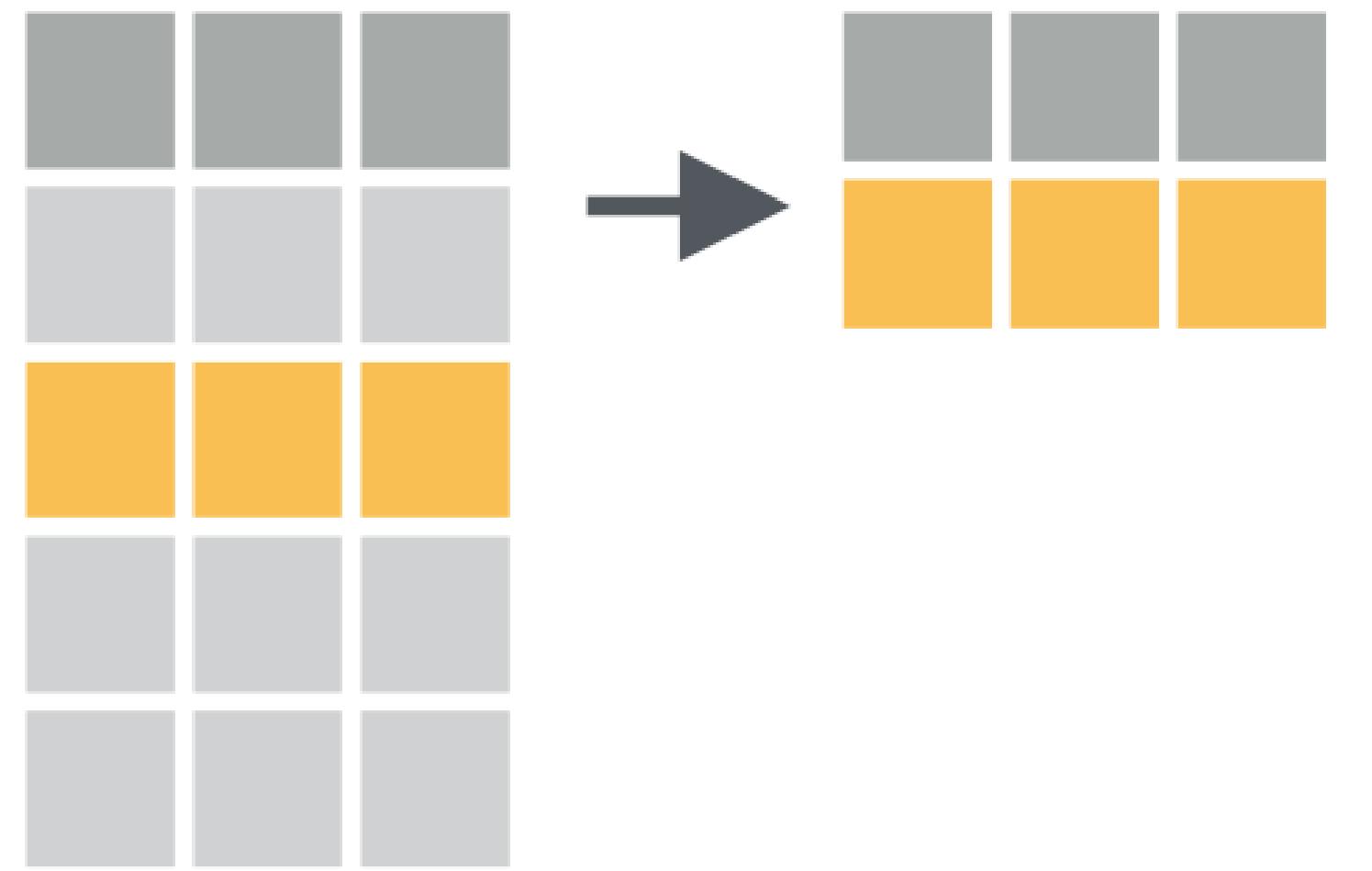


type	food	site
otter	urchin	bay
shark	seal	channel
otter	abalone	bay
otter	crab	wharf

@allison_horst



filter SELECCIONAR FILAS



filter CÓDIGO

```
storms |>
  filter(storm %in% c("Alberto", "Ana"))

# alternativa
storms |>
  filter(storm == "Alberto" | storm == "Ana")
```

```
## # A tibble: 2 × 4
##   storm    wind pressure date
##   <chr>    <dbl>     <dbl> <date>
## 1 Alberto    110      1007 2000-08-03
## 2 Ana        40       1013 1997-06-30

## # A tibble: 2 × 4
##   storm    wind pressure date
##   <chr>    <dbl>     <dbl> <date>
## 1 Alberto    110      1007 2000-08-03
## 2 Ana        40       1013 1997-06-30
```

select SELECCIONAR COLUMNAS



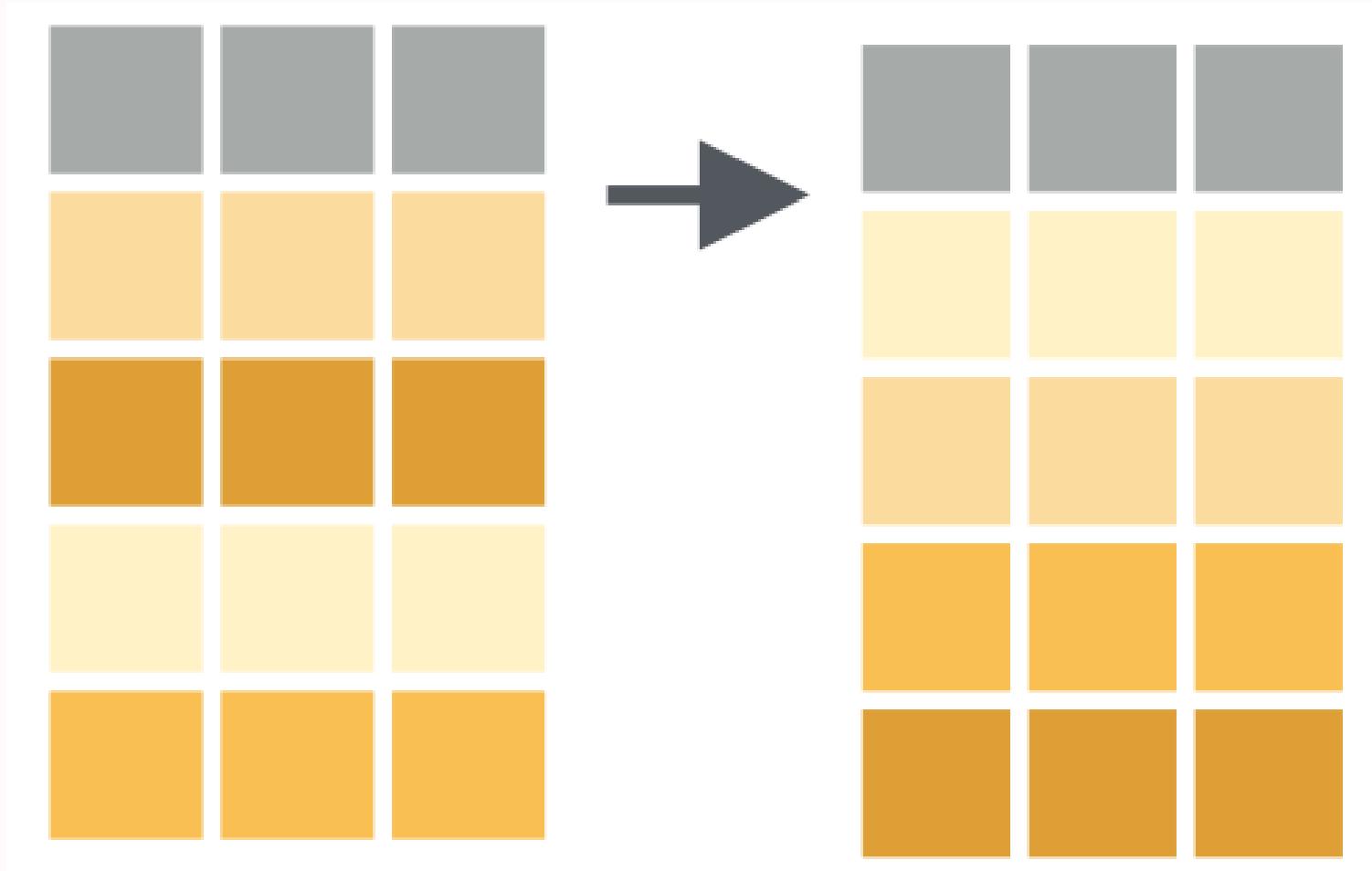
select CÓDIGO

```
storms |>
  select(storm, pressure)

# o alternativa, con índices,
# no tan recomendable
storms |>
  select(1, 3)                                     ## # A tibble: 6 × 2
                                                ##   storm    pressure
                                                ##   <chr>     <dbl>
                                                ## 1 Alberto    1007
                                                ## 2 Alex       1009
                                                ## 3 Allison   1005
                                                ## 4 Ana        1013
                                                ## 5 Arlene    1010
                                                ## 6 Arthur    1010

                                                ## # A tibble: 6 × 2
                                                ##   storm    pressure
                                                ##   <chr>     <dbl>
                                                ## 1 Alberto    1007
                                                ## 2 Alex       1009
                                                ## 3 Allison   1005
                                                ## 4 Ana        1013
                                                ## 5 Arlene    1010
                                                ## 6 Arthur    1010
```

arrange ORDENAR FILAS



arrange CÓDIGO

```
storms |>
  arrange(wind)

# multiple columnas con
# orden inverso
storms |>
  arrange(wind, desc(pressure))
```

```
## # A tibble: 6 × 4
##   storm    wind pressure date
##   <chr>    <dbl>     <dbl> <date>
## 1 Ana        40     1013 1997-06-30
## 2 Alex       45     1009 1998-07-27
## 3 Arthur     45     1010 1996-06-17
## 4 Arlene     50     1010 1999-06-11
## 5 Allison    65     1005 1995-06-03
## 6 Alberto    110    1007 2000-08-03
```

```
## # A tibble: 6 × 4
##   storm    wind pressure date
##   <chr>    <dbl>     <dbl> <date>
## 1 Ana        40     1013 1997-06-30
## 2 Arthur     45     1010 1996-06-17
## 3 Alex       45     1009 1998-07-27
## 4 Arlene     50     1010 1999-06-11
## 5 Allison    65     1005 1995-06-03
## 6 Alberto    110    1007 2000-08-03
```

`dplyr::mutate`
add column(s),
keep existing.



Horst '18

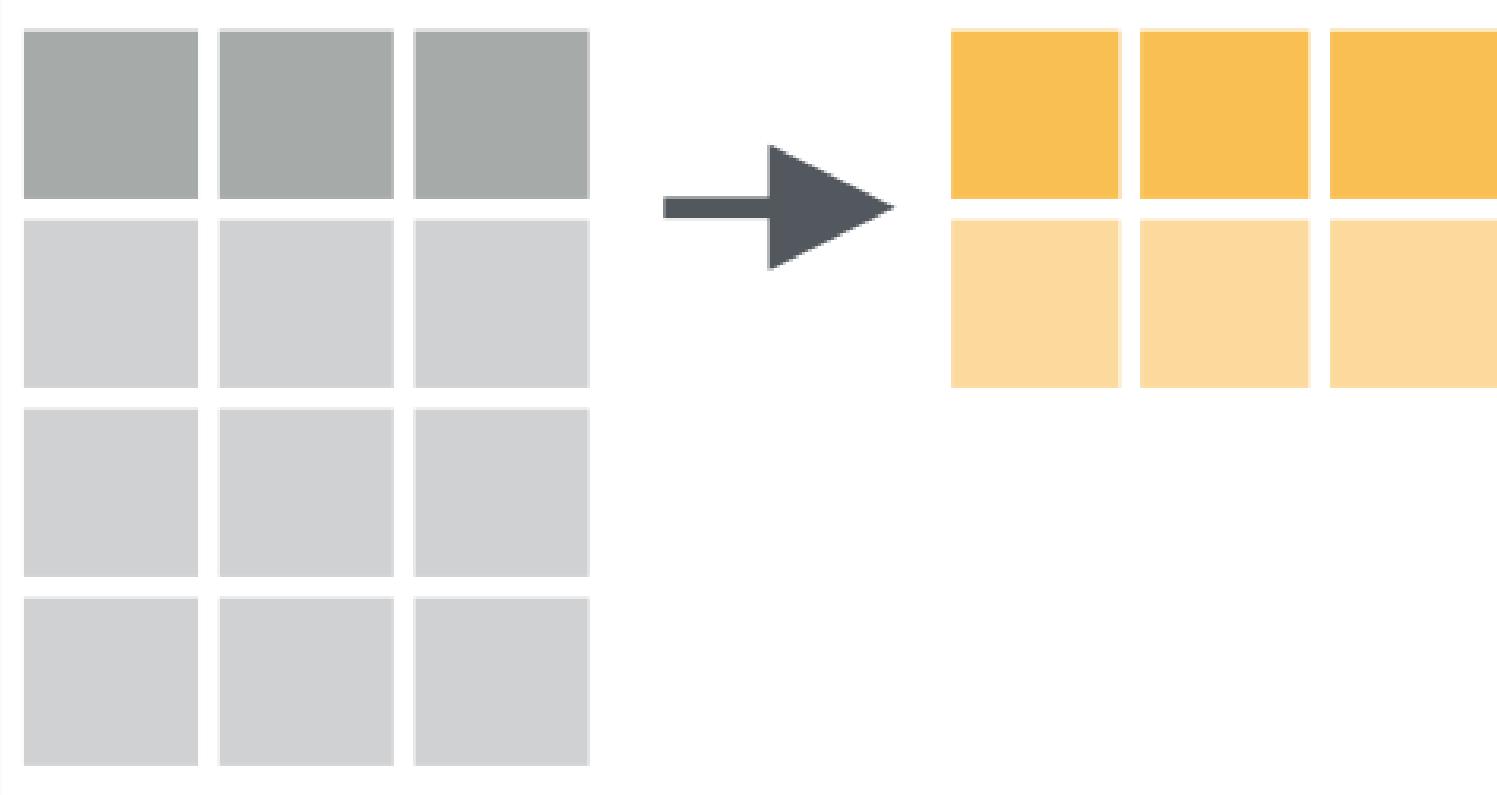
mutate CREAR (O TRANSFORMAR) COLUMNAS



mutate CÓDIGO (DATO)

```
storms |>
  mutate(
    wind_promedio = mean(wind)
  ) |>
  filter(wind > wind_promedio) ## # A tibble: 2 × 5
##   storm     wind pressure date      wind_promedio
##   <chr>    <dbl>    <dbl> <date>        <dbl>
## 1 Alberto    110     1007 2000-08-03      59.2
## 2 Allison    65      1005 1995-06-03      59.2
```

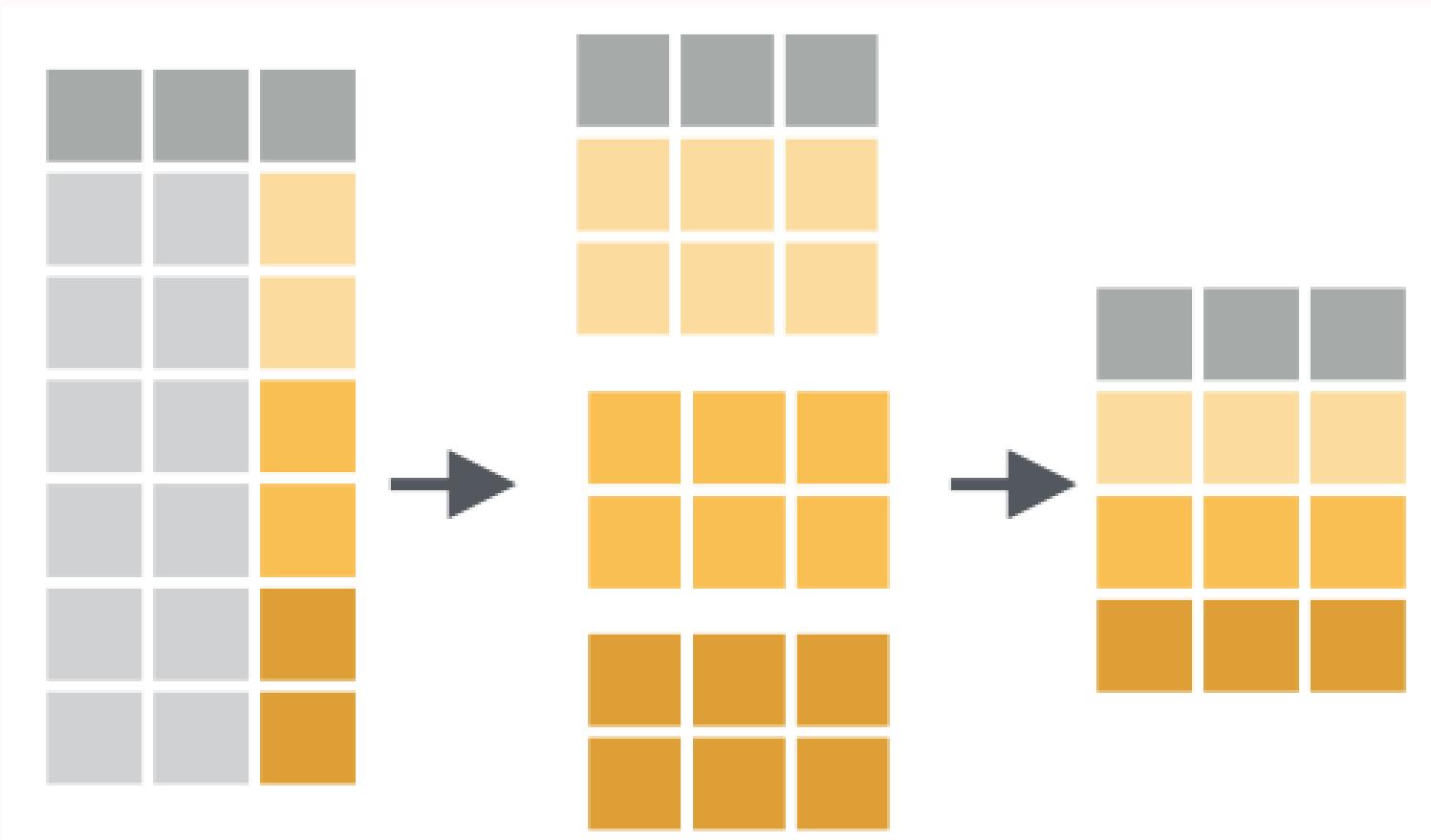
summarise RESUMIR COLUMNAS



summarise CÓDIGO

```
pollution |>  
  summarise(median = median(amount))  
  
## # A tibble: 1 × 1  
##   median  
##   <dbl>  
## 1     22.5
```

group_by |> summarise RESUMIR COLUMNAS POR GRUPOS



group_by |> mutate CÓDIGO (SPANISH VERSION)

```
pollution |>
# no es necesario pero sirve para viz
  arrange(size) |>
  group_by(size) |>
  mutate(promedio_amout = mean(amount)) |>
  filter(amount > promedio_amout) |>
  summarise(promedio_amout = mean(amount)) |>
  mutate(promedio_amout = round(promedio_amout, 2)) |>
  select(-amount) |>
  print()
```

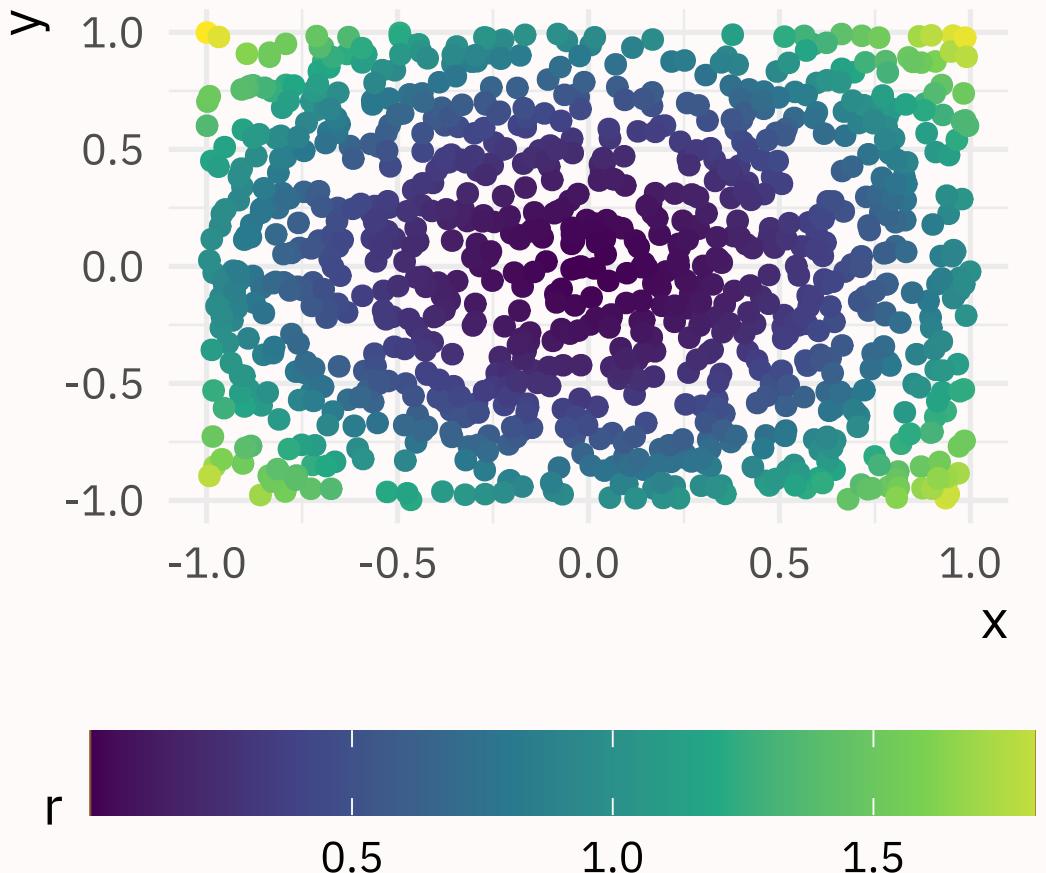
city	size	amount	promedio_amout
Beijing	large	121	55.3
Beijing	small	56	28.7

EXPLORATORIO PREVIO

```
library(tidyverse)
set.seed(123)

df <- tibble(
  x = runif(1000, -1, 1),
  y = runif(1000, -1, 1)
) |>
  mutate(
    r = x^2 + y^2,
    ri = r < 1,
    convergencia = cummean(ri),
    row = row_number()
  )

ggplot(df) +
  geom_point(aes(x, y, color = r)) +
  scale_color_viridis_c()
# ahora cambia r por ri
# y viridis_c por viridis_d
```



TIDY DATA TUTOR

Con *Tidy data tutor* analice el siguiente código.

```
library(dplyr)
library(palmerpenguins)

data(penguins)

penguins %>%
  filter(species == "Gentoo") %>%
  mutate(body_mass_kg = body_mass_g/1000) %>%
  filter(sex == "male" | sex == "female") %>%
  group_by(year, sex) %>%
  summarise(body_mass_kg_mean = mean(body_mass_kg)) %>%
  arrange(desc(body_mass_kg_mean))
```

EJERCICIO EN GRUPO! UN SOLO GRUPO!

1. Cargue **tidyverse** y el paquete **datos**. Explore las tablas **vuelos**, **aeropuertos**, **aerolineas**, **clima** y **aviones**.
2. Filtra los vuelos que han sido cancelados.
3. Filtra los vuelos que salieron del aeropuerto JFK.
4. Encuentra el número de vuelos que llegaron tarde más de 2 horas.
5. Agrupa los vuelos por mes y encuentra el número promedio de pasajeros por mes.
6. Encuentra el día de la semana con la mayor cantidad de vuelos.
7. Encuentra el aeropuerto de destino más popular.

EJERCICIO EN GRUPO! UN SOLO GRUPO! (CONT.)

1. Encuentra el promedio de la duración del vuelo (en minutos) para los vuelos que salieron del aeropuerto JFK y llegaron al aeropuerto LAX.
2. Obtenga el par **origen destino** más común.
3. Encuentra el número promedio de vuelos por día para cada aerolínea.
4. Cuál es el vuelo de mayor duración.
5. Cual es la aerolínea que presenta mayor proporción de atrasos (salida/llegada),
6. Por mes, obtenga el promedio/mediana de atrasos (salida/llegada), y grafique en ggplot2 a modo de explicar los observado.
7. Realice un gráfico de **fecha** por temperatura separando por origen (aeropuerto de salida).

OTRO TIPO DE EJERCICIOS (2)

```
library(tidyverse)
library(datos)

algun_na <- function(x) any(is.na(x))

x <- c(0, NA, 0)
is.na(x)
algun_na(x)

vuelos |>
  select(where(algun_na)) |>
  filter(if_any(everything(), is.na)) |>
  filter(if_all(everything(), is.na)) |>
  glimpse()

## [1] FALSE  TRUE FALSE
## [1] TRUE
## Rows: 2,512
## Columns: 6
## $ horario_salida <int> NA, NA, NA, NA, NA, NA, NA,
## $ atraso_salida   <dbl> NA, NA, NA, NA, NA, NA, NA,
## $ horario_llegada <int> NA, NA, NA, NA, NA, NA, NA,
## $ atraso_llegada  <dbl> NA, NA, NA, NA, NA, NA, NA,
## $ codigo_colar    <chr> NA, NA, NA, NA, NA, NA, NA,
## $ tiempo_vuelo    <dbl> NA, NA, NA, NA, NA, NA, NA,
```

dplyr LO MENOS BÁSICO

dplyr ofrece todo un grupo de funciones/helpers para realizar el análisis más textual (verbose). Se recomienda buscar:

- `contains`, `start_with`
- `across`, `where`
- `if_any`, `if_all`

MÁS INFORMACIÓN SOBRE TRANSFORMACIÓN DE DATOS

- Capítulo [Pipes](#) en R4DS.
- Ejercicio de [Transformación de datos](#) en R4DS.
- [Tidy data tutor](#), visualizar *pipelines* de transformación de datos.
- Buenos artículos de [dplyr](#) en la página de [documentación](#) partiendo por [acá](#).