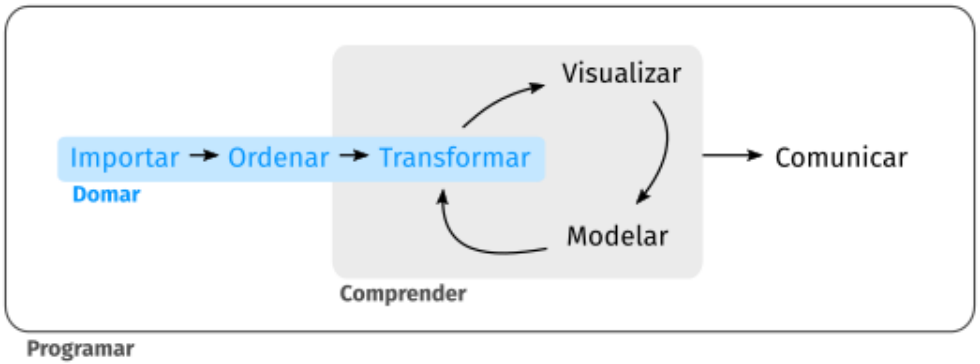
Tema 02 - Tratamiento de datos (una tabla)

Pedro Albarrán

Dpto. de Fundamentos del Análisis Económico. Universidad de Alicante

Contenidos I

Limpieza y "doma" de datos



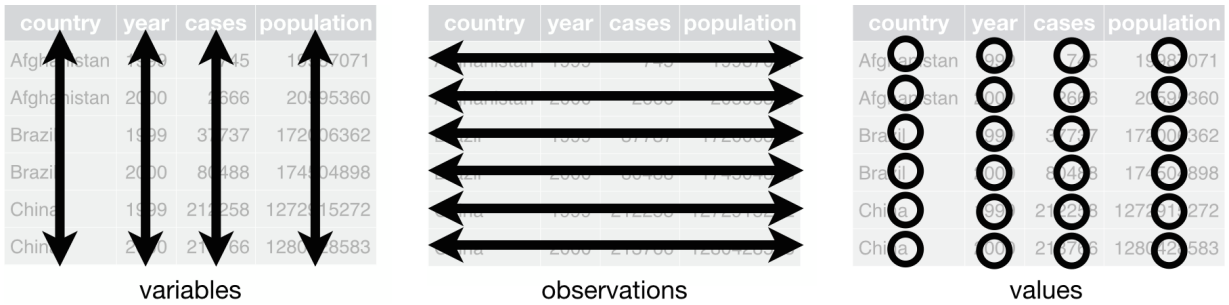
- Un análisis de datos adecuado requiere (mucho) tiempo de trabajo "sucio"
- `tidyverse` incluye una colección de bibliotecas con herramientes eficientes para el proceso de "tratamiento de datos" ("data wrangling")
- El objetivo es tener un conjunto de datos **ordenado** y **limpio** para poder realizar análisis de manera eficiente
- Esto puede requerir seleccionar columnas, filtrar filas, crear nuevas variables, ordenar, agrupar, resumir, etc.

¿Qué son datos ordenados ('tidy data')?

1.- Cada columna es una **variable**: mide el mismo *atributo* entre unidades

2.- Cada fila es una **observación** (caso): misma *unidad* a través de atributos

3.- Cada celda es un **valor**



- Tenemos información similar y no redundante en una misma tabla
- Es una forma natural (variable = vector columna) para trabajar con datos
- `tidyverse` es eficiente con datos ordenados: ej., gráfico temporal

```
ggplot(table1, aes(x = year, y = cases)) +
  geom_line(aes(colour = country))
```

Datos no ordenados

Relaciones de pareja			

Hombres según si tienen hijos o no, situación sentimental y edad
Unidades: Hombres

	Total	Sin hijos	Con hijos
Total			
Total	12.030.071	5.899.310	6.130.761
Sin pareja	3.517.103	3.255.256	261.847
Pareja con la que convive	7.021.533	1.310.179	5.711.354
Pareja con la que no convive	1.491.435	1.333.875	157.560
Menos de 30 años			
Total	2.870.677	2.730.516	140.161
Sin pareja	1.685.274	1.676.571	.
Pareja con la que convive	344.134	217.727	126.407
Pareja con la que no convive	841.269	836.217	.
De 30 a 34 años			
Total	1.387.637	872.102	515.535
Sin pareja	388.933	376.583	.
Pareja con la que convive	779.188	294.321	484.867
Pareja con la que no convive	219.516	201.198	18.318
De 35 a 39 años			
Total	1.748.484	717.629	1.030.855
Sin pareja	366.115	324.690	41.425
Pareja con la que convive	1.213.458	238.283	975.175

Funciones de transformación de datos

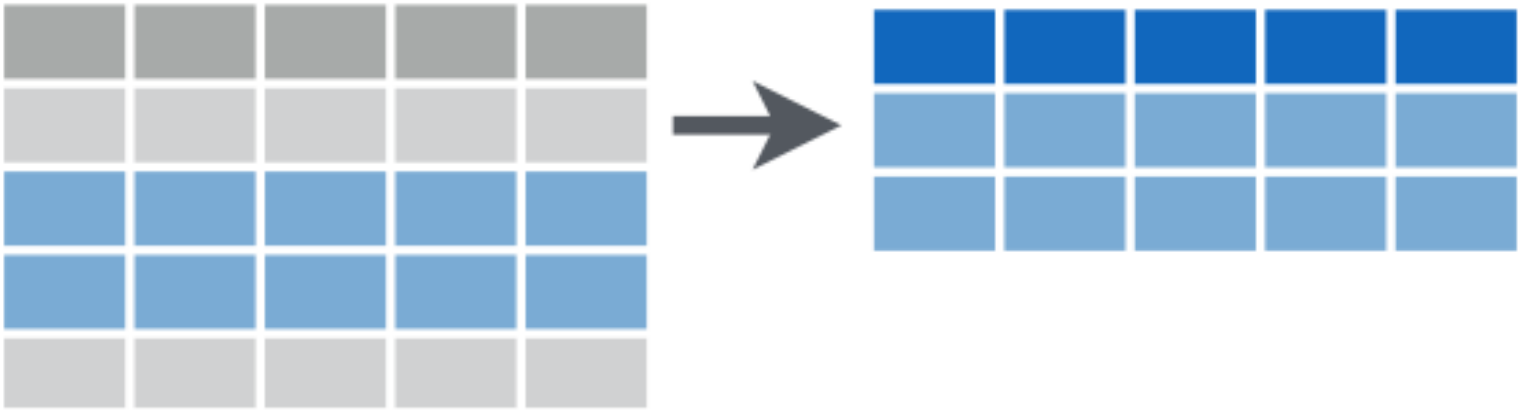
- La mayoría de operaciones pueden realizarse combinando 5 "verbos"
  - NOTA: existe una colección de "chuletas" de R, p.e., para transformación.
- Todos tienen como primer argumento un *data frame*, los siguientes describen qué hacer (con columnas o filas) y devuelven otro *data frame*

1.- `select()`: **selecciona variables** por nombres o posiciones de columnas, separados por comas

```
select(presidential, name, party)
select(presidential, 1:2, 4)
```

Filtrar filas

2.- `filter()`: conserva filas en las que la *condición lógica* es verdadera



```
filter(presidential, party == "Republican")
filter(presidential, start > 1973 & party == "Democratic")
```

- Se pueden combinar (**anidar**) porque ambas toman y devuelve un *data frame*, pero así son difíciles de leer

```
select(filter(presidential, start > 1973, name))
```

El operador de tubería %>%

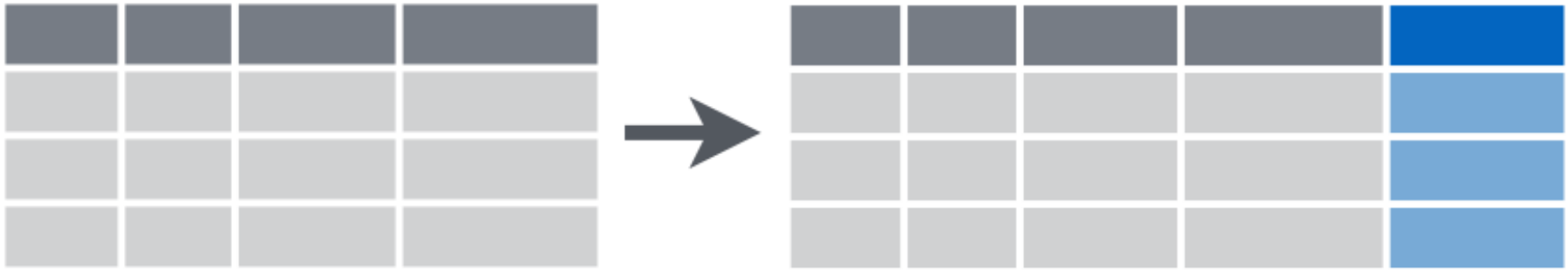
- `datos %>% filter(condition)` equivale a `filter(datos, condition)`
- El anidamiento es fácil: - Tomar `presidential` y pasarlo a filtrar (produce un nuevo *data frame*); - Tomar este resultado y pasarlo a seleccionar.

```
presidential %>%
  filter(start > 1973) %>%
  select(name)
```

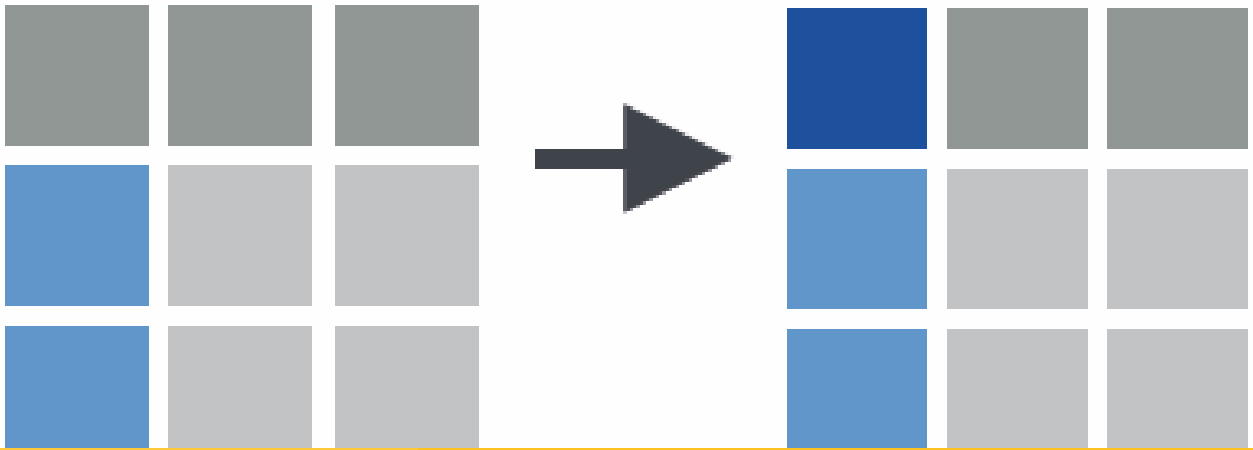
- Atajo de teclado: `Cmd / Ctrl + Mays + M`
- Se puede aplicar a cualquier función: `10 %>% log()` es `log(10)`
- También existe una tubería en R base: `|>`

Crear nuevas variables

3.-`mutate()`: añade nuevas columnas, creando variables según una **fórmula** a partir de otras



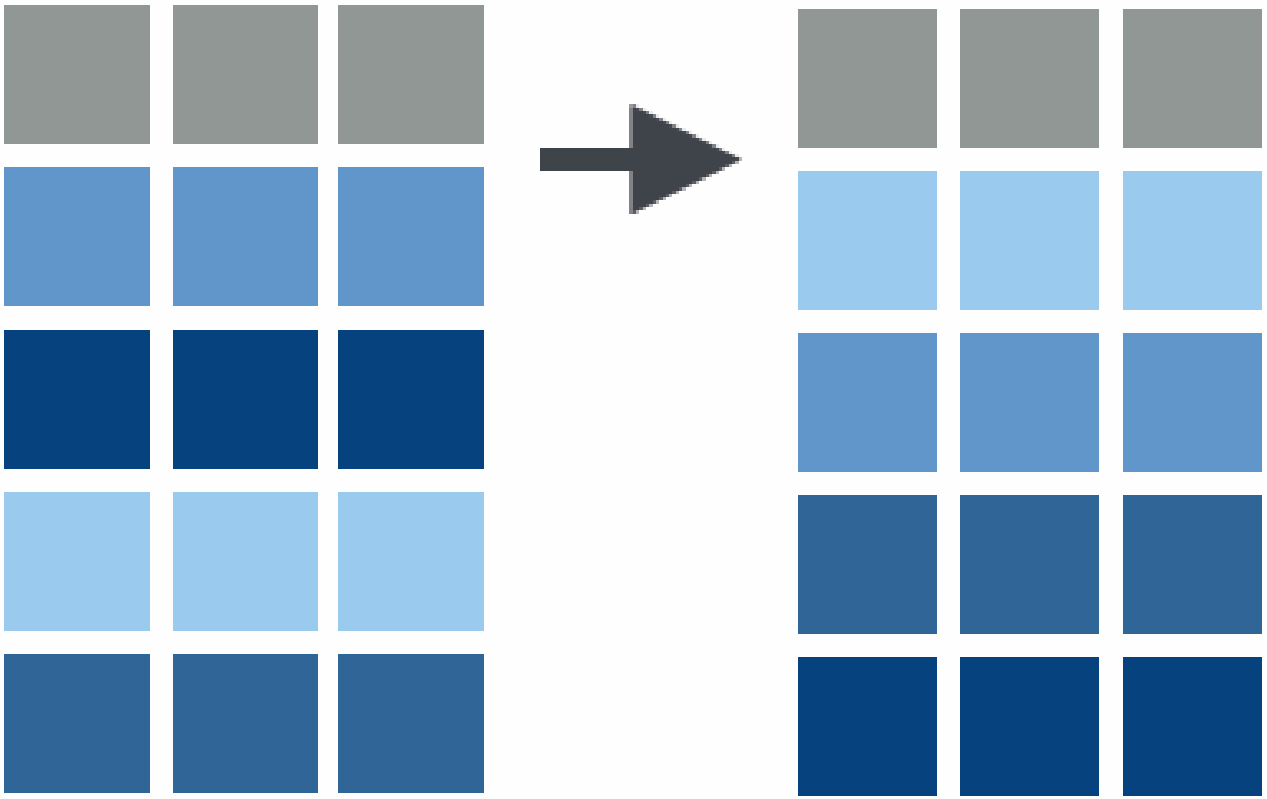
- también `rename()`: cambiar el nombre de una columna



Ordenar filas

4.- `arrange()`: re-ordena las filas todas las columnas de un *data frame*

- en orden ascendente (por defecto) o descendente con `desc()`



Resumir todo el conjunto de datos

5.- `summarize()`: colapsa valores de un *data frame* en una sola fila resumen

- Especificando *cómo* se reducirá una columna entera de datos en un solo valor.

```
library(lubridate)
mypresidents %>%
  summarize(media_duracion = mean(duracion),
            N = n(),                          # n() cuenta número tot
            first_year = min(year(start)),    # year() extrae el año
            num_dems = sum(party == "Democratic") )
```

- `summarize()` suele usarse en conjunción con `group_by()`

group_by()

- group_by(): cambia el alcance de *cada función* para que no actúe sobre todo el *data frame* sino en grupos individuales
- ¿Cuál es la duración media de los demócratas y de los republicanos? Hacerlo por separado no es eficiente: especificamos que las filas deben ser agrupadas

```
mypresidents %>% group_by(party) %>%          # "marca" dos grupos e
  summarize(N = n(),                          # incluye estas variab
            media_duracion = mean(duracion))  # mas las de group_by(
```

- Nuevo *data frame* con distinto nivel de observación (fila): una fila para cada valor de la variable por la que se agrupa (ej., de presidentes a partidos)
- ungroup() elimina la agrupación: volvemos a operar en datos desagrupados

```
mypresidents %>% group_by(party) %>%
  mutate(media_duracion = mean(duracion)) %>%
  ungroup() %>% arrange(duracion) %>%  slice(1)
```

Seleccionar muchas variables

```
library(nycflights13)            # incluye flights:  19 variables
select(flights, year:arr_time)     # desde variable "year" hasta "arr
select(flights, -(year:day))       # todas menos "year, month, day"
```

- Funciones a utilizar dentro de `select()`:
  - `starts_with("abc")`: nombres que comienzan con "abc".
  - `ends_with("xyz")`: nombres que acaban con "xyz".
  - `contains("ijk")`: nombres que contienen "ijk".
  - `num_range("x", 1:3)`: para x1, x2 y x3.
  - `matches()`: nombres que coinciden con una expresión regular

Algunos verbos adicionales

- `slice()`, `slice_sample()`: extrae filas por posición o aleatoriamente

```
mypresidents %>% slice(1, 4)
```

- `drop_na()` y `replace_na()`: elimina filas con valores ausentes o los reemplaza
- `distinct()`: extrae sólo las filas únicas (una o varias variables)

```
mypresidents %>% distinct(party)
```

- `count()`: cuenta los valores únicos de una o más variables

```
mypresidents %>% count(party)     # mypresidents %>% group_by(party)
mypresidents %>% count(party, sort = TRUE)
```

- `across()`: aplica la misma transformación a múltiples columnas

```
flights %>% mutate(across(air_time:distance, ~ log(.x)+1))
flights %>% mutate(across(is.character, ~ parse_factor(.x)))
```

Funciones para crear variables

- Operadores aritméticos (+, -, *, /, ^, %/%, %%) y lógicos (<, <=, >, >=, !=)
- Funciones como `log()`, `lag()`, `lead()`, `cumsum()`, `row_number()` etc.
- `if_else()`: ejecución condicional (también `case_when()`)

```
flights %>% mutate(retraso = if_else(dep_delay > 0, "tarde", "bien")
flights %>% mutate(retraso = if_else(dep_delay > 0, "tarde",    # en
                                     if_else(dep_delay <0, "bien", "norma
```

- Discretizar variables: `cut_interval()`, `cut_number()`, `cut_width()`
- Nota: `dplyr` tiene muchas funciones equivalentes a otras de R base:
  - `parse_number()`, `parse_factor()`, etc. por `as.number()`, `as.factor()`, etc.
  - `bind_cols()` y `bind_rows()` por `cbind()` y `rbind()`

Funciones de resumen útiles

- Medidas de centralidad y de dispersión: `mean(x)`, `median(x)`, `sd(x)`, `IQR(x)`, `mad(x)`
- Medidas de rango: `min(x)`, `quantile(x, 0.25)`, `max(x)`
- Medidas de posición: `first(x)`, `nth(x, 2)`, `last(x)`.
  - similar a `x[1]`, `x[2]` y `x[length(x)]`
- Conteos:
  - `n()`: observaciones totales (tamaño del grupo)
  - `sum(!is.na(x))`: observaciones no ausentes
  - `n_distinct(x)`: filas distintas en `x`

Cuatro representaciones de los mismos datos

```
library(tidyverse)
table1      # datos ordenados
table2      # no tiene un valor por
```

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

country	year	type	count
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272
China	2000	cases	213766
China	2000	population	1280428583

```
table3      # mezcla más de una var
table4a
table4b
```

- `table4a` y `table4b` ofrecen información útil para presentación

country	1999	2000
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

Mismos datos, dos formatos: ancho o largo

- La utilidad de almacenar los datos en un rectángulo ancho ("wide") o en uno largo ("long" ") depende de qué queramos hacer
- El cambio de forma entre formatos es una tarea habitual del analista de datos.
- Cambiar entre representación larga y ancha se conoce como **pivotar (o girar)**

`wide`

id	x	y	z
1	a	c	e
2	b	d	f

`long`

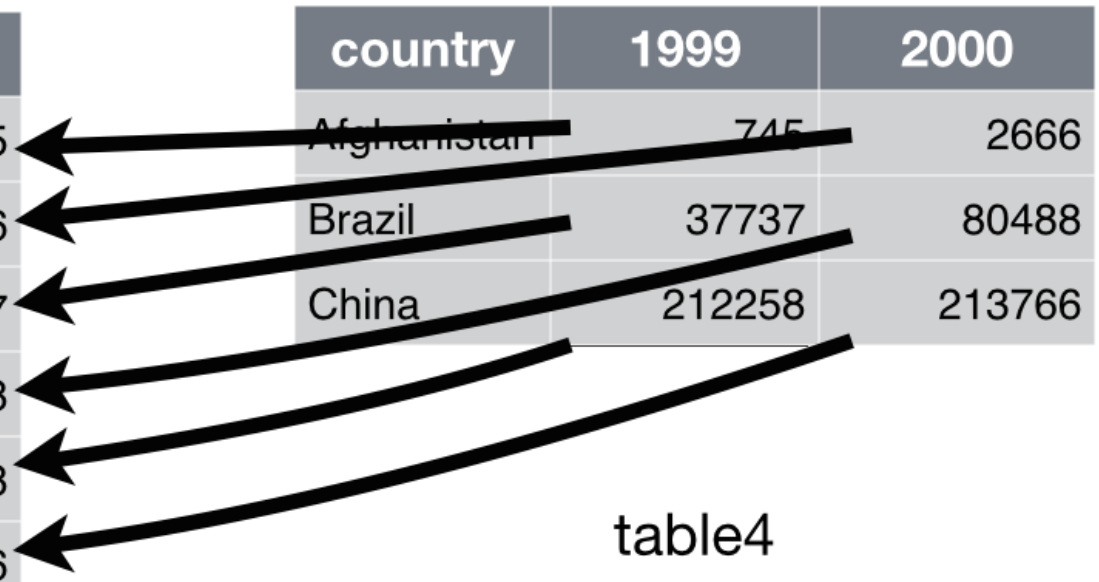
id	key	val
1	x	a
2	x	b
1	y	c

pivot_longer(): de ancho a largo

- Pivotar las variables no ordenadas en dos nuevas columnas (deben crearse)

```
pivot_longer(table4a,
             cols=2:3,
             names_to = "year",
             values_to = "cases")
```

country	year	cases
Afghanistan	1999	745
Afghanistan	2000	2666
Brazil	1999	37737
Brazil	2000	80488
China	1999	212258
China	2000	213766



country	1999	2000
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

table4

1. *data frame* a cambiar de forma

pivot_longer(): de ancho a largo (cont.)

- Recordatorio: formas equivalentes de hacer lo mismo

```
pivot_longer(table4a, 2:3, names_to = "year", values_to = "cases")
table4a %>% pivot_longer(c(`1999`, `2000`), values_to = "cases", nam
table4a %>% pivot_longer(names_to = "year", values_to = "cases", -co
table4a %>% pivot_longer(names_to = "year", values_to = "cases", `19
```

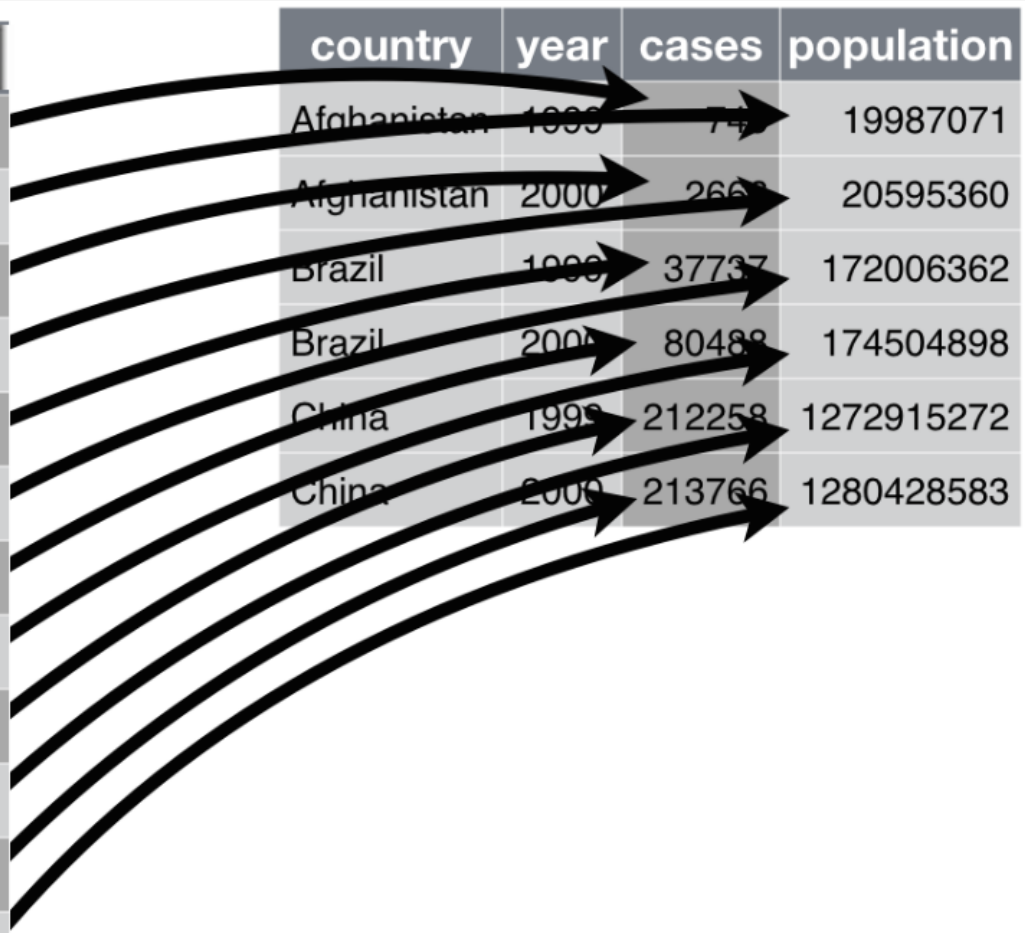
- Notar que los nombres de columna son caracteres y cuando son números van entre ‘ (evita confusión con índice de posición)
- Deberíamos cambiar el tipo de las nuevas variables

```
pivot_longer(table4a, 2:3, names_to = "year", values_to = "cases") %
  mutate(year= parse_number(year))
```

pivot_wider(): de largo a ancho

```
table2 %>%
    pivot_wider(names_from = type,
                values_from = count)
```

country	year	type	count
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272
China	2000	cases	213766



country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

Dos funciones útiles

- `separate()`: dividir una columna en múltiples variables indicando un separador o vector de posiciones en las que dividir

```
table3 %>% separate(rate, into = c("cases", "population"), sep = "/"
table5 <- table3 %>% separate(year, into = c("century", "year"), sep
```

- Con el argumento `convert = TRUE` intenta convertir el tipo de datos (no mantener carácter)

```
table3 %>% separate(rate, into = c("cases", "population"), convert =
```

- `unite()`: combinar múltiples columnas en una

```
table5 %>%
  unite(new, century, year, sep = "-")
```

Comentario sobre valores ausentes

```
accion <- tibble( anio  = c(2015, 2015, 2015, 2015, 2016, 2016, 2016
                  trim  = c(   1,    2,    3,    4,    2,    3,    4
                  rent  = c(1.88, 0.59, 0.35,   NA, 0.92, 0.17, 2.66
```

- **Dos tipos** de valores ausentes: en 2015.Q4 *explícitos* y en 2016.Q1 *implícitos*
- Esto cambia con la forma de representación

```
ancho <- accion %>%
           pivot_wider(names_from = anio, values_from = rent)
ancho             # NA explícitos al convertir a formato ancho

ancho %>%  pivot_longer(cols = c("2015","2016"),
                        names_to = "anio", values_to = "rent")

accion %>% complete(anio, trim)  # todos NA explícitos
                                 # (rellena buscando todas combinaci
```