

# Manipulación de datos en R II

## Programación para el análisis de datos

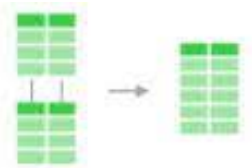
Departamento de Ciencias Sociales, UCU - Martín Opertti

# Unir bases de datos

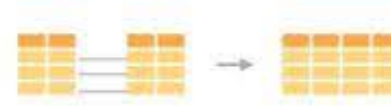
# Unir dataframes

dplyr cuenta con dos funciones para combinar dataframes: `bind_rows()` y `bind_cols()`.

`bind_rows()`



`bind_cols()`



Surles (2017)

# Unir dataframes por columna

Cuando tenemos dos dataframes con las mismas variables podemos usar `bind_rows()`. También se puede utilizar `rbind()` del R Base.

```
nba_data <- read_csv("data/nba_data.csv") %>%  
  janitor::clean_names()
```

```
# Dividamos el dataframe en 2 para volver a unirlo  
nba_data_18 <- filter(nba_data, season == 2018)  
nba_data_17 <- filter(nba_data, season == 2017)  
  
nba_data_17_18 <- bind_rows(nba_data_17, nba_data_18)  
  
table(nba_data_17_18$season)
```

```
##  
## 2017 2018  
## 1382 1378
```

# Unir dataframes por columna

Si las columnas de los dataframes que queremos unir no son exactamente iguales, con `bind_rows()` se generan columnas con datos perdidos, mientras que con `rbind()` da error.

```
# Solo 2018
nba_data_18 <- filter(nba_data, season == 2018) %>%
  select(season, game_date_est)
```

```
# Solo 2017 y dos variables
nba_data_17 <- nba_data %>%
  filter(season == 2017) %>%
  select(season, pts_home)
```

```
colnames(nba_data_17)
```

```
## [1] "season" "pts_home"
```

```
nba_data_17_18 <- bind_rows(nba_data_17, nba_data_18)
head(nba_data_17_18, 3)
```

```
## # A tibble: 3 x 3
##   season pts_home game_date_est
##   <dbl>   <dbl> <date>
## 1  2017     85 NA
## 2  2017    102 NA
## 3  2017    122 NA
```

# Unir dataframes por fila

Cuando tenemos dos dataframes con las mismas observaciones pero distintas variables podemos utilizar `bind_cols()` para unirlos. También se puede utilizar `cbind()` del R Base.

```
# Dividamos el dataframe en 2 para volver a unirlo
nba_data_a <- nba_data %>%
  select(game_date_est, fg3_pct_home)

nba_data_b <- nba_data %>%
  select(ast_home, home_team_wins)

nba_data_C <- bind_cols(nba_data_a, nba_data_b)

colnames(nba_data_C)
```

```
## [1] "game_date_est" "fg3_pct_home" "ast_home" "home_team_wins"
```

# Joins

# Joins

- Cuando trabajamos con datos muchas veces debemos utilizar más de un conjunto de datos. Ya vimos que para combinar datos con las mismas columnas o las mismas filas usamos `bind_cols()` o `bind_rows()`.
- Para combinar datos con distintas estructuras podemos utilizar las funciones `*_join()` de dplyr.
- Para llevar a cabo estas operaciones necesitamos al menos una variable que identifique los casos en ambos dataframes (pueden llamarse de distinta forma). Estas variables se denominan key variables.



# Joins (ejemplo)

Un ejemplo: La NBA se divide en dos conferencias: este y oeste. Supongamos que queremos averiguar los equipos de cuál conferencia ganaron más partidos en los últimos 10 años. Filtrando el dataframe `nba_data` podemos conseguir fácilmente el resultado de todos los partidos en los últimos 10 años:

```
nba_u10 <- nba_data %>%  
  filter(season > 2010) %>%  
  select(home_team, visitor_team, pts_home, pts_away)  
  
dim(nba_u10)
```

```
## [1] 12240      4
```

```
head(nba_u10, 5)
```

```
## # A tibble: 5 x 4  
##   home_team      visitor_team pts_home pts_away  
##   <chr>         <chr>      <dbl>  <dbl>  
## 1 Orlando Magic Charlotte Hornets    120    117  
## 2 Washington Wizards Detroit Pistons      99     96  
## 3 Memphis Grizzlies Atlanta Hawks      116    117  
## 4 Indiana Pacers Philadelphia 76ers    107    113  
## 5 Toronto Raptors Miami Heat        105    117
```

# Joins (ejemplo)

El problema es que `nba_data` no tiene una variable que identifique la conferencia de cada equipo. Pero esa información está disponible en el dataframe `nba_teams`

```
nba_teams <- read_csv(here::here("data/nba_teams.csv")) %>%
  janitor::clean_names()

nba_teams <- nba_teams %>%
  select(city, nickname, conference)

glimpse(nba_teams)
```

```
## Rows: 30
## Columns: 3
## $ city      <chr> "Atlanta", "Boston", "New Orleans", "Chicago", "Dallas", "D~
## $ nickname  <chr> "Hawks", "Celtics", "Pelicans", "Bulls", "Mavericks", "Nugg~
## $ conference <chr> "east", "east", "west", "east", "west", "west", "west", "we~
```

# Joins (ejemplo)

De esta forma, queremos incorporar datos de un dataframe a nivel equipo de 30 observaciones a nuestra data por partido de 12240. La variable "key" es el equipo.

El primer paso antes de realizar un join es chequear que las variables identificadoras coincidan. Una primer mirada a nuestros dataframes nos indica que este no es nuestro caso: en `nba_data` las variables `home_team` y `visitor_team` los equipos están definidos por ciudad y nombre (Chicago Bulls, por ej.) mientras que en `nba_teams` están divididos en dos variables `city` y `nickname`.

```
pull(distinct(nba_teams, nickname))
```

## [1]	"Hawks"	"Celtics"	"Pelicans"	"Bulls"
## [5]	"Mavericks"	"Nuggets"	"Rockets"	"Clippers"
## [9]	"Lakers"	"Heat"	"Bucks"	"Timberwolves"
## [13]	"Nets"	"Knicks"	"Magic"	"Pacers"
## [17]	"76ers"	"Suns"	"Trail Blazers"	"Kings"
## [21]	"Spurs"	"Thunder"	"Raptors"	"Jazz"
## [25]	"Grizzlies"	"Wizards"	"Pistons"	"Hornets"
## [29]	"Cavaliers"	"Warriors"		

```
pull(distinct(nba_teams, city))
```

## [1]	"Atlanta"	"Boston"	"New Orleans"	"Chicago"
## [5]	"Dallas"	"Denver"	"Houston"	"Los Angeles"
## [9]	"Miami"	"Milwaukee"	"Minnesota"	"Brooklyn"
## [13]	"New York"	"Orlando"	"Indiana"	"Philadelphia"
## [17]	"Phoenix"	"Portland"	"Sacramento"	"San Antonio"
## [21]	"Oklahoma City"	"Toronto"	"Utah"	"Memphis"
## [25]	"Washington"	"Detroit"	"Charlotte"	"Cleveland"
## [29]	"Golden State"			

# Joins (ejemplo)

El primero paso entonces es unificar las categorías de las variables identificadoras.

```
nba_teams <- nba_teams %>%  
  mutate(team = paste(city, nickname)) %>% # Concateno ciudad y nombre  
  select(team, conference)  
pull(distinct(nba_teams, team))
```

## [1]	"Atlanta Hawks"	"Boston Celtics"	"New Orleans Pelicans"
## [4]	"Chicago Bulls"	"Dallas Mavericks"	"Denver Nuggets"
## [7]	"Houston Rockets"	"Los Angeles Clippers"	"Los Angeles Lakers"
## [10]	"Miami Heat"	"Milwaukee Bucks"	"Minnesota Timberwolves"
## [13]	"Brooklyn Nets"	"New York Knicks"	"Orlando Magic"
## [16]	"Indiana Pacers"	"Philadelphia 76ers"	"Phoenix Suns"
## [19]	"Portland Trail Blazers"	"Sacramento Kings"	"San Antonio Spurs"
## [22]	"Oklahoma City Thunder"	"Toronto Raptors"	"Utah Jazz"
## [25]	"Memphis Grizzlies"	"Washington Wizards"	"Detroit Pistons"
## [28]	"Charlotte Hornets"	"Cleveland Cavaliers"	"Golden State Warriors"

# Joins (ejemplo)

## Nuestros dataframes a combinar

```
glimpse(nba_teams)
```

```
## Rows: 30
## Columns: 2
## $ team      <chr> "Atlanta Hawks", "Boston Celtics", "New Orleans Pelicans", ~
## $ conference <chr> "east", "east", "west", "east", "west", "west", "west", "we~
```

```
glimpse(nba_u10)
```

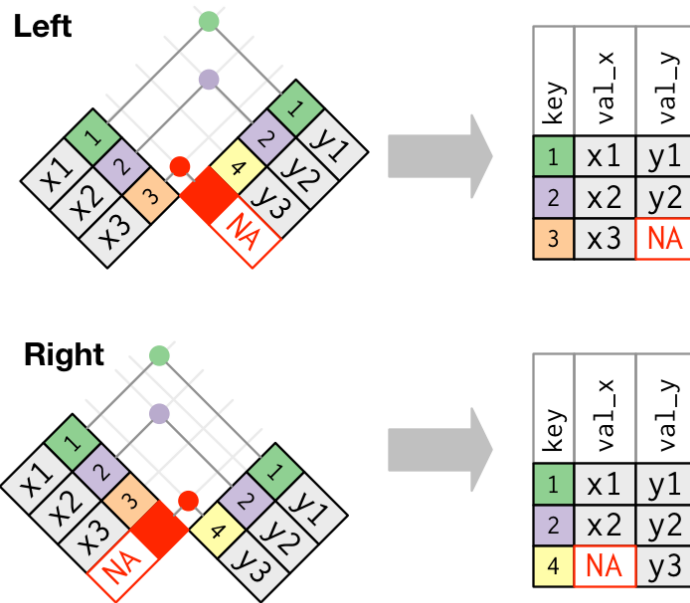
```
## Rows: 12,240
## Columns: 4
## $ home_team   <chr> "Orlando Magic", "Washington Wizards", "Memphis Grizzlies~
## $ visitor_team <chr> "Charlotte Hornets", "Detroit Pistons", "Atlanta Hawks", ~
## $ pts_home    <dbl> 120, 99, 116, 107, 105, 119, 89, 127, 103, 129, 113, 115, ~
## $ pts_away    <dbl> 117, 96, 117, 113, 117, 83, 113, 113, 105, 96, 114, 123, ~
```

# Joins

Ahora que tenemos los dos dataframes y que la variable identificadora tiene las mismas categorías, ¿cómo las unimos?

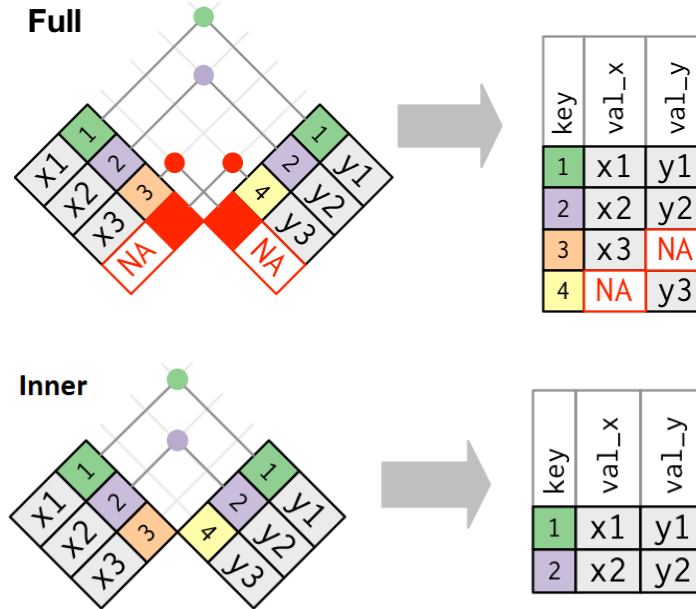
- dplyr tiene seis tipos de joins. Cuatro de ellos son **mutate** joins y dos son **filter** joins.
- Todos los joins tienen tres argumentos principales:
  - **x**: dataframe 1
  - **y**: dataframe 2
  - **by**: especificar variable identificadora
- Los cuatro tipos de mutate joins son:
  - `left_join()`: une incluyendo todas las filas en x
  - `right_join()`: une incluyendo todas las filas en y
  - `inner_join()`: une incluyendo todas las filas en x & y
  - `full_join()`: une incluyendo todas las filas en x o y

# Joins



Wickham & Grolemund (2018)

# Joins



Wickham & Grolemund (2018)



# Joins (ejemplo)

```
## Selecciono variable con nombre de equipo y conferencia
nba_teams_rec <- nba_teams %>%
  select(team, conference)

## Uno ambos dataframes usando left_join()
# Manera tradicional
nba_full <- left_join(x = nba_u10,
                     y = nba_teams_rec,
                     by = c("home_team" = "team"))

# Con pipeline
nba_full <- nba_u10 %>%
  left_join(nba_teams_rec, by = c("home_team" = "team"))

nba_full
```

```
## # A tibble: 12,240 x 5
##   home_team      visitor_team pts_home pts_away conference
##   <chr>          <chr>          <dbl>   <dbl>   <chr>
## 1 Orlando Magic Charlotte Hornets    120     117   east
## 2 Washington Wizards Detroit Pistons     99      96   east
## 3 Memphis Grizzlies Atlanta Hawks    116     117   west
## 4 Indiana Pacers Philadelphia 76ers    107     113   east
## 5 Toronto Raptors Miami Heat      105     117   east
## 6 New York Knicks Cleveland Cavaliers    119      83   east
## 7 Boston Celtics Brooklyn Nets      89     113   east
## 8 New Orleans Pelicans Milwaukee Bucks    127     113   west
## 9 Oklahoma City Thunder Chicago Bulls    103     105   west
## 10 Denver Nuggets Portland Trail Blazers    129      96   west
## # ... with 12,230 more rows
```

# Joins

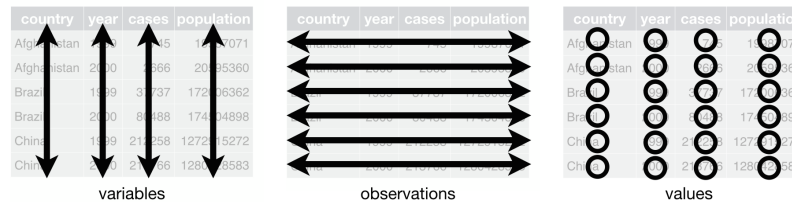
- Tanto `left_join()` como `right_join()` funcionan de la misma manera: mantienen el número de filas de uno de los dataframes. En `left_join()` se mantiene el número de observaciones de `x` mientras que en `right_join()` de `y`. Esto significa que: `left_join(data1, data2) = right_join(data2, data1)`. Usando `left_join()` en caso de que `y` no tenga datos sobre algún valor de la variable identificadora de `x`, se devolverá `NA`, pero no se borrará la observación
- `full_join()` mantiene todas las observaciones tanto de `x` como de `y`
- `inner_join()` mantiene las observaciones presentes en `x` e `y`. Descarta las observaciones presentes en `x` pero ausentes en `y`, y las observaciones presentes en `y` ausentes en `x`.

# Estructura de datos

# Estructura de datos

Un mismo conjunto de datos puede ser estructurado de distintas formas. Los criterios para ellos pueden ir desde facilitar la entrada de datos, a utilizar el formato correcto para correr ciertas funciones.

En el marco del Tidyverse se estructuran los datos en formato "tidy". Esto no significa que siempre que utilicemos R tengamos que estructurar los datos de esta forma, algunas funciones pueden requerir otros formatos. Sin embargo, para la mayoría de las funciones del tidyverse (particularmente para visualizar datos) funcionan mejor con el formato tidy



Wichkham & Grolemond (2018)

# Estructura de datos

El formato tidy parece obvio, pero muchas veces nos encontramos con datos distinta forma. Por ejemplo, no es tan extraño encontrar datos donde una sola variable está esparcida en varias columnas o una observación en más de una fila. Analizemos un caso muy común:

```
wb_desempleo <- data.frame(pais = c("Argentina", "Chile", "Uruguay"),  
                           d_2018 = c(9.2, 7.2, 8.3),  
                           d_2019 = c(9.8, 7.3, 9.3),  
                           d_2020 = c(11.7, 11.5, 12.7))
```

```
print(wb_desempleo)
```

```
##      pais d_2018 d_2019 d_2020  
## 1 Argentina   9.2    9.8   11.7  
## 2   Chile     7.2    7.3   11.5  
## 3  Uruguay    8.3    9.3   12.7
```

# Cambio de estructura de datos

Para pasar datos a estructura tidy tenemos las funciones `pivot_longer()` (para pasar de formato ancho a largo) y `pivot_wider()` (para pasar de formato largo a ancho) del paquete `tidyr`.


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
2	y	d
1	z	e
2	z	f

## De ancho a largo con pivot\_longer()



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

Wickham & Grolemund (2018)

# De ancho a largo con pivot\_longer()

```
wb_unemp_long <- wb_desempleo %>%  
  pivot_longer(cols = c("d_2018", "d_2019", "d_2020"), # Columnas a unir  
               names_to = "year", # Nombre de variable "key"  
               values_to = "desempleo") # Nombre de variable con valores  
print(wb_unemp_long)
```

```
## # A tibble: 9 x 3  
##   pais      year desempleo  
##   <chr>    <chr>      <dbl>  
## 1 Argentina d_2018        9.2  
## 2 Argentina d_2019        9.8  
## 3 Argentina d_2020       11.7  
## 4 Chile     d_2018        7.2  
## 5 Chile     d_2019        7.3  
## 6 Chile     d_2020       11.5  
## 7 Uruguay  d_2018        8.3  
## 8 Uruguay  d_2019        9.3  
## 9 Uruguay  d_2020       12.7
```



# De largo a ancho con pivot\_wider()

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

table2

Wickham & Grolemund (2018)

# De largo a ancho con pivot\_wider()

```
print(wb_unemp)
```

```
##      pais year      variable  valor
## 1 Argentina 2018    desempleo    9.2
## 2 Argentina 2019    desempleo    9.8
## 3 Argentina 2020    desempleo   11.7
## 4 Argentina 2018 pbi_per_capita 11633.0
## 5 Argentina 2019 pbi_per_capita  9912.0
```

```
wb_unemp %>%
  pivot_wider(names_from = variable,
              values_from = valor)
```

```
## # A tibble: 3 x 4
##   pais      year desempleo pbi_per_capita
##   <chr>    <dbl>    <dbl>         <dbl>
## 1 Argentina 2018      9.2         11633
## 2 Argentina 2019      9.8          9912
## 3 Argentina 2020     11.7           NA
```

# pivot\_wider() después de summarise()

Muchas veces luego de resumir los datos (sobretudo cuando agrupamos por más de una variable) queremos pasar los datos de formato largo a formato ancho. Una salida típica de `summarise()` después de agrupar por dos variables:

```
nba_data %>%
  mutate(home_team_wins = case_when(
    home_team_wins == 1 ~ "Ganados",
    TRUE ~ "Perdidos"
  )) %>%
  filter(home_team == "Chicago Bulls") %>%
  group_by(season, home_team_wins) %>%
  summarise(n = n())
```

```
## # A tibble: 36 x 3
## # Groups:   season [18]
##   season home_team_wins      n
##   <dbl> <chr>         <int>
## 1  2003 Ganados          14
## 2  2003 Perdidos         31
## 3  2004 Ganados          30
## 4  2004 Perdidos         16
## 5  2005 Ganados          25
## 6  2005 Perdidos         23
## 7  2006 Ganados          38
## 8  2006 Perdidos         13
## 9  2007 Ganados          24
## 10 2007 Perdidos         21
## # ... with 26 more rows
```

# pivot\_wider() después de summarise()

Si agregamos `pivot_wider()` al final:

```
nba_data %>%  
  mutate(home_team_wins = case_when(  
    home_team_wins == 1 ~ "Ganados",  
    TRUE ~ "Perdidos"  
  )) %>%  
  filter(home_team == "Chicago Bulls") %>%  
  group_by(season, home_team_wins) %>%  
  summarise(n = n()) %>%  
  pivot_wider(names_from = home_team_wins,  
              values_from = n)
```

```
## # A tibble: 18 x 3  
## # Groups:   season [18]  
##   season Ganados Perdidos  
##   <dbl>   <int>   <int>  
## 1  2003     14     31  
## 2  2004     30     16  
## 3  2005     25     23  
## 4  2006     38     13  
## 5  2007     24     21  
## 6  2008     32     17  
## 7  2009     29     19  
## 8  2010     45      9  
## 9  2011     29      8  
## 10 2012     31     21  
## 11 2013     32     17  
## 12 2014     33     19  
## 13 2015     29     17  
## 14 2016     27     22  
## 15 2017     18     26  
## 16 2018     11     33  
## 17 2019     15     22  
## 18 2020      1      1
```