

Análisis de Datos

Tema 3 - Data Wrangling

3.2 Organización de Datos

Roi Naveiro

Data Wrangling

Objetivo: dejar los datos listos para su posterior exploración y modelización

Convertir datos crudos en datos procesados

Datos crudos

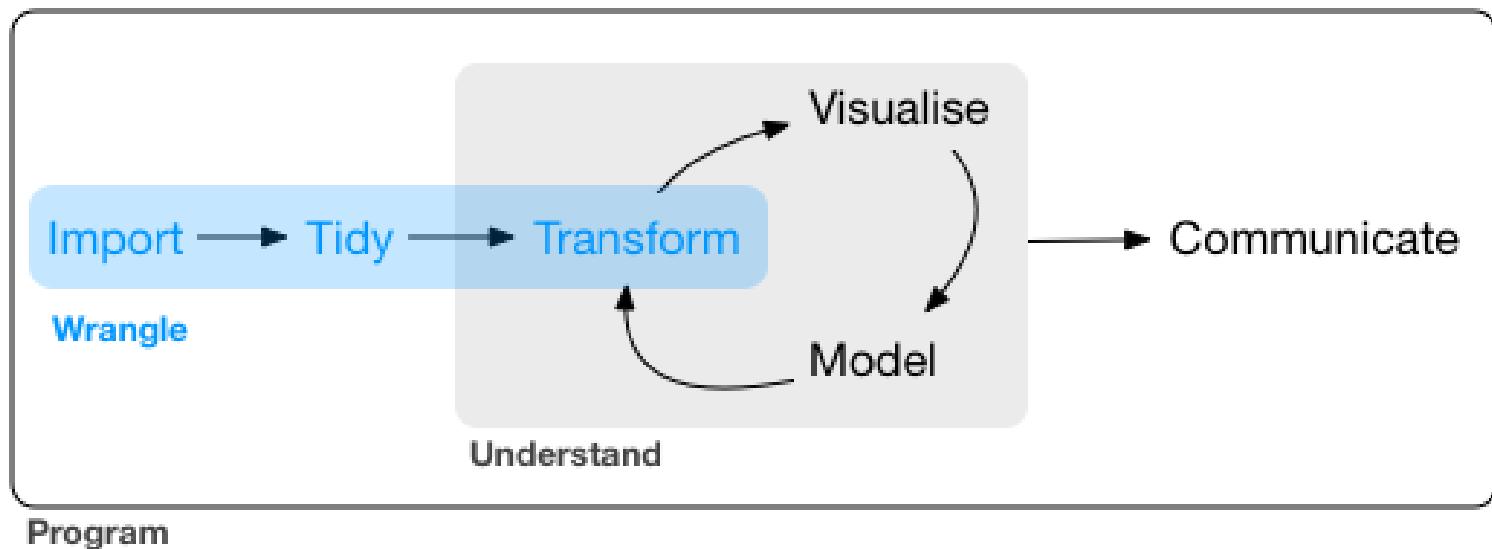
- Los datos tal cual aparecen en la fuente de origen
- No han sufrido ninguna manipulación

Datos procesados

- Cada variable es una columna
- Cada observación una fila
- Cada unidad observacional es una celda
- Datos más complejos, en varias tablas interconectadas

Data Wrangling

- Importación de los datos
- Organización de los datos
- Transformación de los datos



Organización de datos

Datos Organizados

- Aprenderemos forma consistente de organizar los datos en R
- Lo haremos a través del paquete **tidyverse**

Datos Organizados

- Los mismos datos pueden representarse de múltiples formas
- En el siguiente ejemplo presentamos los valores de cuatro variables (*country, year, population, cases*) de cuatro formas distintas

table1

```
## # A tibble: 6 × 4
##   country     year   cases population
##   <chr>       <int>   <int>      <int>
## 1 Afghanistan 1999     745 19987071
## 2 Afghanistan 2000    2666 20595360
## 3 Brazil       1999   37737 172006362
## 4 Brazil       2000   80488 174504898
## 5 China        1999  212258 1272915272
## 6 China        2000  213766 1280428583
```

Datos Organizados

- Los mismos datos pueden representarse de múltiples formas
- En el siguiente ejemplo presentamos los valores de cuatro variables (*country, year, population, cases*) de cuatro formas distintas

table2

```
## # A tibble: 12 × 4
##   country     year type     count
##   <chr>      <int> <chr>    <int>
## 1 Afghanistan 1999 cases     745
## 2 Afghanistan 1999 population 19987071
## 3 Afghanistan 2000 cases     2666
## 4 Afghanistan 2000 population 20595360
## 5 Brazil       1999 cases     37737
## 6 Brazil       1999 population 172006362
## 7 Brazil       2000 cases     80488
## 8 Brazil       2000 population 174504898
## 9 China        1999 cases     212258
## 10 China       1999 population 1272915272
## 11 China       2000 cases     213766
## 12 China       2000 population 1280428583
```

Datos Organizados

- Los mismos datos pueden representarse de múltiples formas
- En el siguiente ejemplo presentamos los valores de cuatro variables (*country, year, population, cases*) de cuatro formas distintas

table3

```
## # A tibble: 6 × 3
##   country     year    rate
## * <chr>      <int> <chr>
## 1 Afghanistan 1999  745/19987071
## 2 Afghanistan 2000  2666/20595360
## 3 Brazil       1999  37737/172006362
## 4 Brazil       2000  80488/174504898
## 5 China        1999  212258/1272915272
## 6 China        2000  213766/1280428583
```

Datos Organizados

- Los mismos datos pueden representarse de múltiples formas
- En el siguiente ejemplo presentamos los valores de cuatro variables (*country, year, population, cases*) de cuatro formas distintas

table4a

```
## # A tibble: 3 × 3
##   country     `1999` `2000`
## * <chr>       <int>   <int>
## 1 Afghanistan    745    2666
## 2 Brazil        37737   80488
## 3 China         212258  213766
```

table4b

```
## # A tibble: 3 × 3
##   country     `1999`     `2000`
## * <chr>       <int>      <int>
## 1 Afghanistan 19987071  20595360
## 2 Brazil      172006362 174504898
## 3 China       1272915272 1280428583
```

Datos Organizados

- Solo uno está organizado
- Hablamos de datos organizados si:
 - Cada variable corresponde a una columna
 - Cada fila a un observación
 - Cada celda a un valor

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

table1

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

variables

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

observations

Datos Organizados

- La principal ventaja es poder utilizar las herramientas de R que, usualmente, trabajan sobre vectores de valores.

```
table1 %>% mutate(rate = 10000 * cases / population)
```

```
## # A tibble: 6 × 5
##   country     year   cases population    rate
##   <chr>     <int>   <int>      <int>   <dbl>
## 1 Afghanistan 1999     745     19987071 0.373
## 2 Afghanistan 2000    2666     20595360 1.29
## 3 Brazil       1999   37737    172006362 2.19
## 4 Brazil       2000   80488    174504898 4.61
## 5 China        1999  212258   1272915272 1.67
## 6 China        2000  213766   1280428583 1.67
```

Datos Organizados

- La mayoría de datos que recibimos, no están organizados.
- Primer paso: determinar qué son las variables y qué son las observaciones. ¿Ejemplos?
- Segundo paso: resolver uno de estos problemas
 - Una variable está dispersa por varias columnas
 - Una observación está dispersa por múltiples filas
- Esto se resuelve con **pivot_longer()** y **pivot_wider()**

pivot_longer()

- Nombres de columnas son valores de variable

table4a

```
## # A tibble: 3 × 3
##   country    `1999` `2000`
## * <chr>      <int>  <int>
## 1 Afghanistan     745    2666
## 2 Brazil          37737   80488
## 3 China           212258  213766
```

pivot_longer()

- Necesitamos **pivolar** estas columnas a un nuevo par de variables. Para ello, debemos conocer:
 - El conjunto de columnas a pivotar
 - El nombre de la variable recibirá las columnas: **year**
 - El nombre de la variable que recibirá los valores: **cases**

pivot_longer()

```
table4a %>% pivot_longer(c(`1999`, `2000`),  
                           names_to = "year", values_to = "cases")
```

```
## # A tibble: 6 × 3  
##   country     year   cases  
##   <chr>       <chr>   <int>  
## 1 Afghanistan 1999     745  
## 2 Afghanistan 2000    2666  
## 3 Brazil      1999  37737  
## 4 Brazil      2000  80488  
## 5 China       1999 212258  
## 6 China       2000 213766
```

- Nótese que los años son nombres que no empiezan por letra y por tanto han de rodearse de tildes...

pivot_longer()

Usa **pivot_longer()** para ordenar el dataset **table4b**

pivot_wider()

- Se utiliza cuando tenemos una observación dispersa en filas
- En el siguiente dataset, cada observación es un país en un año
- Cada una ocupa dos filas

```
table2
```

```
## # A tibble: 12 × 4
##   country     year type     count
##   <chr>       <int> <chr>    <int>
## 1 Afghanistan 1999 cases      745
## 2 Afghanistan 1999 population 19987071
## 3 Afghanistan 2000 cases     2666
## 4 Afghanistan 2000 population 20595360
## 5 Brazil       1999 cases     37737
## 6 Brazil       1999 population 172006362
## 7 Brazil       2000 cases     80488
## 8 Brazil       2000 population 174504898
## 9 China        1999 cases     212258
## 10 China       1999 population 1272915272
## 11 China       2000 cases     213766
## 12 China       2000 population 1280428583
```

pivot_wider()

Para ordenar, necesitamos:

- La columna de la que extraeremos los nombres de las nuevas variables: **type**
- La columna de la que extraeremos sus valores: **count**

pivot_wider()

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

```
## # A tibble: 6 × 4  
##   country     year   cases population  
##   <chr>       <int>   <int>      <int>  
## 1 Afghanistan 1999     745 19987071  
## 2 Afghanistan 2000    2666 20595360  
## 3 Brazil       1999  37737 172006362  
## 4 Brazil       2000  80488 174504898  
## 5 China        1999 212258 1272915272  
## 6 China        2000 213766 1280428583
```

Organizar datos

- **pivot_longer**: elimina columnas y añade filas, hace los datos más largos.
- **pivot_wider**: elimina filas y añade columnas, hace los datos más anchos.

Separar y unir columnas

- A veces, una misma columna contiene información acerca de dos variables.
- Se puede separar en dos columnas utilizando **separate()**

```
table3
```

```
## # A tibble: 6 × 3
##   country     year    rate
## * <chr>       <int> <chr>
## 1 Afghanistan 1999  745/19987071
## 2 Afghanistan 2000  2666/20595360
## 3 Brazil      1999  37737/172006362
## 4 Brazil      2000  80488/174504898
## 5 China       1999  212258/1272915272
## 6 China       2000  213766/1280428583
```

Separar y unir columnas

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

```
## # A tibble: 6 × 4
##   country      year cases population
##   <chr>        <int> <chr>    <chr>
## 1 Afghanistan  1999  745     19987071
## 2 Afghanistan  2000  2666    20595360
## 3 Brazil       1999  37737   172006362
## 4 Brazil       2000  80488   174504898
## 5 China        1999  212258  1272915272
## 6 China        2000  213766  1280428583
```

Separar y unir columnas

- Por defecto, separa cuando encuentra carácter no alfanumérico.
- Se puede especificar el carácter.

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

```
## # A tibble: 6 × 4
##   country     year cases population
##   <chr>       <int> <chr>    <chr>
## 1 Afghanistan 1999  745     19987071
## 2 Afghanistan 2000  2666    20595360
## 3 Brazil       1999  37737   172006362
## 4 Brazil       2000  80488   174504898
## 5 China        1999  212258  1272915272
## 6 China        2000  213766  1280428583
```

Separar y unir columnas

- Además, conviene convertir el tipo de columna, de lo contrario será de tipo string.

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

```
## # A tibble: 6 × 4
##   country     year   cases population
##   <chr>       <int>   <int>      <int>
## 1 Afghanistan 1999     745 19987071
## 2 Afghanistan 2000    2666 20595360
## 3 Brazil       1999  37737 172006362
## 4 Brazil       2000  80488 174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

Separar y unir columnas

- Se puede separar usando enteros

```
table5 <-  
  table3 %>% separate(year, into = c("century", "year"), sep = 2)
```

Separar y unir columnas

- **unite()** hace lo contrario que **separate()**: combina columnas
- Toma como argumentos: nueva variable, conjunto de columnas a juntar y **sep**

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

```
## # A tibble: 6 × 3
##   country     new    rate
##   <chr>      <chr>  <chr>
## 1 Afghanistan 1999  745/19987071
## 2 Afghanistan 2000  2666/20595360
## 3 Brazil       1999  37737/172006362
## 4 Brazil       2000  80488/174504898
## 5 China        1999  212258/1272915272
## 6 China        2000  213766/1280428583
```

Datos relacionales

Datos relacionales

- Generalmente, en análisis de datos se trabaja con más de una base de datos.
- Una de las mayores fuentes de riqueza en análisis de datos surge del cruce de bbdd!
- Hay que saber cómo enganchar diferentes tablas de datos.
- Cuando los datos están repartidos en múltiples tablas, hablamos de **datos relacionales**

Datos relacionales

En **tidyverse**, existen tres familias de verbos para trabajar con datos relacionales:

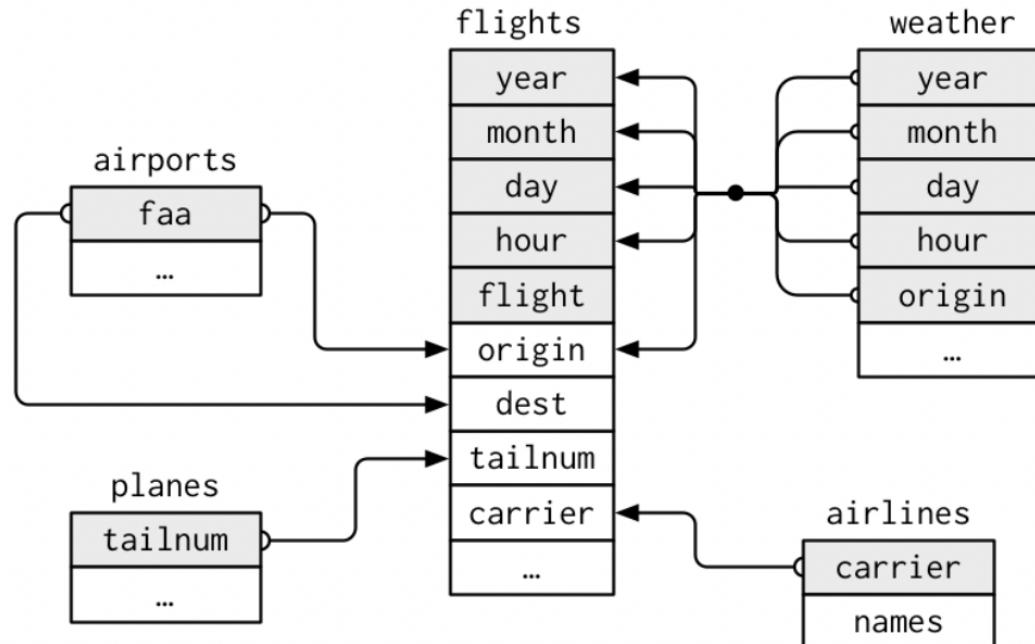
- **Mutating joins**: añaden nuevas variables a una tabla de datos, procedentes de observaciones coincidentes de otra tabla.
- **Filtering joins**: filtran observaciones de una base de datos basándose en si coinciden o no con las observaciones de otra tabla.
- **Set operations**: utilizan las observaciones como si fueran elementos de un conjunto.

nycflights13

Utilizaremos este conjunto de datos para aprender sobre datos relacionales. Contiene las siguientes tablas:

- **airlines**: Nombres de aerolíneas
- **airports**: Información sobre cada aeropuerto
- **planes**: Información sobre cada avión
- **weather**: Información climática en cada aeropuerto de NYC a cada hora
- **flights**: Información sobre todos los vuelos que partieron de NYC en 2013

nycflights13



Keys (Llaves)

- Son **variables** que se usan para conectar pares de tablas.
- Conjunto de variables que identifican únicamente una observación.
- ¿Cuáles son las llaves de cada una de las tablas de datos de **nycflights13**?

Keys (Llaves)

Dos tipos de llaves:

- Primary Key: identifica únicamente una observación en su propia tabla.
- Foreign Key: identifica únicamente una observación en otra tabla.
- ¿Ejemplo en **nycflights13**?

Keys (Llaves)

Una vez identificada una llave primaria, es útil comprobar que es correcta

```
planes %>%
  count(tailnum) %>%
  filter(n > 1)
```

```
## # A tibble: 0 × 2
## # ... with 2 variables: tailnum <chr>, n <int>
```

```
weather %>%
  count(year, month, day, hour, origin) %>%
  filter(n > 1)
```

```
## # A tibble: 3 × 6
##   year month   day hour origin     n
##   <int> <int> <int> <int> <chr>   <int>
## 1 2013    11     3     1 EWR       2
## 2 2013    11     3     1 JFK       2
## 3 2013    11     3     1 LGA       2
```

Keys (Llaves)

- A veces no existen llaves primarias! No hay una combinación de variables que identifique únicamente cada observación.
- Ejemplo, **flights**.

Útil añadir una... llave surrogada

```
flights_key <- flights %>% mutate(key = row_number())
```

```
flights_key %>%
  count(key) %>%
  filter(n > 1)
```

```
## # A tibble: 0 × 2
## # ... with 2 variables: key <int>, n <int>
```

Keys (Llaves)

- Una *primary key* y su *foreign key* en otra table forman una **relación**.
- Usualmente las relaciones son 1 a varios.
- Ejemplo: cada vuelo tiene un único avión, pero cada avión realiza muchos vuelos.

Mutating joins

Mutating joins

- Sirve para combinar variables de dos tablas.
- Primero, empareja observaciones usando las llaves correspondientes, y después copia variables de una tabla a otra.
- Añade nuevas variables a la derecha!! Trabajamos con versión reducida

```
flights2 <- flights %>%
  select(year:day, hour, origin, dest, tailnum, carrier)
flights2 %>% slice(1:3)
```

```
## # A tibble: 3 × 8
##   year month   day hour origin dest tailnum carrier
##   <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>
## 1 2013     1     1     5 EWR    IAH    N14228  UA
## 2 2013     1     1     5 LGA    IAH    N24211  UA
## 3 2013     1     1     5 JFK    MIA    N619AA  AA
```

Mutating joins

Para entender su funcionamiento, trabajamos con datos artificiales

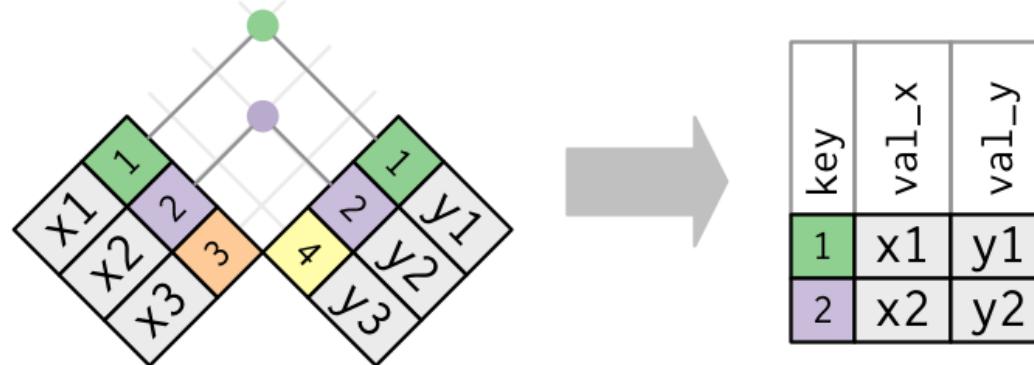
```
x <- tribble(  
  ~key, ~val_x,  
  1, "x1",  
  2, "x2",  
  3, "x3"  
)  
y <- tribble(  
  ~key, ~val_y,  
  1, "y1",  
  2, "y2",  
  4, "y3"  
)
```

Mutating joins

x		y	
1	x1	1	y1
2	x2	2	y2
3	x3	4	y3

Inner join

Empareja observaciones únicamente cuando las llaves son iguales.



Inner join

La salida es un nuevo data frame con llave, variable x y variable y. Las filas no emparejadas no aparecen!!

```
x %>%  
  inner_join(y, by = "key")
```

```
## # A tibble: 2 × 3  
##   key  val_x val_y  
##   <dbl> <chr> <chr>  
## 1     1    x1    y1  
## 2     2    x2    y2
```

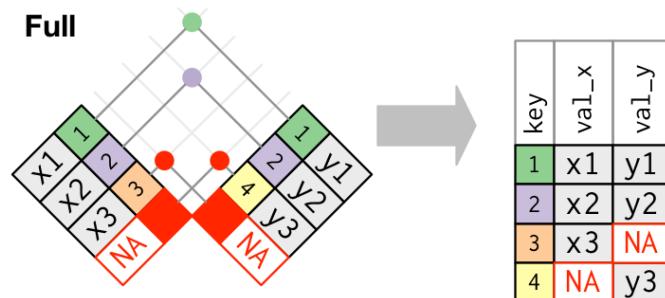
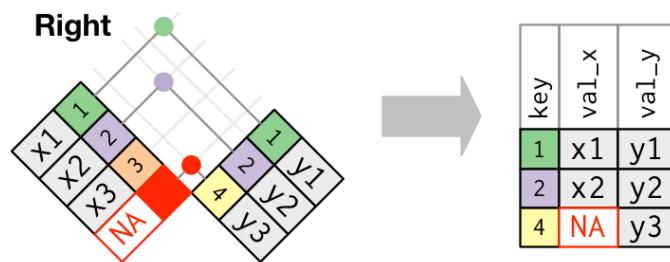
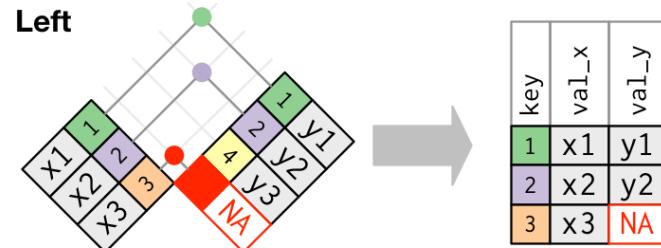
Outer join

Mantiene observaciones que aparecen **al menos** en uno de los dataset!!

Tres tipos:

- **Left join**: mantiene todas las observaciones de **x**
- **Right join**: mantiene todas las observaciones de **y**
- **Full join**: mantiene todas las observaciones de **x e y**

Outer join



Outer join

Predice la respuesta de los siguientes códigos

```
x %>%  
  left_join(y, by = "key")  
  
x %>%  
  right_join(y, by = "key")  
  
x %>%  
  full_join(y, by = "key")
```

Outer join

Predice la respuesta de los siguientes códigos

```
x %>%  
  left_join(y, by = "key")
```

```
## # A tibble: 3 × 3  
##   key val_x val_y  
##   <dbl> <chr> <chr>  
## 1     1 x1    y1  
## 2     2 x2    y2  
## 3     3 x3    <NA>
```

```
x %>%  
  right_join(y, by = "key")
```

```
## # A tibble: 3 × 3  
##   key val_x val_y  
##   <dbl> <chr> <chr>  
## 1     1 x1    y1  
## 2     2 x2    y2  
## 3     3 <NA>  y3
```

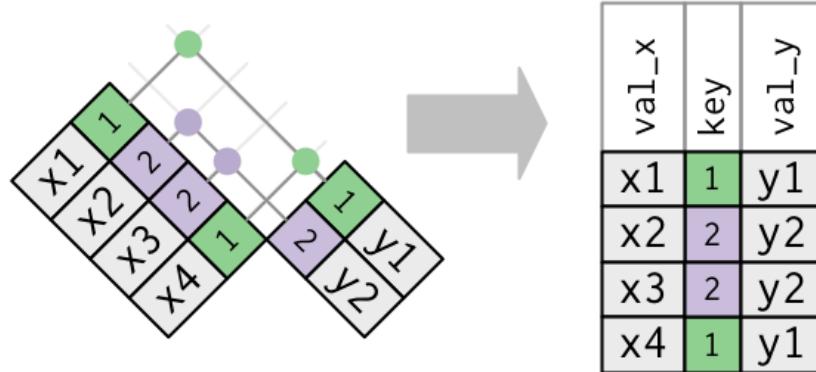
Outer join

Predice la respuesta de los siguientes códigos

```
x %>%
  full_join(y, by = "key")
```

```
## # A tibble: 4 × 3
##       key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
## 3     3 x3    <NA>
## 4     4 <NA>  y3
```

¿Llaves duplicadas?



- Caso 1: Una sola tabla tiene llaves duplicadas
- Caso 2: Ambas tablas tienen llaves duplicadas

Llaves duplicadas - Caso 1

Caso típico!

```
x <- tribble(  
  ~key, ~val_x,  
  1, "x1",  
  2, "x2",  
  2, "x3",  
  1, "x4"  
)  
y <- tribble(  
  ~key, ~val_y,  
  1, "y1",  
  2, "y2"  
)  
left_join(x, y, by = "key")
```

```
## # A tibble: 4 × 3  
##   key val_x val_y  
##   <dbl> <chr> <chr>  
## 1     1 x1    y1  
## 2     2 x2    y2  
## 3     2 x3    y2  
## 4     1 x4    y1
```

Llaves duplicadas - Caso 1

```
left_join(x, y, by = "key")
```

```
## # A tibble: 4 × 3
##       key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
## 3     2 x3    y2
## 4     1 x4    y1
```

Llaves duplicadas - Caso 2

Error, llave no identifica únicamente las observaciones

```
x <- tribble(  
  ~key, ~val_x,  
  1, "x1",  
  2, "x2",  
  2, "x3",  
  3, "x4"  
)  
y <- tribble(  
  ~key, ~val_y,  
  1, "y1",  
  2, "y2",  
  2, "y3",  
  3, "y4"  
)
```

Llaves duplicadas - Caso 2

Todas las combinaciones!

```
left_join(x, y, by = "key")  
  
## # A tibble: 6 × 3  
##   key    val_x val_y  
##   <dbl> <chr> <chr>  
## 1     1 x1    y1  
## 2     2 x2    y2  
## 3     2 x2    y3  
## 4     2 x3    y2  
## 5     2 x3    y3  
## 6     3 x4    y4
```

Ejercicio

Utilizando los datos **flights** y **airlines**, crea una base de datos donde aparezcan año, mes, día y hora de vuelo, así como la variable **carrier** que contiene las siglas de la aerolínea correspondiente y la variable **name** que tiene el nombre completo de la aerolínea.

Definiendo las llaves

- Hasta ahora, la llave era una única variable, con el mismo nombre en ambas tablas.
- Indicado en **by = "key"**
- ¿Otros usos de **by**?

Usos de by

- **by = NULL** es el valor por defecto. Usa todas las variables que aparecen en ambas tablas
- ¿Qué variables se utilizarán en el siguiente código?

```
flights2 %>%
  left_join(weather)
```

Usos de by

- **by** = "x" une según variable/s indicad/sa, con **mismo nombre** en ambas tablas.

```
flights2 %>%
  left_join(planes, by = "tailnum")
```

Usos de by

- **by = c("a" = "b")** empareja la variable **a** de la primera tabla con la variable **b** de la segunda.
- ¿Qué pasa?

```
flights2 %>%
  left_join(airports, c("dest" = "faa"))
```

Ejercicio

¿Qué variables usarías como llave para juntar los datos **planes** y **flights**?

Pista: usa las ayudas para entender cada uno de los datos

Ejercicio

¿Cuáles son los nombres completos de los aeropuertos de destino con el retraso medio más alto? Presenta una tabla con dos columnas, una con nombre completo y la otra con el retraso medio.

Mutating joins

Consejo: utiliza siempre que puedas el **left_join**. Es la función más importante

Filtering joins

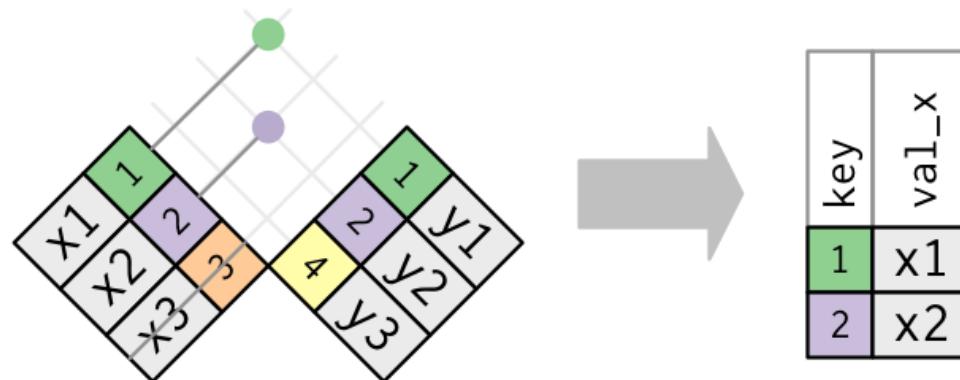
Filtering joins

Similar a los mutating joins, pero afectan a observaciones, no a variables!

Dos tipos

- **semi_join(x,y)**: mantienen todas las observaciones de **x** que tienen pareja en **y**
- **anti_join(x,y)**: elimina todas las observaciones de **x** que tienen pareja en **y**

semi_join(x,y)



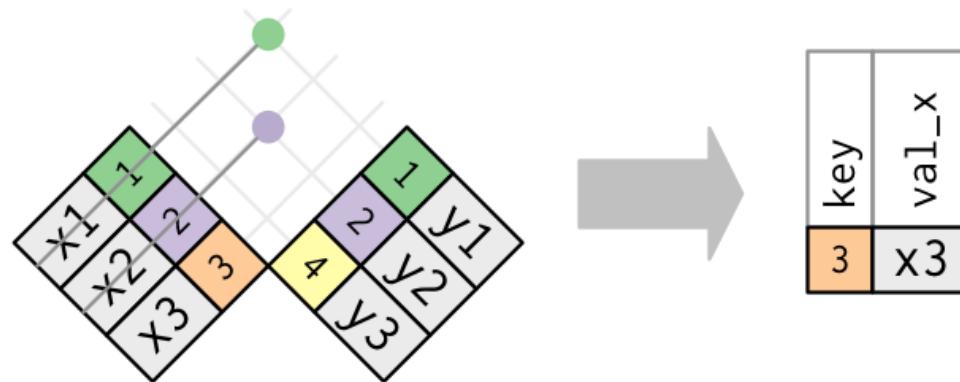
semi_join(x,y)

Útil para recuperar variables tras resumen.

Ejercicio: extrae toda la información de los vuelos de **flights** cuyo destino esté entre los 10 destinos con más observaciones.

```
top_dest <- flights %>% count(dest) %>% arrange(desc(n)) %>%
  slice(1:10)
```

anti_join(x,y)



anti_join(x,y)

Útil para diagnosticar discordancias.

```
flights %>%  
  anti_join(planes, by = "tailnum") %>%  
  count(tailnum) %>% arrange(desc(n))
```

```
## # A tibble: 722 × 2  
##   tailnum     n  
##   <chr>    <int>  
## 1 <NA>      2512  
## 2 N725MQ     575  
## 3 N722MQ     513  
## 4 N723MQ     507  
## 5 N713MQ     483  
## 6 N735MQ     396  
## 7 N0EGMQ     371  
## 8 N534MQ     364  
## 9 N542MQ     363  
## 10 N531MQ    349  
## # ... with 712 more rows
```

Set operations

Set operations

Esperan **x** e **y** con las **mismas** variables. Tratan observaciones como conjuntos:

- **intersect(x,y)**: Devuelve observaciones presentes en ambos **x** e **y**.
- **union(x,y)**: Devuelve observaciones únicas presentes en **x** o **y**.
- **setdiff(x,y)**: Devuelve observaciones presentes en **x** pero no en **y**.

Set operations

```
df1 <- tribble(  
  ~x, ~y,  
  1, 1,  
  2, 1  
)  
df2 <- tribble(  
  ~x, ~y,  
  1, 1,  
  1, 2  
)
```

Set operations

```
intersect(df1, df2)
```

```
## # A tibble: 1 × 2
##       x     y
##   <dbl> <dbl>
## 1     1     1
```

```
union(df1, df2)
```

```
## # A tibble: 3 × 2
##       x     y
##   <dbl> <dbl>
## 1     1     1
## 2     2     1
## 3     1     2
```

```
setdiff(df1, df2)
```

```
## # A tibble: 1 × 2
##       x     y
##   <dbl> <dbl>
## 1     2     1
```

Bibliografía

Este tema está fundamentalmente basado en [R for Data Science](#), Wickham and Grolemund (2016)