# dplyr

## Fundamentos lenguajes: R

Alberto Torres Barrán y Irene Rodríguez Luján

2019-11-15

# Tidy Data

- El 80% del tiempo del análisis de datos se pasa limpiando y preparando datos (Dasu and Johnson 2003).
- Una vez cargados los datos, es conveniente estructurarlos de forma que el procesado posterior sea lo más sencillo posible.
- Una estructura muy común son los datos ordenados o *tidy data*.
- Hadley y Wickhan (2014) los definen como aquellos donde:
    1. Cada variable forma una columna.
    2. Cada observación o muestra forma una fila.
    3. Cada tipo de unidad de observación forma una tabla.
- El paquete `dplyr` ayuda a generar datos ordenados así como transformarlos y analizarlos de forma eficiente. Algunos ejemplos de manipulaciones de datos con dplyr:
    - Seleccionar subconjuntos de filas y/o columnas.
    - Agrupar datos.
    - Calcular diferentes estadísticos sobre datos (posiblemente agrupados).
    - Generar nuevas variables.
    - Combinar/cruzar tablas.

# Introducción

- Implementa una gramática para realizar operaciones básicas con data frames.
- Muy eficiente.
- Operaciones principales:
  - `slice`: Selecciona filas por su posición.
  - `filter`: Selecciona filas por condición.
  - `select`: Selecciona variables (columnas) de un dataframe.
  - `arrange`: Ordena las filas de un dataframe.
  - `mutate`: Añade nuevas variables(columnas) al dataframe como combinación de las ya existentes.
  - `summarize`: Colapsa el dataframe a una única fila.
- Cada una de estas operaciones realiza una tarea concreta. Esto simplifica la legibilidad del código.
- Estas operaciones se pueden componer para realizar otras más complejas utilizando **pipelines** (`%>%`).

```
iris %>%
  select(Sepal.Width) %>%
  arrange(Sepal.Width)
```

# Tibbles

- Las tablas en dplyr (tibbles) son del tipo `tbl`.
- La clase `tbl` es una "redefinición moderna" de `data.frame`: mantiene lo que es efectivo de `data.frame` y elimina lo que no.

- **¿Qué diferencia este tipo de datos de un data.frame?**

    - La impresión de los datos es diferente.
    - Tibbles son estrictos en cuanto a la selección de columnas.
    - La selección de filas y/o columnas en tibbles siempre devuelve otro tibble.

- Se puede crear una tabla tibble con la función `tibble`:

```
tibble(
  x = 1:3,
  z = x ^ 2,
  txt = "hola"
)
```

- Se puede hacer que un data.frame pase a ser tibble con las funcniones `as_tibble()` o `tbl_df`: `as_tibble(iris)` o `tbl_df(iris)`.

# Funciones básicas

- Las funciones vistas para data.frames `head`, `tail`, `View`, `dim`, `nrow`, `ncol`, `rownames`, `colnames`, `str`, `summary` son válidas también con tablas de tipo `tbl`.
- Además, `dplyr` tiene la función `glimpse` que muestra una visión compacta del dataframe.

```
glimpse(iris)
```

```
## Observations: 150
## Variables: 5
## $ Sepal.Length <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4…
## $ Sepal.Width  <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2…
## $ Petal.Length <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1…
## $ Petal.Width  <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0…
## $ Species      <fct> setosa, setosa, setosa, setosa, setosa, s…
```

# slice

Selecciona filas por su posición

```
# con indices positivos
slice(mpg, 1:3)
```

```
## # A tibble: 3 x 11
##   manufacturer model displ  year   cyl trans drv     cty   hwy
##   <chr>        <chr> <dbl> <int> <int> <chr> <chr> <int> <int>
## 1 audi         a4      1.8  1999     4 auto… f        18    29
## 2 audi         a4      1.8  1999     4 manu… f        21    29
## 3 audi         a4      2    2008     4 manu… f        20    31
## # … with 2 more variables: fl <chr>, class <chr>
```

```
# descartar filas usando indices negativos
slice(mpg, -4:-234)
```

```
## # A tibble: 3 x 11
##   manufacturer model displ  year   cyl trans drv     cty   hwy
##   <chr>        <chr> <dbl> <int> <int> <chr> <chr> <int> <int>
## 1 audi         a4      1.8  1999     4 auto… f        18    29
## 2 audi         a4      1.8  1999     4 manu… f        21    29
```

# filter

Selecciona filas por condiciones.

```
filter(mpg, model == "a4")
```

```
## # A tibble: 7 x 11
##   manufacturer model displ  year   cyl trans drv     cty   hwy
##   <chr>        <chr> <dbl> <int> <int> <chr> <chr> <int> <int>
## 1 audi         a4      1.8  1999     4 auto… f        18    29
## 2 audi         a4      1.8  1999     4 manu… f        21    29
## 3 audi         a4      2    2008     4 manu… f        20    31
## 4 audi         a4      2    2008     4 auto… f        21    30
## 5 audi         a4      2.8  1999     6 auto… f        16    26
## 6 audi         a4      2.8  1999     6 manu… f        18    26
## 7 audi         a4      3.1  2008     6 auto… f        18    27
## # … with 2 more variables: fl <chr>, class <chr>
```

# Operadores lógicos

- Con la función `filter` se pueden seleccionar filas por condiciones lógicas.
- Dado que las condiciones se aplican sobre columnas del tibble y estas columnas son vectores, los operadores lógicos sobre vectores (devuelven vectores lógicos) se pueden utilizar para filtrar las filas de un dataframe.

| Logic in R - ?Comparison, ?base::Logic | | | |
|---|---|---|---|
| < | Less than | != | Not equal to |
| > | Greater than | %in% | Group membership |
| == | Equal to | is.na | Is NA |
| <= | Less than or equal to | !is.na | Is not NA |
| >= | Greater than or equal to | &, \|, !, xor, any, all | Boolean operators |

# filter (cont.)

- Se pueden combinar multiples condiciones separadas por `,` (and lógico)

```
filter(mpg, model == "a4", cyl >= 5)
```

- También se puede usar explicitamente el operador `&`.

```
filter(mpg, model == "a4" & cyl >= 5)
```

- En el caso del or lógico es obligatorio el uso del operador `|`.

```
filter(mpg, model == "a4" | model == "mustang")
```

- Se pueden usar algunas de las funciones lógicas anteriores

```
filter(mpg, model %in% c("tiburon","new beetle"))
```

# Otras funciones para seleccionar filas

- `distinct`: Eliminar filas duplicadas (todas las columnas deben ser idénticas)

```
mpg2 <- rbind(mpg, slice(mpg,1))
dim(mpg2)
nodup <- distinct(mpg2)
dim(mpg)
```

- `sample_frac`: Seleccionar aleatoriamente una proporción de las filas.

```
sample_frac(mpg, 0.5, replace=T)
```

- `sample_n`: Seleccionar aleatoriamente un número de filas.

```
sample_n(mpg, 10, replace=F)
```

- `top_n`: Seleccionar y ordenar el top n de entradas.

```
top_n(mpg,2,displ)
```

# Select

- Seleccionar variables (columnas) de un data frame.

```
select(mpg, model, displ, cyl)
```

```
## # A tibble: 234 x 3
##    model      displ   cyl
##    <chr>      <dbl> <int>
##  1 a4           1.8     4
##  2 a4           1.8     4
##  3 a4           2       4
##  4 a4           2       4
##  5 a4           2.8     6
##  6 a4           2.8     6
##  7 a4           3.1     6
##  8 a4 quattro   1.8     4
##  9 a4 quattro   1.8     4
## 10 a4 quattro   2       4
## # … with 224 more rows
```

# Select (cont.)

- Con un `-` se ignoran variables.

```
select(mpg, -manufacturer)
```

```
## # A tibble: 234 x 10
##    model   displ  year   cyl trans  drv     cty   hwy fl    class
##    <chr>   <dbl> <int> <int> <chr>  <chr> <int> <int> <chr> <chr>
##  1 a4        1.8  1999     4 auto(… f        18    29 p     comp…
##  2 a4        1.8  1999     4 manua… f        21    29 p     comp…
##  3 a4        2    2008     4 manua… f        20    31 p     comp…
##  4 a4        2    2008     4 auto(… f        21    30 p     comp…
##  5 a4        2.8  1999     6 auto(… f        16    26 p     comp…
##  6 a4        2.8  1999     6 manua… f        18    26 p     comp…
##  7 a4        3.1  2008     6 auto(… f        18    27 p     comp…
##  8 a4 qua…   1.8  1999     4 manua… 4        18    26 p     comp…
##  9 a4 qua…   1.8  1999     4 auto(… 4        16    25 p     comp…
## 10 a4 qua…   2    2008     4 manua… 4        20    28 p     comp…
## # … with 224 more rows
```

# Select (cont.)

- Puesto que las variables están ordenadas, se puede seleccionar un rango con `:`

```
select(mpg, model:trans)
```

```
## # A tibble: 234 x 5
##    model      displ  year   cyl trans
##    <chr>      <dbl> <int> <int> <chr>
##  1 a4           1.8  1999     4 auto(l5)
##  2 a4           1.8  1999     4 manual(m5)
##  3 a4           2    2008     4 manual(m6)
##  4 a4           2    2008     4 auto(av)
##  5 a4           2.8  1999     6 auto(l5)
##  6 a4           2.8  1999     6 manual(m5)
##  7 a4           3.1  2008     6 auto(av)
##  8 a4 quattro   1.8  1999     4 manual(m5)
##  9 a4 quattro   1.8  1999     4 auto(l5)
## 10 a4 quattro   2    2008     4 manual(m6)
## # … with 224 more rows
```

# Select - Funciones auxiliares

- Las siguientes funciones se pueden usar dentro de `select()` para seleccionar variables en base a sus nombres.
    - `starts_with()` : empiezan con un prefijo
    - `ends_with()` : terminan con un sufijo
    - `contains()` : contienen una string
    - `matches()` : concuerdan con una expresión regular
    - `num_range()` : rango numérico como "X01", "X02", "X03"
    - `one_of()` : selecciona columnas cuyo nombre está dentro del grupo de nombres.
- Ejemplo: seleccionar variables cuyo nomble contiene el caracter 'c'.

```
glimpse(select(mpg,contains("c")))
```

```
## Observations: 234
## Variables: 4
## $ manufacturer <chr> "audi", "audi", "audi", "audi", "audi", "…
## $ cyl          <int> 4, 4, 4, 4, 6, 6, 6, 4, 4, 4, 4, 6, 6, 6,…
## $ cty          <int> 18, 21, 20, 21, 16, 18, 18, 18, 16, 20, 1…
## $ class        <chr> "compact", "compact", "compact", "compact…
```

# arrange

- Ordena las filas de un data frame por el valor de una o varias columnas.
- Por defecto ordena de menor a mayor (orden ascendente), pero es posible utilizar la función `desc` sobre el nombre de la variable para ordenar descendentemente.

```
arrange(mpg, desc(year), cyl)
```

```
## # A tibble: 234 x 11
##    manufacturer model displ  year   cyl trans drv     cty   hwy
##    <chr>        <chr> <dbl> <int> <int> <chr> <chr> <int> <int>
##  1 audi         a4      2    2008     4 manu… f        20    31
##  2 audi         a4      2    2008     4 auto… f        21    30
##  3 audi         a4 q…   2    2008     4 manu… 4        20    28
##  4 audi         a4 q…   2    2008     4 auto… 4        19    27
##  5 chevrolet    mali…   2.4  2008     4 auto… f        22    30
##  6 honda        civic   1.8  2008     4 manu… f        26    34
##  7 honda        civic   1.8  2008     4 auto… f        25    36
##  8 honda        civic   1.8  2008     4 auto… f        24    36
##  9 honda        civic   2    2008     4 manu… f        21    29
## 10 hyundai      sona…   2.4  2008     4 auto… f        21    30
## # … with 224 more rows, and 2 more variables: fl <chr>,
## #   class <chr>
```

# mutate

- Añade nuevas variables (columnas) al data frame como combinación de las ya existentes.

```
glimpse(mutate(mpg, avg_mpg = (cty+hwy)/2))
```

```
## Observations: 234
## Variables: 12
## $ manufacturer <chr> "audi", "audi", "audi", "audi", "audi", "…
## $ model        <chr> "a4", "a4", "a4", "a4", "a4", "a4", "a4",…
## $ displ        <dbl> 1.8, 1.8, 2.0, 2.0, 2.8, 2.8, 3.1, 1.8, 1…
## $ year         <int> 1999, 1999, 2008, 2008, 1999, 1999, 2008,…
## $ cyl          <int> 4, 4, 4, 4, 6, 6, 6, 4, 4, 4, 4, 6, 6, 6,…
## $ trans        <chr> "auto(l5)", "manual(m5)", "manual(m6)", "…
## $ drv          <chr> "f", "f", "f", "f", "f", "f", "f", "4", "…
## $ cty          <int> 18, 21, 20, 21, 16, 18, 18, 18, 16, 20, 1…
## $ hwy          <int> 29, 29, 31, 30, 26, 26, 27, 26, 25, 28, 2…
## $ fl           <chr> "p", "p", "p", "p", "p", "p", "p", "p", "…
## $ class        <chr> "compact", "compact", "compact", "compact…
## $ avg_mpg      <dbl> 23.5, 25.0, 25.5, 25.5, 21.0, 22.0, 22.5,…
```

# mutate (cont.)

- Es posible crear más de una variable en la misma llamada separando cada nueva variable con `,`.

```
glimpse(mutate(mpg, avg_mpg = (cty+hwy)/2, twice_cyl=2*cyl))
```

```
## Observations: 234
## Variables: 13
## $ manufacturer <chr> "audi", "audi", "audi", "audi", "audi", "…
## $ model        <chr> "a4", "a4", "a4", "a4", "a4", "a4", "a4",…
## $ displ        <dbl> 1.8, 1.8, 2.0, 2.0, 2.8, 2.8, 3.1, 1.8, 1…
## $ year         <int> 1999, 1999, 2008, 2008, 1999, 1999, 2008,…
## $ cyl          <int> 4, 4, 4, 4, 6, 6, 6, 4, 4, 4, 4, 6, 6, 6,…
## $ trans        <chr> "auto(l5)", "manual(m5)", "manual(m6)", "…
## $ drv          <chr> "f", "f", "f", "f", "f", "f", "f", "4", "…
## $ cty          <int> 18, 21, 20, 21, 16, 18, 18, 18, 16, 20, 1…
## $ hwy          <int> 29, 29, 31, 30, 26, 26, 27, 26, 25, 28, 2…
## $ fl           <chr> "p", "p", "p", "p", "p", "p", "p", "p", "…
## $ class        <chr> "compact", "compact", "compact", "compact…
## $ avg_mpg      <dbl> 23.5, 25.0, 25.5, 25.5, 21.0, 22.0, 22.5,…
## $ twice_cyl    <dbl> 8, 8, 8, 8, 12, 12, 12, 8, 8, 8, 8, 12, 1…
```

# Otras funciones para crear nuevas variables

- `transmute`: Calcula una o más columnas nuevas y elimina las columnas originales.

```
glimpse(transmute(mpg, avg_mpg = (cty+hwy)/2))
```

```
## Observations: 234
## Variables: 1
## $ avg_mpg <dbl> 23.5, 25.0, 25.5, 25.5, 21.0, 22.0, 22.5, 22.0…
```

# Otras funciones para crear nuevas variables (cont.)

- `mutate_all`: Aplica una función a todas las columnas. Se aplica la función a cada columna individualmente.

```
# todas las columnas de tipo caracter
mutate_all(mpg,as.character)
```

- `mutate_at`: Aplica una función a un subconjunto de columnas pasadas como argumento. Se aplica la función a cada columna individualmente.

```
glimpse(mutate_at(mpg,c("displ","year","cyl","cty","hwy"),min))
```

- `mutate_if`: Aplica una función a todas las columnas que cumplen una condición lógica especificada. Se aplica la función a cada columna individualmente.

```
glimpse(mutate_if(mpg,is.numeric,min))
```

# Algunas funciones útiles para generar nuevas columnas

**vectorized function**

**OFFSETS**

dplyr::**lag()** - Offset elements by 1
dplyr::**lead()** - Offset elements by -1

**CUMULATIVE AGGREGATES**

dplyr::**cumall()** - Cumulative all()
dplyr::**cumany()** - Cumulative any()
**cummax()** - Cumulative max()
dplyr::**cummean()** - Cumulative mean()
**cummin()** - Cumulative min()
**cumprod()** - Cumulative prod()
**cumsum()** - Cumulative sum()

**RANKINGS**

dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank with ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1
dplyr::**row_number()** - rank with ties = "first"

**MATH**

**+, - , *, /, ^, %/%, %%** - arithmetic ops
**log(), log2(), log10()** - logs
**<, <=, >, >=, !=, ==** - logical comparisons

**MISC**

dplyr::**between()** - x >= left & x <= right
dplyr::**case_when()** - multi-case if_else()
dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**if_else()** - element-wise if() + else()
dplyr::**na_if()** - replace specific values with NA
**pmax()** - element-wise max()
**pmin()** - element-wise min()
dplyr::**recode()** - Vectorized switch()
dplyr::**recode_factor()** - Vectorized switch() for factors

# summarize

- Calcula variables agregadas.
- Es posible crear más de una variable en la misma llamada separando cada nueva variable con `,`.

```
summarize(mpg, max_cyl = max(cyl), avg_cty = mean(cty), min_year = min(year))
```

```
## # A tibble: 1 x 3
##   max_cyl avg_cty min_year
##     <int>   <dbl>    <int>
## 1       8    16.9     1999
```

```
# Algunas de las funciones de agregacion pueden operar sobre varias columnas
summarize(mpg, pares_distintos = n_distinct(cyl,cty), maxim = max(cyl, cty))
```

```
## # A tibble: 1 x 2
##   pares_distintos maxim
##             <int> <int>
## 1              33    35
```

# Funciones de agregación

- Las funciones más comunes para usar dentro de `summarize()` son:
    - Aritméticas: `prod()`, `sum()`
    - Centralidad: `mean()`, `median()`
    - Dispersión: `sd()`, `var()`, `mad()`
    - Rango: `max()`, `min()`, `quantile()`
    - Posición: `first()`, `last()`, `nth()`
    - Lógicas: `any()`, `all()`
    - *Conteo*: `n()`, `n_distinct()`
        - Solo se pueden usar dentro de `summarize()`
        - `n()` no recibe argumentos, `n_distinct()` el nombre de la(s) columna(s).
- Todas reducen un vector de números a un único resultado.

# Otras funciones para agregar variables

- `sumarize_all` : Aplica la función de agregación a todas las columnas. Se aplica la función a cada columna individualmente.

```
# todas las columnas de tipo caracter
summarize_all(mpg,max)
```

- `summarize_at` : Aplica la función de agragación a un subconjunto de columnas pasadas como argumento. Se aplica la función a cada columna individualmente.

```
summarize_at(mpg,c("displ","year","cyl","cty","hwy"),max)
```

- `summarize_if` : Aplica la función de agregación a todas las columnas que cumplen una condición lógica especificada. Se aplica la función a cada columna individualmente.

```
summarize_if(mpg,is.numeric,max)
```

# Concatenación de funciones

- Todas las funciónes de `dplyr` toman como primer argumento un data frame y devuelven otro data frame
- Se pueden aplicar de manera consecutiva:

```
arrange(select(filter(mpg, model == "a4"), model, year), year)

arrange(
  select(
    filter(mpg, model == "a4"),
    model, year
  ),
  year
)
```

# Concatenación de funciones (cont.)

-Otra opción:

```
df1 <- filter(mpg, model == "a4")
df2 <- select(df1, model, year)
df3 <- arrange(df2, year)
```

- Habitualmente no nos interesan los valores intermedios, solo el resultado final

# Operador "tubería" (*pipe*)

- La sintaxis es `%>%` y permite reescribir el código anterior como

```
mpg %>%
  filter(model == "a4") %>%
  select(model, year) %>%
  arrange(year)
```

- En general el codigo `df %>% foo()` es equivalente a `foo(df)`
- Esto permite concatenar funciones sin almacenar resultados intermedios y siguiendo el orden lógico

# Operaciones agrupadas

- La función `group_by()` convierte un data frame en otro agrupado por una o más variables.
- En los data frames agrupados todas las operaciones anteriores se realizan "por grupo".
- `ungroup()` elimina la agrupación.

# Slice con group_by

- Los indices son relativos al grupo.

```
mpg %>%
  group_by(cyl) %>%
  slice(1:2)
```

```
## # A tibble: 8 x 11
## # Groups:   cyl [4]
##   manufacturer model displ  year   cyl trans drv     cty   hwy
##   <chr>        <chr> <dbl> <int> <int> <chr> <chr> <int> <int>
## 1 audi         a4      1.8  1999     4 auto… f        18    29
## 2 audi         a4      1.8  1999     4 manu… f        21    29
## 3 volkswagen   jetta   2.5  2008     5 auto… f        21    29
## 4 volkswagen   jetta   2.5  2008     5 manu… f        21    29
## 5 audi         a4      2.8  1999     6 auto… f        16    26
## 6 audi         a4      2.8  1999     6 manu… f        18    26
## 7 audi         a6 q…   4.2  2008     8 auto… 4        16    23
## 8 chevrolet    c150…   5.3  2008     8 auto… r        14    20
## # … with 2 more variables: fl <chr>, class <chr>
```

# Select con group_by

- `select()` mantiene siempre las variables agrupadas, aunque no se indique explicitamente.

```
dim(mpg)
data <- mpg %>%
  group_by(cyl) %>%
  select(cty)
dim(data)
```

```
data <- mpg %>%
  group_by(cyl) %>%
  select(cty)
glimpse(data)
```

```
## Observations: 234
## Variables: 2
## Groups: cyl [4]
## $ cyl <int> 4, 4, 4, 4, 6, 6, 6, 4, 4, 4, 4, 6, 6, 6, 6, 6, 6,…
## $ cty <int> 18, 21, 20, 21, 16, 18, 18, 18, 16, 20, 19, 15, 17…
```

# arrange con group_by

- `arrange()` ordena por la(s) variable(s) especificadas como parámetros.

```
data <- mpg %>%
  group_by(cyl) %>%
  arrange(manufacturer)
glimpse(data)
```

```
## Observations: 234
## Variables: 11
## Groups: cyl [4]
## $ manufacturer <chr> "audi", "audi", "audi", "audi", "audi", "…
## $ model        <chr> "a4", "a4", "a4", "a4", "a4", "a4", "a4",…
## $ displ        <dbl> 1.8, 1.8, 2.0, 2.0, 2.8, 2.8, 3.1, 1.8, 1…
## $ year         <int> 1999, 1999, 2008, 2008, 1999, 1999, 2008,…
## $ cyl          <int> 4, 4, 4, 4, 6, 6, 6, 4, 4, 4, 4, 6, 6, 6,…
## $ trans        <chr> "auto(l5)", "manual(m5)", "manual(m6)", "…
## $ drv          <chr> "f", "f", "f", "f", "f", "f", "f", "4", "…
## $ cty          <int> 18, 21, 20, 21, 16, 18, 18, 18, 16, 20, 1…
## $ hwy          <int> 29, 29, 31, 30, 26, 26, 27, 26, 25, 28, 2…
## $ fl           <chr> "p", "p", "p", "p", "p", "p", "p", "p", "…
## $ class        <chr> "compact", "compact", "compact", "compact…
```

# summarize con group_by

Un `summarize()` sobre un data frame agrupado devuelve otro con tantas filas como grupos (valores distintos de la/s variable/s usadas para agrupar).

```
mpg %>%
  group_by(cyl) %>%
  summarize(avg_cty = mean(cty))
```

```
## # A tibble: 4 x 2
##     cyl avg_cty
##   <int>   <dbl>
## 1     4    21.0
## 2     5    20.5
## 3     6    16.2
## 4     8    12.6
```

# mutate con group_by

Un `mutate()` sobre un data frame agrupado devuelve siempre otro data frame con el mismo número de filas que el original.

```
data <- mpg %>%
  group_by(cyl) %>%
  mutate(avg_cty = mean(cty))
glimpse(data)
```

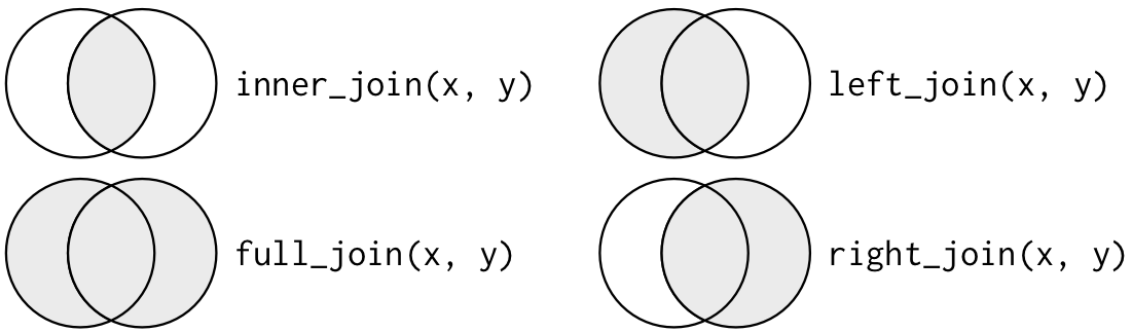
```
## Observations: 234
## Variables: 12
## Groups: cyl [4]
## $ manufacturer <chr> "audi", "audi", "audi", "audi", "audi", "…
## $ model        <chr> "a4", "a4", "a4", "a4", "a4", "a4", "a4",…
## $ displ        <dbl> 1.8, 1.8, 2.0, 2.0, 2.8, 2.8, 3.1, 1.8, 1…
## $ year         <int> 1999, 1999, 2008, 2008, 1999, 1999, 2008,…
## $ cyl          <int> 4, 4, 4, 4, 6, 6, 6, 4, 4, 4, 4, 6, 6, 6,…
## $ trans        <chr> "auto(l5)", "manual(m5)", "manual(m6)", "…
## $ drv          <chr> "f", "f", "f", "f", "f", "f", "f", "4", "…
## $ cty          <int> 18, 21, 20, 21, 16, 18, 18, 18, 16, 20, 1…
## $ hwy          <int> 29, 29, 31, 30, 26, 26, 27, 26, 25, 28, 2…
## $ fl           <chr> "p", "p", "p", "p", "p", "p", "p", "p", "…
## $ class        <chr> "compact", "compact", "compact", "compact…
## $ avg_cty      <dbl> 21.01235, 21.01235, 21.01235, 21.01235, 1…
```

# Ejercicio

1. En el dataset iris queremos saber si la media de la longitud del pétalo es muy distinta entre los distintos tipos de flor.

# joins

- La librería `dplyr` implementa funciones para unir data frames:
  - `inner_join(x,y)` : Devuelve las filas que crucen tant en x como en y.
  - `left_join(x,y)` : Devuelve todas, las filas en x y las que crucen en y (completa con NA)
  - `right_join(x,y)` : Devuelve todas las filas en y y las que crucen en x (completa con NA).
  - `full_join(x,y)` : Devuelve todas las filas de x e y (completa con NA).
  - `semi_join(x,y)` : Devuelve solo las filas de x que crucen con y (pero no y).
  - `anti_join(x,y)` : Devuelve solo las filas de x que NO crucen con y.
- Diagrama de Venn [R for Data Science]



# Equivalencia con SQL

| dplyr | SQL |
|---|---|
| `inner_join(x, y, by = "z")` | `SELECT * FROM x INNER JOIN y USING (z)` |
| `left_join(x, y, by = "z")` | `SELECT * FROM x LEFT OUTER JOIN y USING (z)` |
| `right_join(x, y, by = "z")` | `SELECT * FROM x RIGHT OUTER JOIN y USING (z)` |
| `full_join(x, y, by = "z")` | `SELECT * FROM x FULL OUTER JOIN y USING (z)` |

[R for Data Science]

# Ejemplo

```
t4a <- gather(table4a, key = "year", value = "cases", num_range("", 1999:2000))
head(t4a,4)
```

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

```
t4b <- gather(table4b, key = "YEAR", value = "population", `1999`:`2000`)
head(t4b,4)
```

```
## # A tibble: 4 x 3
##   country     YEAR  population
##   <chr>       <chr>      <int>
## 1 Afghanistan 1999    19987071
## 2 Brazil      1999   172006362
## 3 China       1999  1272915272
## 4 Afghanistan 2000    20595360
```

# Ejemplo (cont.)

```
inner_join(t4a, t4b, by=c("year" = "YEAR", "country"))
```

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

# Operaciones de conjuntos con dplyr

- `dplyr` implementa la lógica de operaciones con conjuntos sobre tibbles.
  - `intersect(x,y)` : Filas que aparecen tanto en x como en y.
  - `union(x,y)` : Filas que aparecen en x, en y, o en ambos.
  - `setdiff(x,y)` : Filas que aparecen en x, pero no en y.

```
x <- tibble(
  x1=c("A","B","C"),
  x2=1:3
)
y <- tibble(
  x1=c("B","C","D"),
  x2=2:4
)
dplyr::intersect(x,y)
dplyr::union(x,y)
dplyr::setdiff(x,y)
```

# Añadir filas y/o columnas en dplyr

- `dplyr` implementa las funciones `bind_rows` y `bind_cols` para añadir filas o columnas a un tibble, respectivamente.
- Las funciones de `dplyr` son más eficientes que las funciones `rbind` y `cbind` de R base.
- En `bin_rows` las columnas se combinan por nombre y las columnas que no están en alguno de los dataframes se rellenan con NAs.

```
# You can mix vectors and data frames:
bind_rows(
  c(a = 1, b = 2),
  tibble(saludo="hola",a = 3:4, b = 5:6),
  c(a = 7, b = 8)
)
```

```
## # A tibble: 4 x 3
##       a     b saludo
##   <dbl> <dbl> <chr>
## 1     1     2 <NA>
## 2     3     5 hola
## 3     4     6 hola
## 4     7     8 <NA>
```

# Añadir filas y/o columnas en dplyr (cont.)

- En `bind_cols` se unen las subtablas por posición -> todos los dataframes deben tener el mismo número de filas.
  - Para unir por valores, usar `join`.

```
# Both have to be tibbles
bind_cols(
  tibble(a = 3:4, b = c("a","b")),
  tibble(logical = c(T,F))
)
```

```
## # A tibble: 2 x 3
##       a b     logical
##   <int> <chr> <lgl>
## 1     3 a     TRUE
## 2     4 b     FALSE
```