

Manejo y limpieza de datos

FSC

Contents

Antes de empezar...	1
El ecosistema <i>tidyverse</i>	1
Ejemplo motivacional: Gapminder foundation	3
Paquete dplyr: <i>filter</i> y <i>select</i>	4
<i>Tidy</i> data	5
<i>gather</i> and <i>spread</i> → <i>pivot_longer</i> and <i>pivot_wider</i>	8
Si los nombres de las columnas son numericos	10
Variables dentro de los nombres de columna	12
Varias observaciones por fila	14
<i>separate</i> y <i>unite</i>	15
Ejercicios: tidy/wide	17
Sumarizando data con <i>dplyr()</i>	19
<i>summarize()</i>	19
<i>dot</i>	20
<i>group_by()</i>	22
Ordenar data.frames: <i>arrange()</i> <i>top_n()</i>	22
Ejercicios: NCHS Data	23
Tidyverse conditionals	24
<i>case_when</i>	24
<i>between</i>	25
Limpieza de datos: el paquete janitor	25
<i>tbl</i> better than table: <i>adorn_()</i> functions	25
Ejemplo 1: mtcars	26
Ejemplo 2: starwars	26
Ejemplo 3: Modelos de coches	28
Otras funciones del paquete janitor	28
<i>get_dupes()</i>	28
<i>get_dupes()</i>	29
<i>round_half_up()</i>	29
factores	29

Antes de empezar...

Los materiales de esta clase han sido preparados utilizando dos libros que están disponibles bajo licencia OUP y que os invito a explorar:

Hands-on programming with R <https://rstudio-education.github.io/hopr/>

Data Science with R <https://rafalab.github.io/dsbook/>

El ecosistema *tidyverse*

Se trata de una colección de paquetes de R diseñados por Hadley Wickham específicamente para data science. Todos los paquetes comparten la misma filosofía, diseño, gramática y estructura.

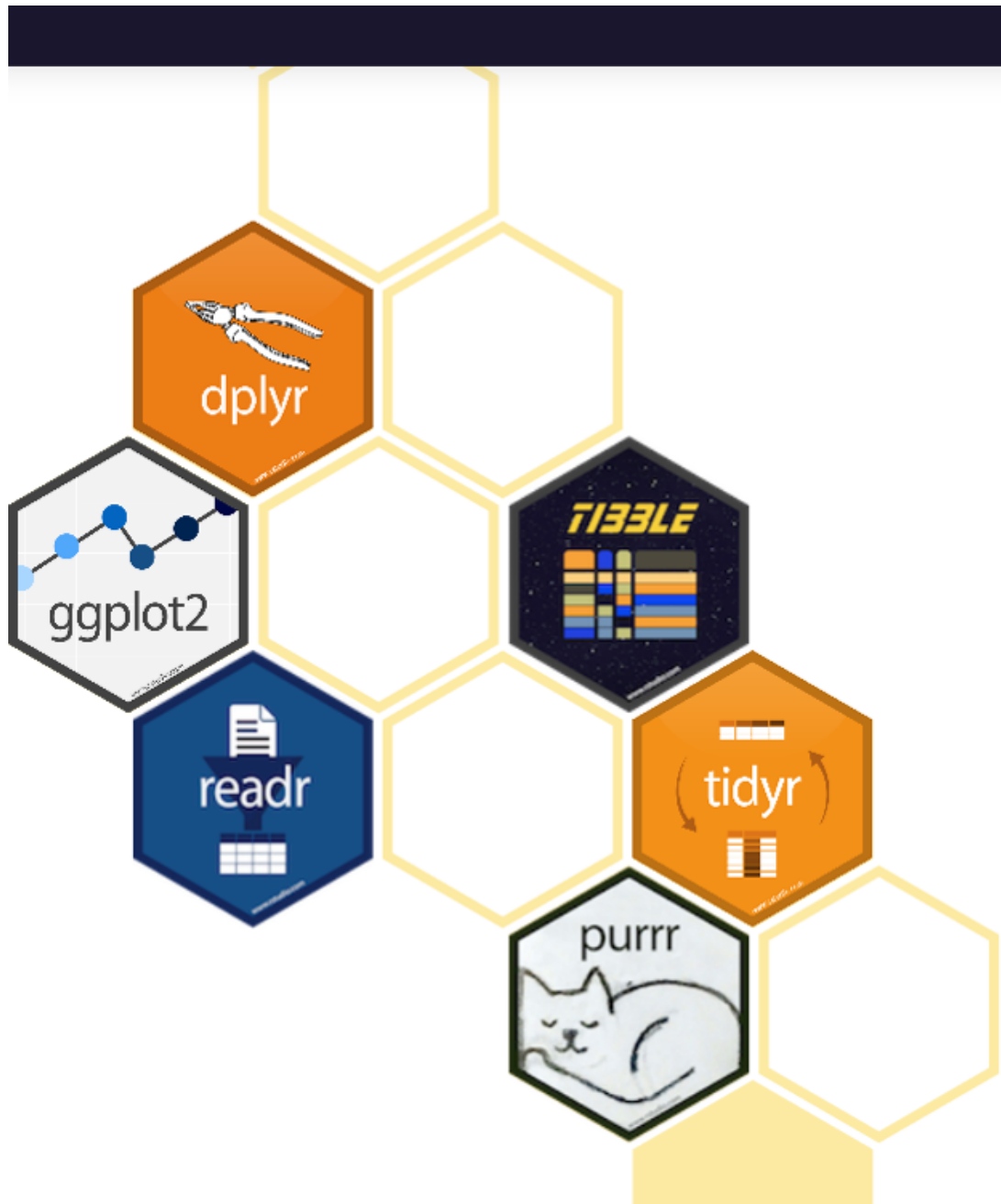


Figure 1: Tidyverse

Todos los paquetes de *tidyverse* pueden instalarse de una vez con:

```
#install.packages("tidyverse")
```

Ejemplo motivacional: Gapminder foundation

La organización Gapminder trata de desenmascarar falsos mitos acerca del estado del mundo en terminos de pobreza, desigualdad, etc a través del uso de datos. Utilizando el dataset *gapminder* del paquete *dslabs* trataremos de contestar a dos preguntas:

1. Es cierto que el mundo se divide en países occidentales ricos y no-occidentales pobres?
 2. Han aumentado las diferencias entre países en los últimos 40 años?
- Descargamos los datos:

```
library(dslabs)
data(gapminder)
#si no tienes el paquete instalado
#gapminder=read.csv("DataSets/Gapminder.csv")
```

*Exploramos los datos:

```
View(gapminder)
head(gapminder)
```

```
##           country year infant_mortality life_expectancy fertility
## 1      Albania 1960          115.40           62.87         6.19
## 2      Algeria 1960          148.20           47.50         7.65
## 3        Angola 1960          208.00           35.98         7.32
## 4 Antigua and Barbuda 1960           NA           62.97         4.43
## 5      Argentina 1960           59.87           65.39         3.11
## 6      Armenia 1960           NA           66.86         4.55
##   population      gdp continent      region
## 1   1636054         NA    Europe Southern Europe
## 2   11124892 13828152297    Africa Northern Africa
## 3    5270844         NA    Africa  Middle Africa
## 4     54681         NA  Americas    Caribbean
## 5   20619075 108322326649  Americas  South America
## 6    1867396         NA     Asia  Western Asia
```

```
str(gapminder)
```

```
## 'data.frame':   10545 obs. of  9 variables:
##  $ country      : Factor w/ 185 levels "Albania","Algeria",...: 1 2 3 4 5 6 7 8 9 10 ...
##  $ year         : int  1960 1960 1960 1960 1960 1960 1960 1960 1960 1960 ...
##  $ infant_mortality: num  115.4 148.2 208 NA 59.9 ...
##  $ life_expectancy: num  62.9 47.5 36 63 65.4 ...
##  $ fertility     : num  6.19 7.65 7.32 4.43 3.11 4.55 4.82 3.45 2.7 5.57 ...
##  $ population    : num  1636054 11124892 5270844 54681 20619075 ...
##  $ gdp           : num  NA 1.38e+10 NA NA 1.08e+11 ...
##  $ continent     : Factor w/ 5 levels "Africa","Americas",...: 4 1 1 2 2 3 2 5 4 3 ...
##  $ region        : Factor w/ 22 levels "Australia and New Zealand",...: 19 11 10 2 15 21 2 1 22 21
```

- De las siguientes parejas, cual dirías que tuvo la mayor mortalidad infantil en 2015?

++ Sri Lanka or Turkey

++ Poland or South Korea

++ Malaysia or Russia

++ Pakistan or Vietnam

++ Thailand or South Africa

Ahora, trata de responderlo usando los datos por medio de los comandos *filter* y *select*

Paquete dplyr: *filter* y *select*

filter se encarga de hacer *subsetting* de un data.frame, tibble o matriz mientras que *select* selecciona las columnas. Como cualquier otra función tidyverse el primer argumento tiene que ser el objeto que se va a manipular. A continuación con *filter* se da una condición lógica que devuelve TRUE or FALSE para cada fila. Para *select* es suficiente con dar el nombre de la columna pero también pueden proporcionarse funciones como *starts_with()*, *ends_with()* o *contain()*. Si se quiere quitar una columna basta con poner un “-” delante.

Por ejemplo, usando *filter* and *select* podemos quedarnos solo con la información de mortalidad infantil en cada pareja de países en el año 2015:

```
library(dplyr)
gapminder %>%
  filter(year == 2015 & country %in% c("Sri Lanka","Turkey")) %>%
  select(country, infant_mortality)
```

```
##      country infant_mortality
## 1 Sri Lanka             8.4
## 2   Turkey             11.6
```

```
gapminder %>%
  filter(year == 2015 & country %in% c("Poland","South Korea")) %>%
  select(country, infant_mortality)
```

```
##      country infant_mortality
## 1 South Korea             2.9
## 2   Poland              4.5
```

```
gapminder %>%
  filter(year == 2015 & country %in% c("Malaysia","Russia")) %>%
  select(country, infant_mortality)
```

```
##      country infant_mortality
## 1 Malaysia             6.0
## 2   Russia             8.2
```

```
gapminder %>%
  filter(year == 2015 & country %in% c("Pakistan","Vietnam")) %>%
  select(country, infant_mortality)
```

```
##      country infant_mortality
## 1 Pakistan            65.8
## 2  Vietnam            17.3
```

```
gapminder %>%
  filter(year == 2015 & country %in% c("Thailand","South Africa")) %>%
  select(country, infant_mortality)
```

```
##      country infant_mortality
## 1 South Africa            33.6
## 2   Thailand            10.5
```

Si queremos centrarnos en los países de una cierta región para ver por ejemplo la media pero no nos acordamos del nombre exacto de la columna:

```
gp.reduced<-gapminder %>%
  filter(year == 2015 ) %>%
  select(country, starts_with("reg"), infant_mortality)

head(gp.reduced)
```

	country	region	infant_mortality
## 1	Albania	Southern Europe	12.5
## 2	Algeria	Northern Africa	21.9
## 3	Angola	Middle Africa	96.0
## 4	Antigua and Barbuda	Caribbean	5.8
## 5	Argentina	South America	11.1
## 6	Armenia	Western Asia	12.6

Tidy data

Los datos están en formato tidy si la información de una o varias columnas constituyen una clave única para cada dato, frente a los datos de tipo “wide” para los que dicha clave viene dada por los nombres o identificadores de filas y columnas, haciendo complicado tener claves constituidas por mas de tres campos.

gapminder es un data.frame *tidy*: para cada país y año tenemos información de varios indicadores como mortalidad infantil, fertilidad, esperanza de vida, etc.

Fácilmente, gracias a que los datos están en formato *tidy* podemos usar el paquete *ggplot2* para comparar la evolución de la fertilidad y la esperanza de vida en Europa y Asia y así responder a las preguntas que teníamos:

```
library(ggplot2)
years <- c(1962, 1980, 1990, 2000, 2012)
continents <- c("Europe", "Asia")
gapminder %>%
  filter(year %in% years & continent %in% continents) %>%
  ggplot(aes(fertility, life_expectancy, col = continent)) +
  geom_point() +
  facet_wrap(~year)
```

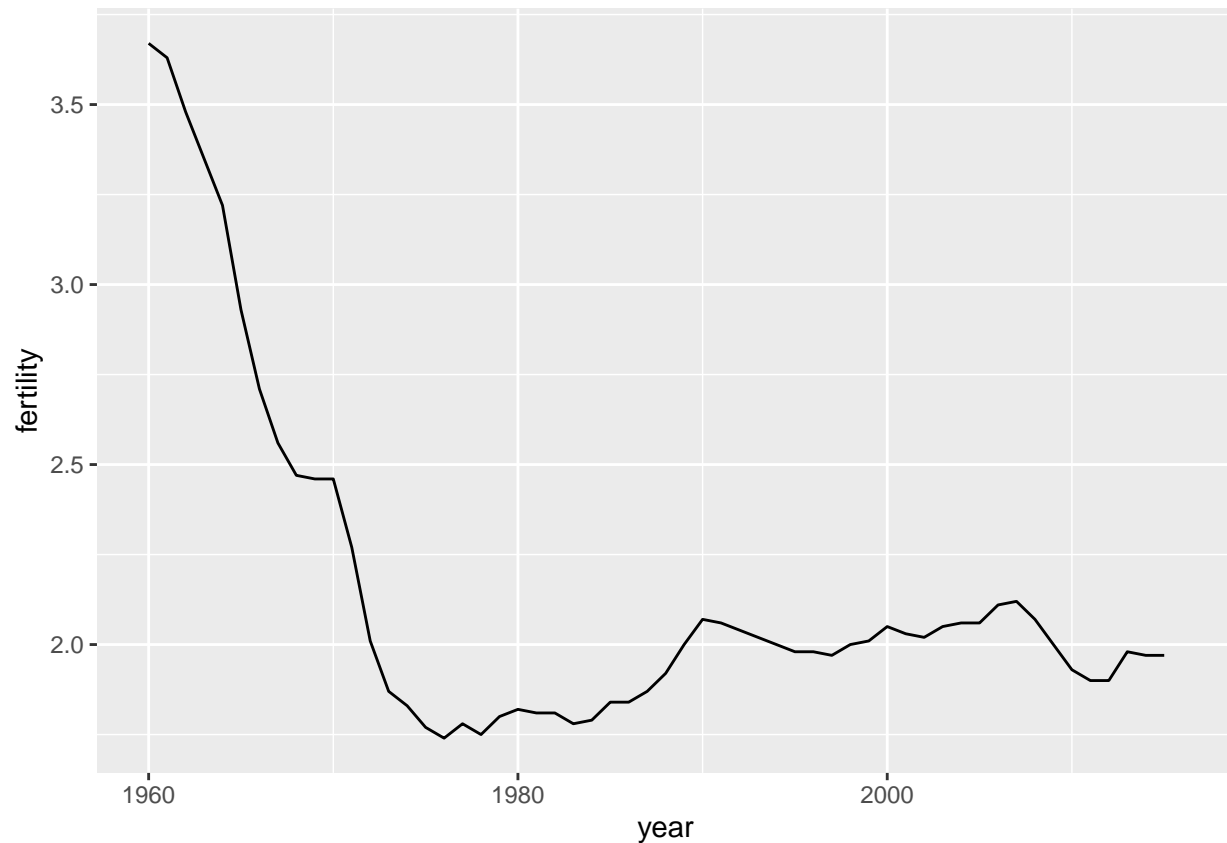


Viendo este plot parece que las diferencias entre Asia y Europa han ido desapareciendo con el paso de los años.

Centrémonos ahora sólo en el indicador de fertilidad de un país, USA:

```
gapminder %>%
  filter(country == "United States") %>%
  ggplot(aes(year,fertility)) +
  geom_line()
```

```
## Warning: Removed 1 rows containing missing values (geom_path).
```



Centrémonos en comparar Alemania y South Korea

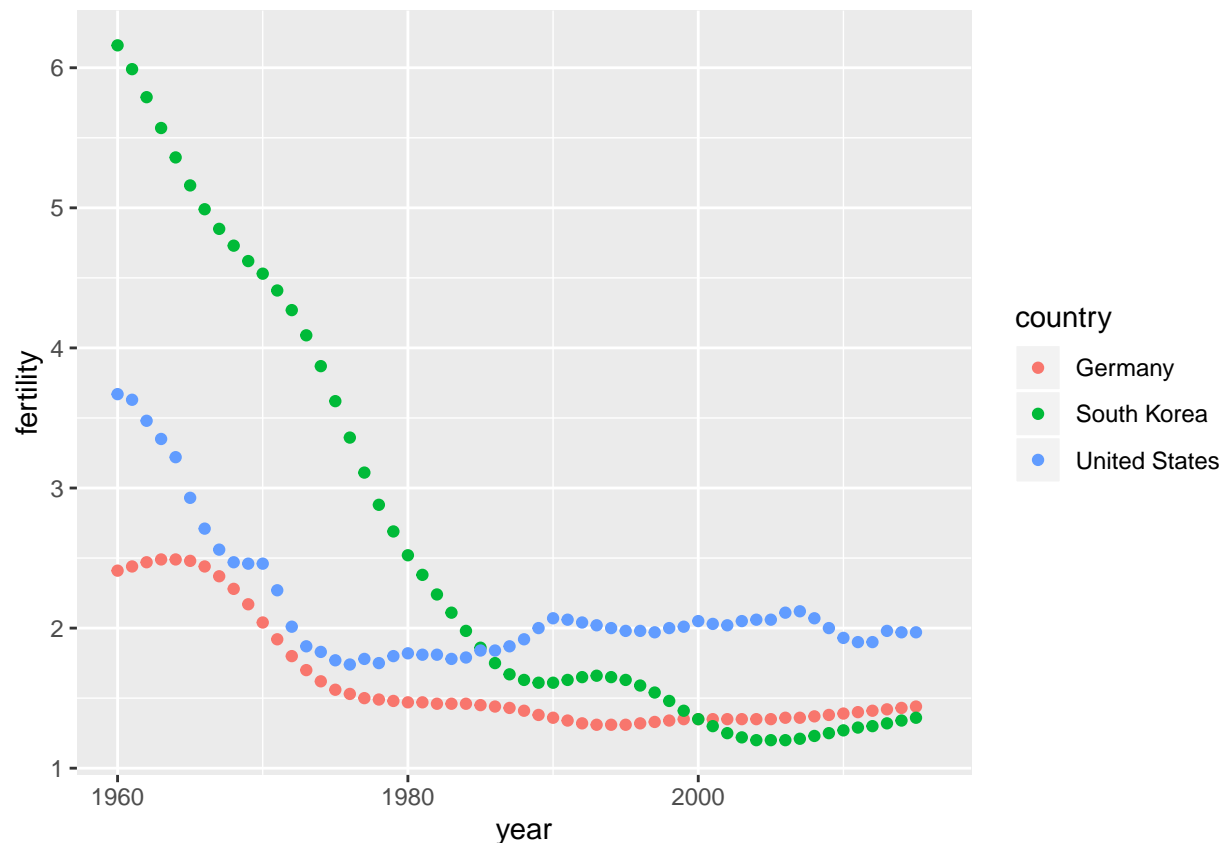
```
data("gapminder")
tidy_data <- gapminder %>%
  filter(country %in%
    c("South Korea", "Germany", "United States")) %>%
  select(country, year, fertility)

head(tidy_data)
```

```
##      country year fertility
## 1      Germany 1960      2.41
## 2  South Korea 1960      6.16
## 3  United States 1960      3.67
## 4      Germany 1961      2.44
## 5  South Korea 1961      5.99
## 6  United States 1961      3.63
```

```
tidy_data %>%
  ggplot(aes(year, fertility, color = country)) +
  geom_point()
```

```
## Warning: Removed 3 rows containing missing values (geom_point).
```



gather and *spread* → *pivot_longer* and *pivot_wider*

Todo esto ha sido posible porque *gapminder* contiene la información en un formato *tidy*. Irizarry y sus colegas han hecho un gran esfuerzo para poder tener los datos limpios y en este formato. Cómo podemos transformar datos *wide* en datos *tidy*? Utilizando la función *gather* o mejor aun, su sustituta *pivot_longer()*

Primero leemos los datos en formato *tibble*.

```
path <- system.file("extdata", package="dslabs")
filename <- file.path(path, "fertility-two-countries-example.csv")
wide_data <- read_csv(filename)
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   country = col_character()
## )
## See spec(...) for full column specifications.
```

```
wide_data
```

```
## # A tibble: 2 x 57
##   country `1960` `1961` `1962` `1963` `1964` `1965` `1966` `1967` `1968`
##   <chr>    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 Germany  2.41  2.44  2.47  2.49  2.49  2.48  2.44  2.37  2.28
## 2 South ~  6.16  5.99  5.79  5.57  5.36  5.16  4.99  4.85  4.73
## # ... with 47 more variables: `1969` <dbl>, `1970` <dbl>, `1971` <dbl>,
## #   `1972` <dbl>, `1973` <dbl>, `1974` <dbl>, `1975` <dbl>, `1976` <dbl>,
```



```
## # `1977` <dbl>, `1978` <dbl>, `1979` <dbl>, `1980` <dbl>, `1981` <dbl>,
## # `1982` <dbl>, `1983` <dbl>, `1984` <dbl>, `1985` <dbl>, `1986` <dbl>,
## # `1987` <dbl>, `1988` <dbl>, `1989` <dbl>, `1990` <dbl>, `1991` <dbl>,
## # `1992` <dbl>, `1993` <dbl>, `1994` <dbl>, `1995` <dbl>, `1996` <dbl>,
## # `1997` <dbl>, `1998` <dbl>, `1999` <dbl>, `2000` <dbl>, `2001` <dbl>,
## # `2002` <dbl>, `2003` <dbl>, `2004` <dbl>, `2005` <dbl>, `2006` <dbl>,
## # `2007` <dbl>, `2008` <dbl>, `2009` <dbl>, `2010` <dbl>, `2011` <dbl>,
## # `2012` <dbl>, `2013` <dbl>, `2014` <dbl>, `2015` <dbl>
```

```
#como ejemplo: seleccionamos las primeras 9 columnas por nombre
select(wide_data, country, `1960`:`1967`)
```

```
## # A tibble: 2 x 9
##   country    `1960` `1961` `1962` `1963` `1964` `1965` `1966` `1967`
##   <chr>      <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 Germany      2.41  2.44  2.47  2.49  2.49  2.48  2.44  2.37
## 2 South Korea  6.16  5.99  5.79  5.57  5.36  5.16  4.99  4.85
```

Las funciones *gather* y *pivot_longer()* van a “normalizar” los datos en las columnas que le digamos (1960:2015) preservando la información del resto de columnas de la matriz. Los primeros valores que le damos es el nombre del “key” y el nombre para el “value”

```
new_tidy_data <- wide_data %>%
  gather(year, fertility, -country)
new_tidy_data
```

```
## # A tibble: 112 x 3
##   country    year fertility
##   <chr>      <chr>      <dbl>
## 1 Germany    1960        2.41
## 2 South Korea 1960        6.16
## 3 Germany    1961        2.44
## 4 South Korea 1961        5.99
## 5 Germany    1962        2.47
## 6 South Korea 1962        5.79
## 7 Germany    1963        2.49
## 8 South Korea 1963        5.57
## 9 Germany    1964        2.49
## 10 South Korea 1964        5.36
## # ... with 102 more rows
```

```
class(new_tidy_data$year)
```

```
## [1] "character"
```

gather toma el tipo de las variables que recoge, y como son caracteres no les cambia el tipo a no ser que digamos:

```
new_tidy_data <- wide_data %>%
  gather(year, fertility, -country, convert = TRUE)
class(new_tidy_data$year)
```

```
## [1] "integer"
```

pivot_longer() es mas intuitiva:

```
new_tidy_data <- wide_data %>%
  pivot_longer(-country, names_to = "year", values_to = "fertility")
class(new_tidy_data$year)
```

```
## [1] "character"
```

A veces se necesita volver de tidy data a wide data usando *spread* o *pivot_wider()*

```
new_wide_data <- new_tidy_data %>%  
  spread(year, fertility)  
select(new_wide_data, country, `1960`:`1967`)
```

```
## # A tibble: 2 x 9  
##   country      `1960` `1961` `1962` `1963` `1964` `1965` `1966` `1967`  
##   <chr>         <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 Germany      2.41  2.44  2.47  2.49  2.49  2.48  2.44  2.37  
## 2 South Korea  6.16  5.99  5.79  5.57  5.36  5.16  4.99  4.85
```

```
new_wide_data <- new_tidy_data %>%  
  pivot_wider(names_from=year, values_from=fertility)  
select(new_wide_data, country, `1960`:`1967`)
```

```
## # A tibble: 2 x 9  
##   country      `1960` `1961` `1962` `1963` `1964` `1965` `1966` `1967`  
##   <chr>         <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 Germany      2.41  2.44  2.47  2.49  2.49  2.48  2.44  2.37  
## 2 South Korea  6.16  5.99  5.79  5.57  5.36  5.16  4.99  4.85
```

Si los nombres de las columnas son numericos

billboard

```
## # A tibble: 317 x 79  
##   artist track date.entered  wk1  wk2  wk3  wk4  wk5  wk6  wk7  
##   <chr> <chr> <date>      <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 2 Pac Baby~ 2000-02-26      87   82   72   77   87   94   99  
## 2 2Ge+h~ The ~ 2000-09-02      91   87   92   NA   NA   NA   NA  
## 3 3 Doo~ Kryp~ 2000-04-08      81   70   68   67   66   57   54  
## 4 3 Doo~ Loser 2000-10-21      76   76   72   69   67   65   55  
## 5 504 B~ Wobb~ 2000-04-15      57   34   25   17   17   31   36  
## 6 98~0 Give~ 2000-08-19      51   39   34   26   26   19    2  
## 7 A*Tee~ Danc~ 2000-07-08      97   97   96   95  100   NA   NA  
## 8 Aaliy~ I Do~ 2000-01-29      84   62   51   41   38   35   35  
## 9 Aaliy~ Try ~ 2000-03-18      59   53   38   28   21   18   16  
## 10 Adams~ Open~ 2000-08-26      76   76   74   69   68   67   61  
## # ... with 307 more rows, and 69 more variables: wk8 <dbl>, wk9 <dbl>,  
## # wk10 <dbl>, wk11 <dbl>, wk12 <dbl>, wk13 <dbl>, wk14 <dbl>,  
## # wk15 <dbl>, wk16 <dbl>, wk17 <dbl>, wk18 <dbl>, wk19 <dbl>,  
## # wk20 <dbl>, wk21 <dbl>, wk22 <dbl>, wk23 <dbl>, wk24 <dbl>,  
## # wk25 <dbl>, wk26 <dbl>, wk27 <dbl>, wk28 <dbl>, wk29 <dbl>,  
## # wk30 <dbl>, wk31 <dbl>, wk32 <dbl>, wk33 <dbl>, wk34 <dbl>,  
## # wk35 <dbl>, wk36 <dbl>, wk37 <dbl>, wk38 <dbl>, wk39 <dbl>,  
## # wk40 <dbl>, wk41 <dbl>, wk42 <dbl>, wk43 <dbl>, wk44 <dbl>,  
## # wk45 <dbl>, wk46 <dbl>, wk47 <dbl>, wk48 <dbl>, wk49 <dbl>,  
## # wk50 <dbl>, wk51 <dbl>, wk52 <dbl>, wk53 <dbl>, wk54 <dbl>,  
## # wk55 <dbl>, wk56 <dbl>, wk57 <dbl>, wk58 <dbl>, wk59 <dbl>,  
## # wk60 <dbl>, wk61 <dbl>, wk62 <dbl>, wk63 <dbl>, wk64 <dbl>,  
## # wk65 <dbl>, wk66 <lgl>, wk67 <lgl>, wk68 <lgl>, wk69 <lgl>,  
## # wk70 <lgl>, wk71 <lgl>, wk72 <lgl>, wk73 <lgl>, wk74 <lgl>,  
## # wk75 <lgl>, wk76 <lgl>
```

```
billboard %>%
  pivot_longer(
    cols = starts_with("wk"),
    names_to = "week",
    values_to = "rank",
    values_drop_na = TRUE
  )
```

```
## # A tibble: 5,307 x 5
##   artist track          date.entered week  rank
##   <chr>   <chr>          <date>    <chr> <dbl>
## 1 2 Pac    Baby Don't Cry (Keep... 2000-02-26 wk1     87
## 2 2 Pac    Baby Don't Cry (Keep... 2000-02-26 wk2     82
## 3 2 Pac    Baby Don't Cry (Keep... 2000-02-26 wk3     72
## 4 2 Pac    Baby Don't Cry (Keep... 2000-02-26 wk4     77
## 5 2 Pac    Baby Don't Cry (Keep... 2000-02-26 wk5     87
## 6 2 Pac    Baby Don't Cry (Keep... 2000-02-26 wk6     94
## 7 2 Pac    Baby Don't Cry (Keep... 2000-02-26 wk7     99
## 8 2Ge+her The Hardest Part Of ... 2000-09-02 wk1     91
## 9 2Ge+her The Hardest Part Of ... 2000-09-02 wk2     87
## 10 2Ge+her The Hardest Part Of ... 2000-09-02 wk3     92
## # ... with 5,297 more rows
```

Convertimos el objeto “wide” en “tidy”:

```
billboard
```

```
## # A tibble: 317 x 79
##   artist track date.entered wk1 wk2 wk3 wk4 wk5 wk6 wk7
##   <chr>   <chr> <date>    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 2 Pac    Baby~ 2000-02-26 87 82 72 77 87 94 99
## 2 2Ge+h~ The ~ 2000-09-02 91 87 92 NA NA NA NA
## 3 3 Doo~ Kryp~ 2000-04-08 81 70 68 67 66 57 54
## 4 3 Doo~ Loser 2000-10-21 76 76 72 69 67 65 55
## 5 504 B~ Wobb~ 2000-04-15 57 34 25 17 17 31 36
## 6 98~0 Give~ 2000-08-19 51 39 34 26 26 19 2
## 7 A*Tee~ Danc~ 2000-07-08 97 97 96 95 100 NA NA
## 8 Aaliy~ I Do~ 2000-01-29 84 62 51 41 38 35 35
## 9 Aaliy~ Try ~ 2000-03-18 59 53 38 28 21 18 16
## 10 Adams~ Open~ 2000-08-26 76 76 74 69 68 67 61
## # ... with 307 more rows, and 69 more variables: wk8 <dbl>, wk9 <dbl>,
## # wk10 <dbl>, wk11 <dbl>, wk12 <dbl>, wk13 <dbl>, wk14 <dbl>,
## # wk15 <dbl>, wk16 <dbl>, wk17 <dbl>, wk18 <dbl>, wk19 <dbl>,
## # wk20 <dbl>, wk21 <dbl>, wk22 <dbl>, wk23 <dbl>, wk24 <dbl>,
## # wk25 <dbl>, wk26 <dbl>, wk27 <dbl>, wk28 <dbl>, wk29 <dbl>,
## # wk30 <dbl>, wk31 <dbl>, wk32 <dbl>, wk33 <dbl>, wk34 <dbl>,
## # wk35 <dbl>, wk36 <dbl>, wk37 <dbl>, wk38 <dbl>, wk39 <dbl>,
## # wk40 <dbl>, wk41 <dbl>, wk42 <dbl>, wk43 <dbl>, wk44 <dbl>,
## # wk45 <dbl>, wk46 <dbl>, wk47 <dbl>, wk48 <dbl>, wk49 <dbl>,
## # wk50 <dbl>, wk51 <dbl>, wk52 <dbl>, wk53 <dbl>, wk54 <dbl>,
## # wk55 <dbl>, wk56 <dbl>, wk57 <dbl>, wk58 <dbl>, wk59 <dbl>,
## # wk60 <dbl>, wk61 <dbl>, wk62 <dbl>, wk63 <dbl>, wk64 <dbl>,
## # wk65 <dbl>, wk66 <lgl>, wk67 <lgl>, wk68 <lgl>, wk69 <lgl>,
## # wk70 <lgl>, wk71 <lgl>, wk72 <lgl>, wk73 <lgl>, wk74 <lgl>,
## # wk75 <lgl>, wk76 <lgl>
```

```
billboard %>%
  pivot_longer(
    cols = starts_with("wk"),
    names_to = "week",
    values_to = "rank",
    values_drop_na = TRUE
  )
```

```
## # A tibble: 5,307 x 5
##   artist track date.entered week rank
##   <chr> <chr> <date> <chr> <dbl>
## 1 2 Pac Baby Don't Cry (Keep... 2000-02-26 wk1 87
## 2 2 Pac Baby Don't Cry (Keep... 2000-02-26 wk2 82
## 3 2 Pac Baby Don't Cry (Keep... 2000-02-26 wk3 72
## 4 2 Pac Baby Don't Cry (Keep... 2000-02-26 wk4 77
## 5 2 Pac Baby Don't Cry (Keep... 2000-02-26 wk5 87
## 6 2 Pac Baby Don't Cry (Keep... 2000-02-26 wk6 94
## 7 2 Pac Baby Don't Cry (Keep... 2000-02-26 wk7 99
## 8 2Ge+her The Hardest Part Of ... 2000-09-02 wk1 91
## 9 2Ge+her The Hardest Part Of ... 2000-09-02 wk2 87
## 10 2Ge+her The Hardest Part Of ... 2000-09-02 wk3 92
## # ... with 5,297 more rows
```

Ahora queremos convertir la variable semana en numerica

```
billboard %>%
  pivot_longer(
    cols = starts_with("wk"),
    names_to = "week",
    names_prefix = "wk",
    names_ptypes = list(week = integer()),
    values_to = "rank",
    values_drop_na = TRUE,
  )
```

```
## # A tibble: 5,307 x 5
##   artist track date.entered week rank
##   <chr> <chr> <date> <int> <dbl>
## 1 2 Pac Baby Don't Cry (Keep... 2000-02-26 1 87
## 2 2 Pac Baby Don't Cry (Keep... 2000-02-26 2 82
## 3 2 Pac Baby Don't Cry (Keep... 2000-02-26 3 72
## 4 2 Pac Baby Don't Cry (Keep... 2000-02-26 4 77
## 5 2 Pac Baby Don't Cry (Keep... 2000-02-26 5 87
## 6 2 Pac Baby Don't Cry (Keep... 2000-02-26 6 94
## 7 2 Pac Baby Don't Cry (Keep... 2000-02-26 7 99
## 8 2Ge+her The Hardest Part Of ... 2000-09-02 1 91
## 9 2Ge+her The Hardest Part Of ... 2000-09-02 2 87
## 10 2Ge+her The Hardest Part Of ... 2000-09-02 3 92
## # ... with 5,297 more rows
```

Variables dentro de los nombres de columna

```
who
```

```
## # A tibble: 7,240 x 60
##   country iso2 iso3 year new_sp_m014 new_sp_m1524 new_sp_m2534
```

```
##      <chr>      <chr> <chr> <int>          <int>          <int>          <int>
## 1 Afghan~ AF      AFG      1980            NA            NA            NA
## 2 Afghan~ AF      AFG      1981            NA            NA            NA
## 3 Afghan~ AF      AFG      1982            NA            NA            NA
## 4 Afghan~ AF      AFG      1983            NA            NA            NA
## 5 Afghan~ AF      AFG      1984            NA            NA            NA
## 6 Afghan~ AF      AFG      1985            NA            NA            NA
## 7 Afghan~ AF      AFG      1986            NA            NA            NA
## 8 Afghan~ AF      AFG      1987            NA            NA            NA
## 9 Afghan~ AF      AFG      1988            NA            NA            NA
## 10 Afghan~ AF      AFG      1989            NA            NA            NA
## # ... with 7,230 more rows, and 53 more variables: new_sp_m3544 <int>,
## #   new_sp_m4554 <int>, new_sp_m5564 <int>, new_sp_m65 <int>,
## #   new_sp_f014 <int>, new_sp_f1524 <int>, new_sp_f2534 <int>,
## #   new_sp_f3544 <int>, new_sp_f4554 <int>, new_sp_f5564 <int>,
## #   new_sp_f65 <int>, new_sn_m014 <int>, new_sn_m1524 <int>,
## #   new_sn_m2534 <int>, new_sn_m3544 <int>, new_sn_m4554 <int>,
## #   new_sn_m5564 <int>, new_sn_m65 <int>, new_sn_f014 <int>,
## #   new_sn_f1524 <int>, new_sn_f2534 <int>, new_sn_f3544 <int>,
## #   new_sn_f4554 <int>, new_sn_f5564 <int>, new_sn_f65 <int>,
## #   new_ep_m014 <int>, new_ep_m1524 <int>, new_ep_m2534 <int>,
## #   new_ep_m3544 <int>, new_ep_m4554 <int>, new_ep_m5564 <int>,
## #   new_ep_m65 <int>, new_ep_f014 <int>, new_ep_f1524 <int>,
## #   new_ep_f2534 <int>, new_ep_f3544 <int>, new_ep_f4554 <int>,
## #   new_ep_f5564 <int>, new_ep_f65 <int>, newrel_m014 <int>,
## #   newrel_m1524 <int>, newrel_m2534 <int>, newrel_m3544 <int>,
## #   newrel_m4554 <int>, newrel_m5564 <int>, newrel_m65 <int>,
## #   newrel_f014 <int>, newrel_f1524 <int>, newrel_f2534 <int>,
## #   newrel_f3544 <int>, newrel_f4554 <int>, newrel_f5564 <int>,
## #   newrel_f65 <int>
```

```
who %>%
pivot_longer(
  cols = new_sp_m014:newrel_f65,
  names_to = c("diagnosis", "gender", "age"),
  names_pattern = "new_(.*)_(.)(.*)",
  values_to = "count"
)
```

```
## # A tibble: 405,440 x 8
##   country      iso2 iso3   year diagnosis gender age   count
##   <chr>        <chr> <chr> <int> <chr>      <chr> <chr> <int>
## 1 Afghanistan AF      AFG   1980 sp        m      014     NA
## 2 Afghanistan AF      AFG   1980 sp        m     1524     NA
## 3 Afghanistan AF      AFG   1980 sp        m     2534     NA
## 4 Afghanistan AF      AFG   1980 sp        m     3544     NA
## 5 Afghanistan AF      AFG   1980 sp        m     4554     NA
## 6 Afghanistan AF      AFG   1980 sp        m     5564     NA
## 7 Afghanistan AF      AFG   1980 sp        m      65     NA
## 8 Afghanistan AF      AFG   1980 sp        f      014     NA
## 9 Afghanistan AF      AFG   1980 sp        f     1524     NA
## 10 Afghanistan AF      AFG   1980 sp        f     2534     NA
## # ... with 405,430 more rows
```

```
## # A tibble: 405,440 x 8
##   country    iso2 iso3   year diagnosis gender age   count
##   <chr>      <chr> <chr> <int> <chr>      <fct> <ord> <int>
## 1 Afghanistan AF    AFG   1980 sp         m     014    NA
## 2 Afghanistan AF    AFG   1980 sp         m    1524    NA
## 3 Afghanistan AF    AFG   1980 sp         m    2534    NA
## 4 Afghanistan AF    AFG   1980 sp         m    3544    NA
## 5 Afghanistan AF    AFG   1980 sp         m    4554    NA
## 6 Afghanistan AF    AFG   1980 sp         m    5564    NA
## 7 Afghanistan AF    AFG   1980 sp         m     65    NA
## 8 Afghanistan AF    AFG   1980 sp         f     014    NA
## 9 Afghanistan AF    AFG   1980 sp         f    1524    NA
## 10 Afghanistan AF    AFG   1980 sp         f    2534    NA
## # ... with 405,430 more rows
```

```
View(anscombe)
anscombe %>%
  pivot_longer(everything(),
    names_to = c(".value", "set"),
    names_pattern = "(.)(.)"
  )
```

separate y unite

Lo que hemos hecho hasta ahora es relativamente sencillo comparado con lo que solemos necesitar. ¿Qué sucede si el nombre de las columnas que queremos “gather” no son muy claros como para usarlos como clave?

```
path <- system.file("extdata", package = "dslabs")
filename <- file.path(path, "life-expectancy-and-fertility-two-countries-example.csv")

raw_dat <- read_csv(filename)
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   country = col_character()
## )

## See spec(...) for full column specifications.
```

```
select(raw_dat, 1:5)
```

```
## # A tibble: 2 x 5
##   country `1960_fertility` `1960_life_expe~` `1961_fertility`
##   <chr>      <dbl>          <dbl>          <dbl>
## 1 Germany      2.41            69.3            2.44
## 2 South ~      6.16            53.0            5.99
## # ... with 1 more variable: `1961_life_expectancy` <dbl>
```

lo transformamos en data tidy

```
dat <- raw_dat %>% gather(key, value, -country)
head(dat)
```

```
## # A tibble: 6 x 3
##   country    key          value
##   <chr>      <chr>      <dbl>
## 1 Germany  1960_fertility      2.41
## 2 South Korea 1960_fertility      6.16
## 3 Germany  1960_life_expectancy 69.3
## 4 South Korea 1960_life_expectancy 53.0
## 5 Germany  1961_fertility      2.44
## 6 South Korea 1961_fertility      5.99
```

pero tenemos dos observaciones (año&variable) en cada fila

```
head(dat %>% separate(key, c("year", "variable_name"), "_"))
```

```
## Warning: Expected 2 pieces. Additional pieces discarded in 112 rows [3, 4,
## 7, 8, 11, 12, 15, 16, 19, 20, 23, 24, 27, 28, 31, 32, 35, 36, 39, 40, ...].
```

```
## # A tibble: 6 x 4
##   country    year variable_name value
##   <chr>      <chr> <chr>          <dbl>
## 1 Germany  1960  fertility      2.41
## 2 South Korea 1960  fertility      6.16
## 3 Germany  1960   life        69.3
## 4 South Korea 1960   life        53.0
## 5 Germany  1961  fertility      2.44
## 6 South Korea 1961  fertility      5.99
```

"_" es el separador por defecto y por tanto no hace falta

```
head(dat %>% separate(key, c("year", "variable_name")))
```

```
## Warning: Expected 2 pieces. Additional pieces discarded in 112 rows [3, 4,
## 7, 8, 11, 12, 15, 16, 19, 20, 23, 24, 27, 28, 31, 32, 35, 36, 39, 40, ...].
```

```
## # A tibble: 6 x 4
##   country    year variable_name value
##   <chr>      <chr> <chr>          <dbl>
## 1 Germany    1960 fertility      2.41
## 2 South Korea 1960 fertility      6.16
## 3 Germany    1960 life          69.3
## 4 South Korea 1960 life          53.0
## 5 Germany    1961 fertility      2.44
## 6 South Korea 1961 fertility      5.99
```

```
head(dat %>% separate(key, c("year", "first_variable_name", "second_variable_name"),
  fill = "right"))
```

```
## # A tibble: 6 x 5
##   country    year first_variable_name second_variable_name value
##   <chr>      <chr> <chr>                <chr>          <dbl>
## 1 Germany    1960 fertility          <NA>          2.41
## 2 South Korea 1960 fertility          <NA>          6.16
## 3 Germany    1960 life            expectancy    69.3
## 4 South Korea 1960 life            expectancy    53.0
## 5 Germany    1961 fertility          <NA>          2.44
## 6 South Korea 1961 fertility          <NA>          5.99
```

```
head(dat %>% separate(key, c("year", "variable_name"), extra = "merge"))
```

```
## # A tibble: 6 x 4
##   country    year variable_name    value
##   <chr>      <chr> <chr>          <dbl>
## 1 Germany    1960 fertility      2.41
## 2 South Korea 1960 fertility      6.16
## 3 Germany    1960 life_expectancy 69.3
## 4 South Korea 1960 life_expectancy 53.0
## 5 Germany    1961 fertility      2.44
## 6 South Korea 1961 fertility      5.99
```

#pero queremos crear una columna para cada variable

```
dat %>%
  separate(key, c("year", "variable_name"), extra = "merge") %>%
  spread(variable_name, value)
```

```
## # A tibble: 112 x 4
##   country year fertility life_expectancy
##   <chr>   <chr>    <dbl>      <dbl>
## 1 Germany 1960      2.41      69.3
## 2 Germany 1961      2.44      69.8
## 3 Germany 1962      2.47      70.0
## 4 Germany 1963      2.49      70.1
## 5 Germany 1964      2.49      70.7
## 6 Germany 1965      2.48      70.6
## 7 Germany 1966      2.44      70.8
## 8 Germany 1967      2.37      71.0
## 9 Germany 1968      2.28      70.6
## 10 Germany 1969      2.17      70.5
## # ... with 102 more rows
```


A veces queremos hacer lo contrario: juntar dos columnas en una.

```
dat %>%
  separate(key, c("year", "first_variable_name", "second_variable_name"), fill = "right")
```

```
## # A tibble: 224 x 5
##   country    year first_variable_name second_variable_name value
##   <chr>      <chr> <chr>                <chr>                <dbl>
## 1 Germany   1960 fertility            <NA>                  2.41
## 2 South Korea 1960 fertility            <NA>                  6.16
## 3 Germany   1960 life               expectancy            69.3
## 4 South Korea 1960 life               expectancy            53.0
## 5 Germany   1961 fertility            <NA>                  2.44
## 6 South Korea 1961 fertility            <NA>                  5.99
## 7 Germany   1961 life               expectancy            69.8
## 8 South Korea 1961 life               expectancy            53.8
## 9 Germany   1962 fertility            <NA>                  2.47
## 10 South Korea 1962 fertility            <NA>                  5.79
## # ... with 214 more rows
```

Llegamos a lo mismo si primero unimos la primera y segunda columna y después expandimos las columnas y renombramos fertility_NA a fertility:

```
dat %>%
  separate(key, c("year", "first_variable_name", "second_variable_name"), fill = "right") %>%
  unite(variable_name, first_variable_name, second_variable_name, sep="_") %>%
  spread(variable_name, value) %>%
  rename(fertility = fertility_NA)
```

```
## # A tibble: 112 x 4
##   country year fertility life_expectancy
##   <chr>   <chr>    <dbl>         <dbl>
## 1 Germany 1960      2.41         69.3
## 2 Germany 1961      2.44         69.8
## 3 Germany 1962      2.47         70.0
## 4 Germany 1963      2.49         70.1
## 5 Germany 1964      2.49         70.7
## 6 Germany 1965      2.48         70.6
## 7 Germany 1966      2.44         70.8
## 8 Germany 1967      2.37         71.0
## 9 Germany 1968      2.28         70.6
## 10 Germany 1969      2.17         70.5
## # ... with 102 more rows
```

Ejercicios: tidy/wide

1. Niveles de CO2.

Definimos los datos de CO2 en formato wide:

```
##? co2
co2_wide <- data.frame(matrix(co2, ncol = 12, byrow = TRUE)) %>%
  setNames(1:12) %>%
  mutate(year = as.character(1959:1997))
```

Utiliza la función gather para reordenar este objeto en formato *tidy*. Llama CO2 a la columna con las mediciones de co2 y mes a la columna de los meses. El objeto se llamará co2_tidy.

No es capaz de hacer el plot porque month no es un numérico. Rehaz el objeto asegurandote de que month es numérico.

Que nos dice este plot?

- Los niveles de CO2 aumentan monotonicamente de 1959 a 1997.
- Los niveles de CO2 son mas altos en verano y la media anual aumento de 1959 a 1997.
- Los niveles de CO2 son constantes y variabilidad aleatoria es lo que explica las diferencias.
- Los niveles de CO2 no tienen una tendencia estacional.

2. Porcentaje de admision de hombres y mujeres Utilizando los datos del paquete dslabs:

```
admissions
```

```
##      major gender admitted applicants
## 1      A   men        62         825
## 2      B   men        63         560
## 3      C   men        37         325
## 4      D   men        33         417
## 5      E   men        28         191
## 6      F   men         6         373
## 7      A women       82         108
## 8      B women       68          25
## 9      C women       34         593
## 10     D women       35         375
## 11     E women       24         393
## 12     F women         7         341
```

- Transformalo en tidy usando spread, una fila para cada major.
- Usa gather para crear un tmp data.frame con una columna que contenga el tipo de información (applicant/admitted).
- Usa unite para crear la columna column_name i que contenga la informacion de admitted_men, admitted_women, applicants_men and applicants_women
- Usa spread para generar los datos tidy con cuatro variables para cada major
- Usa %>% para escribir una sola linea de codigo que convierta admissions en la tabla tidy final.

3. Tibble family: convierte los datos en tidy

```
family <- tribble(
  ~family, ~dob_child1, ~dob_child2, ~gender_child1, ~gender_child2,
    1L, "1998-11-26", "2000-01-29", 1L, 2L,
    2L, "1996-06-22", NA, 2L, NA,
    3L, "2002-07-11", "2004-04-05", 2L, 2L,
    4L, "2004-10-10", "2009-08-27", 1L, 1L,
    5L, "2000-12-05", "2005-02-28", 2L, 1L,
)
family
```

```
## # A tibble: 5 x 5
##   family dob_child1 dob_child2 gender_child1 gender_child2
##   <int> <chr>      <chr>          <int>      <int>
## 1     1 1998-11-26 2000-01-29         1          2
## 2     2 1996-06-22 <NA>             2         NA
## 3     3 2002-07-11 2004-04-05         2          2
## 4     4 2004-10-10 2009-08-27         1          1
```

```
## 5      5 2000-12-05 2005-02-28      2      1
family <- family %>% mutate_at(vars(starts_with("dob")), parse_date)
family
```

```
## # A tibble: 5 x 5
##   family dob_child1 dob_child2 gender_child1 gender_child2
##   <int> <date>      <date>      <int>      <int>
## 1     1 1998-11-26 2000-01-29      1          2
## 2     2 1996-06-22 NA              2         NA
## 3     3 2002-07-11 2004-04-05      2          2
## 4     4 2004-10-10 2009-08-27      1          1
## 5     5 2000-12-05 2005-02-28      2          1
```

```
## # A tibble: 9 x 4
##   family child  dob      gender
##   <int> <chr>  <date>    <int>
## 1     1  child1 1998-11-26      1
## 2     1  child2 2000-01-29      2
## 3     2  child1 1996-06-22      2
## 4     3  child1 2002-07-11      2
## 5     3  child2 2004-04-05      2
## 6     4  child1 2004-10-10      1
## 7     4  child2 2009-08-27      1
## 8     5  child1 2000-12-05      2
## 9     5  child2 2005-02-28      1
```

Sumarizando data con *dplyr()*

summarize()

Vamos a utilizar los datos de alturas del paquete dslabs

```
library(dslabs)
data(heights)
head(heights)
```

```
##      sex height
## 1  Male     75
## 2  Male     70
## 3  Male     68
## 4  Male     74
## 5  Male     61
## 6 Female    65
```

```
str(heights)
```

```
## 'data.frame':  1050 obs. of  2 variables:
##  $ sex      : Factor w/ 2 levels "Female","Male": 2 2 2 2 2 1 1 1 1 2 ...
##  $ height: num  75 70 68 74 61 65 66 62 66 67 ...
```

La función *summarize()* del paquete *dplyr* nos calcula cualquier agregado que le pidamos de un vector de un data.frame o de un tibble. Como el input era un data.frame() el output también lo es.

```
heights %>%
  filter(sex == "Female") %>%
  summarize(
    average = mean(height),
```

```
standard_deviation = sd(height)
)
```

```
##      average standard_deviation
## 1 64.93942          3.760656
```

Se pueden utilizar medidas mas robustas:

```
s <- heights %>%
  summarize(
    median = median(height),
    mad=mad(height),
    min=min(height),
    max=max(height))
```

```
s
```

```
##      median      mad min      max
## 1    68.5 3.7065  50 82.67717
```

```
str(s)
```

```
## 'data.frame':  1 obs. of  4 variables:
## $ median: num 68.5
## $ mad   : num 3.71
## $ min   : num 50
## $ max   : num 82.7
```

NOTA: con la función *summarize* solo podemos llamar funciones que devuelvan un solo valor.

dot

Recordemos en el último ejercicio de la sesión II habiamos descargado la tabla con el rate de asesinatos en todo el mundo.

```
url="https://en.wikipedia.org/wiki/List_of_countries_by_intentional_homicide_rate"
h <- read_html(url)
class(h)
```

```
## [1] "xml_document" "xml_node"
```

```
h
```

```
## {html_document}
## <html class="client-nojs" lang="en" dir="ltr">
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset= ...
## [2] <body class="mediawiki ltr sitedir-ltr mw-hide-empty-elt ns-0 ns-sub ...
```

```
tab <- h %>% html_nodes("table")
tab <- tab[[4]] %>% html_table
head(tab)
```

```
##      Country (or dependent territory,subnational area, etc.) Region
## 1                                     Burundi Africa
## 2                                     Comoros Africa
## 3                                     Djibouti Africa
## 4                                     Eritrea Africa
## 5                                     Ethiopia Africa
## 6                                     Kenya Africa
##      Subregion Rate Count Yearlisted      Source
## 1 Eastern Africa 6.02   635      2016    CTS/SDG
```

```
## 2 Eastern Africa 7.70    60      2015 WHO Estimate
## 3 Eastern Africa 6.48    60      2015 WHO Estimate
## 4 Eastern Africa 8.04   390      2015 WHO Estimate
## 5 Eastern Africa 7.56 7,552      2015 WHO Estimate
## 6 Eastern Africa 5.00 2,466      2017          CTS
```

```
class(tab)
```

```
## [1] "data.frame"
```

```
tab <- tab %>%
  select(starts_with("Country"),
         Region, Count, Rate, starts_with("Year")) %>%
  setNames(c("country", "continent", "total", "murder_rate", "year"))
```

```
head(tab)
```

```
##   country continent total murder_rate year
## 1  Burundi   Africa   635         6.02 2016
## 2  Comoros   Africa    60         7.70 2015
## 3 Djibouti   Africa    60         6.48 2015
## 4  Eritrea   Africa   390         8.04 2015
## 5 Ethiopia   Africa 7,552         7.56 2015
## 6   Kenya   Africa 2,466         5.00 2017
```

Para comparar USA (nuestros datos por estados en murders) necesitabamos primero hacer la media de todos los estados, calculando el ratio de asesinatos por 100.000 habitantes:

```
s <- murders %>%
  mutate(rate=total/population*100000) %>%
  summarize(mean(rate))
```

```
s
```

```
##   mean(rate)
## 1    2.779125
```

```
str(s)
```

```
## 'data.frame':   1 obs. of  1 variable:
##  $ mean(rate): num 2.78
```

Como las funciones de dplyr devuelven el mismo tipo de objeto que su input en este caso queremos acceder sólo al valor que tienen almacenado. Podemos hacerlo así:

```
s
```

```
##   mean(rate)
## 1    2.779125
```

```
s %>% .$rate
```

```
## NULL
```

“.” simplemente reemplaza al objeto que pasamos por el pipe, en este caso s que es n data frame. Por eso accedemos su información con \$

```
s <- murders %>%
  summarize(rate=mean(total/population*100000)) %>%
  .$rate
```

```
s
```

```
## [1] 2.779125
```

```
s2<-murders %>%
  summarize(total=sum(total)) %>%
  .$total
```

group_by()

Construimos un objeto que contenga la informacion que teniamos de USA en el objeto tab, quedandonos solo con medidas del año 2015:

```
tab.USA<-tab %>%
  filter(year=="2015") %>%
  select(-year) %>%
  add_row(country = "USA",
          continent = "Americas",
          total = s2,
          murder_rate=s)
```

```
tab.USA %>%
  group_by(continent) %>%
  summarize(mean.rate=mean(murder_rate)) %>% arrange(desc(mean.rate))
```

```
## # A tibble: 5 x 2
##   continent mean.rate
##   <chr>         <dbl>
## 1 Americas      14.3
## 2 Africa         8.55
## 3 Asia           3.51
## 4 Oceania        3.4
## 5 Europe         0.235
```

```
tab.USA %>% filter(continent=="Americas") %>% arrange(murder_rate)
```

```
##           country continent total murder_rate
## 1              USA  Americas  9403    2.779125
## 2      Paraguay  Americas   617    9.290000
## 3 Trinidad and Tobago Americas   420   30.880000
```

Ordenar data.frames: arrange() top_n()

¿Cual es el estado con mayor poblacion? La función *arrange()* ordena tablas enteras por una variable

```
murders %>% arrange(population) %>% head()
```

```
##           state abb      region population total
## 1      Wyoming  WY      West      563626      5
## 2 District of Columbia DC      South      601723     99
## 3      Vermont  VT      Northeast      625741      2
## 4 North Dakota ND North Central      672591      4
## 5      Alaska  AK      West      710231     19
## 6 South Dakota SD North Central      814180      8
```

¿Y con menor numero de asesinatos?

```
murders %>% arrange(total) %>% head()
```

```
##           state abb      region population total
## 1      Vermont  VT      Northeast      625741      2
## 2 North Dakota ND North Central      672591      4
```

```
## 3 New Hampshire NH Northeast 1316470 5
## 4 Wyoming WY West 563626 5
## 5 Hawaii HI West 1360301 7
## 6 South Dakota SD North Central 814180 8
```

¿Y en por cien mil habitantes?

```
murders %>% mutate(rate=total/population*100000)%>%
  arrange(rate) %>%
  head()
```

```
##      state abb      region population total      rate
## 1  Vermont VT Northeast 625741 2 0.3196211
## 2 New Hampshire NH Northeast 1316470 5 0.3798036
## 3 Hawaii HI West 1360301 7 0.5145920
## 4 North Dakota ND North Central 672591 4 0.5947151
## 5 Iowa IA North Central 3046355 21 0.6893484
## 6 Idaho ID West 1567582 12 0.7655102
```

Si tenemos empates podemos usar una segunda columna para deshacer dicho empate:

```
murders %>% mutate(rate=total/population*100000)%>%
  arrange(total,rate) %>%
  head()
```

```
##      state abb      region population total      rate
## 1  Vermont VT Northeast 625741 2 0.3196211
## 2 North Dakota ND North Central 672591 4 0.5947151
## 3 New Hampshire NH Northeast 1316470 5 0.3798036
## 4 Wyoming WY West 563626 5 0.8871131
## 5 Hawaii HI West 1360301 7 0.5145920
## 6 South Dakota SD North Central 814180 8 0.9825837
```

¿Y el mayor? Por último podemos seleccionar las primeras filas de un data.frame o de un tibble usando la función `top_n()`. Nota que la función `desc()` indica que se ordena de manera descendente el data.frame.

```
murders %>% mutate(rate=total/population*100000)%>%
  arrange(desc(rate)) %>%
  top_n(10)
```

Selecting by rate

```
##      state abb      region population total      rate
## 1 District of Columbia DC South 601723 99 16.452753
## 2 Louisiana LA South 4533372 351 7.742581
## 3 Missouri MO North Central 5988927 321 5.359892
## 4 Maryland MD South 5773552 293 5.074866
## 5 South Carolina SC South 4625364 207 4.475323
## 6 Delaware DE South 897934 38 4.231937
## 7 Michigan MI North Central 9883640 413 4.178622
## 8 Mississippi MS South 2967297 120 4.044085
## 9 Georgia GA South 9920000 376 3.790323
## 10 Arizona AZ West 6392017 232 3.629527
```

Ejercicios: NCHS Data

El National Center for Health Statistics ha realizado encuestas de hábitos de vida y salud desde 1960. Desde 1999 5000 individuos han sido entrevistados cada año junto con una exploración médica. Parte de estos datos están en el paquete **NHANES**

```
library(NHANES)
data(NHANES)
```

Los datos en el paquete **NHANES** contienen una gran cantidad de valores perdidos. Las principales funciones de sumariación de R devuelven un NA si hay NA en los datos.

```
data(na_example)
mean(na_example)
```

```
## [1] NA
```

```
sd(na_example)
```

```
## [1] NA
```

Para ignorar los valores perdidos usamos el argumento `na.rm`:

```
mean(na_example, na.rm = TRUE)
```

```
## [1] 2.301754
```

```
sd(na_example, na.rm = TRUE)
```

```
## [1] 1.22338
```

Vamos a explorar los datos en **NHANES**

1. Tensión sanguínea: Seleccionemos a las mujeres entre 20 y 29 años. **AgeDecade** es una variable categórica que contiene las edades. La categoría es " 20-29", con un espacio delante! Cual es su media? (**BPSysAve** variable). Guardalo en una variable llamada **ref**.

Hint: Usa `filter` y `summarize` y luego usa `na.rm = TRUE` al calcular su meda y desviacion estandard. Tambien podrias quitar los missing usando `filter`.

2. Usando pipe asigna ese valor a una variable llamada **ref_avg**. Hint: Use the code similar to above and then `pull`.
3. min y max para cada grupo de edad dentro de las mujeres
4. Media y desviacion estandar para cada grupo de edad. Hint: filtra por genero y luego usa `group_by`.
5. Hacer lo mismo para los hombres
6. Usando `group_by(AgeDecade, Gender)`, repite 4 y 5 con una sola linea de código.
7. Para los hombres entre 40-49 compara su presion sistolica para las distintas razas reportadas en la variable **Race1** y ordenalas de mayor a menor.

Tidyverse conditionals

`case_when`

```
x <- c(-2, -1, 0, 1, 2)
case_when(x < 0 ~ "Negative",
          x > 0 ~ "Positive",
          TRUE ~ "Zero")
```

```
## [1] "Negative" "Negative" "Zero"      "Positive" "Positive"
```

Imaginemos que queremos comparar los rates de asesinatos en tres regiones: *New England*, *West Coast*, *South*, y *other*. Para cada estado necesitamos preguntar si es de cada uno de ellos y si no pasar al siguiente:


```
data(murders)
murders %>%
  mutate(group = case_when(
    abb %in% c("ME", "NH", "VT", "MA", "RI", "CT") ~ "New England",
    abb %in% c("WA", "OR", "CA") ~ "West Coast",
    region == "South" ~ "South",
    TRUE ~ "other")) %>%
  group_by(group) %>%
  summarize(rate = sum(total) / sum(population) * 10^5) %>%
  arrange(rate)
```

```
## # A tibble: 4 x 2
##   group      rate
##   <chr>    <dbl>
## 1 New England 1.72
## 2 other      2.71
## 3 West Coast 2.90
## 4 South     3.63
```

between

```
x >= a & x <= b
```

Utilizando tidyverse:

```
between(x, a, b)
```

Limpieza de datos: el paquete janitor

```
library(janitor)
test_df <- as.data.frame(matrix(ncol = 6))
names(test_df) <- c("firstName", "ábc@!*", "% successful (2009)",
  "REPEAT VALUE", "REPEAT VALUE", "")
test_df %>%
  clean_names()
```

```
##   first_name abc percent_successful_2009 repeat_value repeat_value_2 x
## 1      NA  NA              NA              NA              NA  NA
```

```
make.names(names(test_df))
```

```
## [1] "firstName"      "ábc..."      "X..successful..2009."
## [4] "REPEAT.VALUE"   "REPEAT.VALUE"   "X"
```

tabyl better than table: adorn__() functions

tabyl es la alternativa en el universo tidyverse a table. Es un tipo de objeto que vais a usar mucho como data scientists. Cuenta combinaciones de 1,2,3 variables de una forma mas eficiente que table. Una vez creada la tabla podemos formatearla como queramos usando adorn_* functions

Problemas de table:

- No acepta data.frame inputs y no combina bien con %>%
- No produce data.frames

- Su resultado es difícil de formatear

`tabyl()` que forma parte del paquete *janitor* soluciona estos problemas. Además funciona con vectores y no cuenta los NAs.

```
x <- c("big", "big", "small", "small", "small", NA)
tabyl(x)
```

```
##      x n   percent valid_percent
##    big 2 0.3333333          0.4
##  small 3 0.5000000          0.6
##   <NA> 1 0.1666667           NA
```

Ejemplo 1: mtcars

```
mtcars %>%
  tabyl(gear, cyl) %>%
  adorn_totals("col") %>%
  adorn_percentages("row") %>%
  adorn_pct_formatting(digits = 2) %>%
  adorn_ns() %>%
  adorn_title()
```

```
##           cyl
## gear      4      6      8      Total
##   3 6.67% (1) 13.33% (2) 80.00% (12) 100.00% (15)
##   4 66.67% (8) 33.33% (4)  0.00% (0) 100.00% (12)
##   5 40.00% (2) 20.00% (1) 40.00% (2) 100.00% (5)
```

`adorn()` se puede llamar en todo tipo de objetos, no solo en un `tabyl()`

```
mtcars %>% adorn_totals("col") %>% adorn_percentages("col") %>% head()
```

```
##   mpg      cyl      disp      hp      drat      wt      qsec
## 21.0 0.03030303 0.02167111 0.02343417 0.03388652 0.02544875 0.02881854
## 21.0 0.03030303 0.02167111 0.02343417 0.03388652 0.02792564 0.02979901
## 22.8 0.02020202 0.01462800 0.01981253 0.03345208 0.02253477 0.03258281
## 21.4 0.03030303 0.03494467 0.02343417 0.02676166 0.03122815 0.03403600
## 18.7 0.04040404 0.04876001 0.03728164 0.02736988 0.03341363 0.02979901
## 18.1 0.03030303 0.03047500 0.02236898 0.02398123 0.03360789 0.03540164
##           vs      am      gear      carb      Total
## 0.00000000 0.07692308 0.03389831 0.04444444 0.02315761
## 0.00000000 0.07692308 0.03389831 0.04444444 0.02321889
## 0.07142857 0.07692308 0.03389831 0.01111111 0.01780394
## 0.07142857 0.00000000 0.02542373 0.01111111 0.03043280
## 0.00000000 0.00000000 0.02542373 0.02222222 0.04298045
## 0.07142857 0.00000000 0.02542373 0.01111111 0.02762852
```

Ejemplo 2: starwars

```
humans <- starwars %>%
  filter(species == "Human")

t1 <- humans %>%
  tabyl(eye_color)
```

```
t1 %>%
  adorn_totals("row") %>%
  adorn_pct_formatting()
```

```
## eye_color n percent
##      blue 12  34.3%
## blue-gray 1   2.9%
##      brown 17  48.6%
##      dark 1   2.9%
##      hazel 2   5.7%
##      yellow 2   5.7%
##      Total 35 100.0%
```

Se pueden ademas producir tablas de contingencia con tabyl

```
t2 <- humans %>%
  tabyl(gender, eye_color)

t2 %>%
  adorn_percentages("row") %>%
  adorn_pct_formatting(digits = 2) %>%
  adorn_ns()
```

```
## gender      blue blue-gray      brown      dark      hazel      yellow
## female 33.33% (3) 0.00% (0) 55.56% (5) 0.00% (0) 11.11% (1) 0.00% (0)
##      male 34.62% (9) 3.85% (1) 46.15% (12) 3.85% (1) 3.85% (1) 7.69% (2)
```

Tablas con 3 variables tabyl

```
t3 <- humans %>%
  tabyl(eye_color, skin_color, gender)
```

```
humans %>%
  tabyl(eye_color, skin_color, gender, show_missing_levels = FALSE) %>%
  adorn_totals("row") %>%
  adorn_percentages("all") %>%
  adorn_pct_formatting(digits = 1) %>%
  adorn_ns %>%
  adorn_title
```

```
## $female
##      skin_color
## eye_color      fair      light
##      blue 22.2% (2) 11.1% (1)
##      brown 11.1% (1) 44.4% (4)
##      hazel 0.0% (0) 11.1% (1)
##      Total 33.3% (3) 66.7% (6)
##
## $male
##      skin_color
## eye_color      dark      fair      light      pale      tan      white
##      blue 0.0% (0) 26.9% (7) 7.7% (2) 0.0% (0) 0.0% (0) 0.0% (0)
## blue-gray 0.0% (0) 3.8% (1) 0.0% (0) 0.0% (0) 0.0% (0) 0.0% (0)
##      brown 11.5% (3) 15.4% (4) 11.5% (3) 0.0% (0) 7.7% (2) 0.0% (0)
##      dark 3.8% (1) 0.0% (0) 0.0% (0) 0.0% (0) 0.0% (0) 0.0% (0)
##      hazel 0.0% (0) 3.8% (1) 0.0% (0) 0.0% (0) 0.0% (0) 0.0% (0)
```

```
##      yellow  0.0% (0)  0.0% (0)  0.0% (0)  3.8% (1)  0.0% (0)  3.8% (1)
##      Total  15.4% (4) 50.0% (13) 19.2% (5)  3.8% (1)  7.7% (2)  3.8% (1)
```

Se pueden presentar tablas de manera incluso mas elegante usando kable del paquete knitr

```
humans %>%
  tabyl(gender, eye_color) %>%
  adorn_totals(c("row", "col")) %>%
  adorn_percentages("row") %>%
  adorn_pct_formatting(rounding = "half up", digits = 0) %>%
  adorn_ns() %>%
  adorn_title("combined") %>%
  knitr::kable()
```

gender/eye_color	blue	blue-gray	brown	dark	hazel	yellow	Total
female	33% (3)	0% (0)	56% (5)	0% (0)	11% (1)	0% (0)	100% (9)
male	35% (9)	4% (1)	46% (12)	4% (1)	4% (1)	8% (2)	100% (26)
Total	34% (12)	3% (1)	49% (17)	3% (1)	6% (2)	6% (2)	100% (35)

Ejemplo 3: Modelos de coches

```
mpg_by_cyl_and_am <- mtcars %>%
  group_by(cyl, am) %>%
  summarise(mpg = mean(mpg)) %>%
  spread(am, mpg)
```

```
mpg_by_cyl_and_am
```

```
## # A tibble: 3 x 3
## # Groups:   cyl [3]
##   cyl   `0`   `1`
##   <dbl> <dbl> <dbl>
## 1     4  22.9  28.1
## 2     6  19.1  20.6
## 3     8  15.0  15.4
```

```
mpg_by_cyl_and_am %>%
  adorn_rounding() %>%
  adorn_ns(
    ns = mtcars %>% # calculate the Ns on the fly by calling tabyl on the original data
      tabyl(cyl, am)
  ) %>%
  adorn_title("combined", row_name = "Cylinders", col_name = "Is Automatic")
```

```
##   Cylinders/Is Automatic      0      1
## 1                   4 22.9 (3) 28.1 (8)
## 2                   6 19.1 (4) 20.6 (3)
## 3                   8 15.1 (12) 15.4 (2)
```

Otras funciones del paquete janitor

```
get_dupes()
```

```
get_dupes(mtcars, wt, cyl)
```

```
## # A tibble: 4 x 12
##   wt    cyl dupe_count  mpg  disp    hp  drat  qsec    vs    am  gear
##   <dbl> <dbl>    <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1  3.44     6         2  19.2  168.   123  3.92  18.3     1     0     4
## 2  3.44     6         2  17.8  168.   123  3.92  18.9     1     0     4
## 3  3.57     8         2  14.3  360    245  3.21  15.8     0     0     3
## 4  3.57     8         2   15    301    335  3.54  14.6     0     1     5
## # ... with 1 more variable: carb <dbl>
```

get_dupes()

```
# remove_empty() rows and columns

q <- data.frame(v1 = c(1, NA, 3),
                v2 = c(NA, NA, NA),
                v3 = c("a", NA, "b"))
q %>%
  remove_empty(c("rows", "cols"))
```

```
##   v1 v3
## 1  1  a
## 3  3  b
```

round_half_up()

```
nums <- c(2.5, 3.5)
round(nums)
```

```
## [1] 2 4
```

```
round_half_up(nums)
```

```
## [1] 3 4
```

factores

```
# Count factor levels in groups of high, medium, and low with top_levels()

f <- factor(
  c("strongly agree", "agree", "neutral", "neutral", "disagree", "strongly agree"),
  levels = c("strongly agree", "agree", "neutral", "disagree", "strongly disagree"))
top_levels(f)
```

```
##           f n  percent
##   strongly agree, agree 3 0.5000000
##           neutral 2 0.3333333
##   disagree, strongly disagree 1 0.1666667
```

```
top_levels(f, n = 1)
```

```
##           f n  percent
##   strongly agree 2 0.3333333
##   agree, neutral, disagree 4 0.6666667
##   strongly disagree 0 0.0000000
```