

Introducción al análisis de datos con R

Introducción a la limpieza y manipulación de datos

Manuel Mejías Leiva

Universidad de Valladolid | manuel.mejias@uva.es

5 - 9 junio de 2023

**Primeros pasos: cargar librerías, definir directorio
de trabajo e importar datos**

Librerías y datos

Cargamos las librerías

```
library(tidyverse)
```

Definimos el directorio de trabajo

```
setwd("~/Desktop/Intro-R/Dia2")
```

Importamos los datos

```
ecv19 <- read_csv("ecv19.csv")
```

```
## Rows: 39852 Columns: 13
## — Column specification ——————
## Delimiter: ","
## chr (7): sexo, nacionalidad, ccaa, tipo_hogar, regimen_vivienda, nivel_educa...
## dbl (6): edad, unidades_consumo, renta_hogar, ingresos_anuales, riesgo_pobre...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

¿Qué aspecto tienen? 🔎

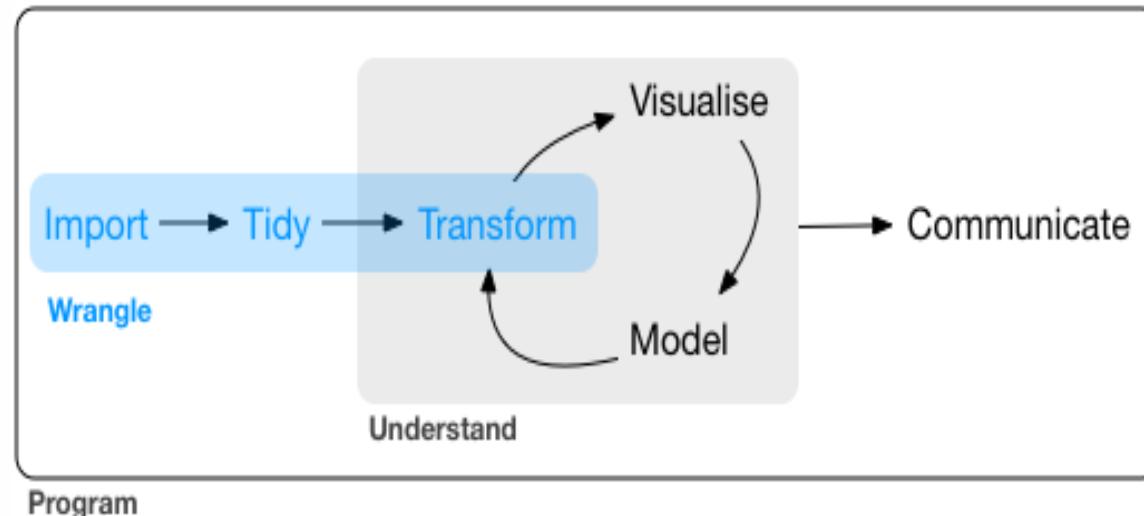
```
glimpse(ecv19)
```

```
## Rows: 39,852
## Columns: 13
## $ sexo <chr> "Hombre", "Mujer", "Mujer", "Hombre", "Mu...
## $ edad <dbl> 70, 68, 72, 60, 54, 26, 23, 61, 22, 78, 4...
## $ nacionalidad <chr> "Espana", "Espana", "Espana", "Espana", "...
## $ ccaa <chr> "ES21", "ES21", "ES21", "ES21", "ES21", "...
## $ tipo_hogar <chr> "2 adultos sin niños", "2 adultos sin niñ...
## $ regimen_vivienda <chr> "En propiedad", "En propiedad", "En propi...
## $ nivel_educativo <chr> "300", "300", "200", "300", "500", "500", ...
## $ unidades_consumo <dbl> 1.5, 1.5, 1.5, 1.5, 2.0, 2.0, 2.0, 1.5, 1...
## $ renta_hogar <dbl> 31626.70, 31626.70, 42250.00, 42250.00, 7...
## $ ingresos_anuales <dbl> 0.00, 0.00, 0.00, 41600.00, 0.00, 4680.00...
## $ riesgo_pobreza <dbl> 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, ...
## $ situacion_economica_infancia <chr> NA, NA, NA, NA, "Moderadamente buena", "M...
## $ pesos <dbl> 906.1583, 906.1583, 1227.6887, 1227.6887, ...
```

El flujo de trabajo de la ciencia de datos

Procesamiento de datos

- La gestión de datos suele ser la parte de cualquier proyecto de análisis de datos que requiere más tiempo.
- El proceso de tratamiento de datos incluye su importación, ordenación y transformación.
- La gestión de datos es la manipulación o combinación de conjuntos de datos con fines de análisis, y a menudo hay que aclarar y repetir este proceso a medida que cambia la comprensión de los datos y también cambian las necesidades de modelización y visualización.



Depurando, ordenando y resumiendo datos con `{dplyr}`

Empecemos a limpiar datos



Estas son las cinco funciones clave de **dplyr** que permiten resolver la gran mayoría de los desafíos de manipulación de datos:

- Filtrar o elegir las observaciones por sus valores (**filter()** --- del inglés filtrar).
- Seleccionar las variables por sus nombres (**select()** --- del inglés seleccionar).
- Crear nuevas variables con transformaciones de variables existentes (**mutate()** --- del inglés mutar o transformar).
- Contraer muchos valores en un solo resumen (**summarise()** --- del inglés resumir).
- Reordenar las filas (**arrange()** --- del inglés organizar).

Antes de empezar, el pipe %>%

%>% puede leerse como "y a continuación", y nos **permite encadenar múltiples operaciones en nuestro código.**

Así escribimos código más simple, legible y con menos repeticiones.

```
# Filtrar los datos  
ecv_filtrado <- filter(ecv19,  
                           renta_hogar > 100000)
```

```
# Contar número de hogares  
conteo_hogares <- count(ecv_filtrado,  
                           tipo_hogar)
```

```
# Resultado final  
conteo_hogares
```

```
## # A tibble: 6 × 2  
##   tipo_hogar      n  
##   <chr>        <int>  
## 1 2 adultos sin niños    88  
## 2 Biparental con niños/as 254  
## 3 Monomarental con niños/as    7  
## 4 Otros hogares       260  
## 5 Unipersonal         5  
## 6 <NA>                  4
```

```
ecv19 %>%  
  filter(renta_hogar > 100000) %>%  
  count(tipo_hogar)
```

```
## # A tibble: 6 × 2  
##   tipo_hogar      n  
##   <chr>        <int>  
## 1 2 adultos sin niños    88  
## 2 Biparental con niños/as 254  
## 3 Monomarental con niños/as    7  
## 4 Otros hogares       260  
## 5 Unipersonal         5  
## 6 <NA>                  4
```

FILTRAR registros: filter()

dplyr::filter()
KEEP ROWS THAT
s.a.t.i.s.f.y
your CONDITIONS

keep rows from... this data... ONLY IF... type is "otter" AND site is "bay"
filter(df, type == "otter" & site == "bay")



type	food	site
otter	urchin	bay
Shark	seal	channel
otter	abalone	bay
otter	crab	wharf

@allison_horst

FILTRAR registros: filter()

Una de las **operaciones más comunes** es **filtrar registros** en base a alguna **condición lógica**: con **filter()** se seleccionarán solo individuos que cumplan ciertas condiciones.

```
ecv19 %>%  
  filter(sexo == "Mujer")
```

```
## # A tibble: 20,484 × 13  
##   sexo   edad nacionalidad ccaa tipo_hogar    regimen_vivienda nivel_educativo  
##   <chr> <dbl> <chr>      <chr> <chr>          <chr>                <chr>  
## 1 Mujer     68 Espana       ES21  2 adultos si... En propiedad      300  
## 2 Mujer     72 Espana       ES21  2 adultos si... En propiedad      200  
## 3 Mujer     54 Espana       ES21  Biparental c... En propiedad      500  
## 4 Mujer     26 Espana       ES21  Biparental c... En propiedad      500  
## 5 Mujer     23 Espana       ES21  Biparental c... En propiedad      344  
## 6 Mujer     61 Espana       ES21  Monomarental... En alquiler      500  
## 7 Mujer     78 Espana       ES21  2 adultos si... En propiedad      100  
## 8 Mujer     48 Espana       ES21  2 adultos si... En propiedad      500  
## 9 Mujer     52 Espana       ES21  Biparental c... En propiedad      500  
## 10 Mujer    14 <NA>        ES21  Biparental c... En propiedad <NA>  
## # i 20,474 more rows  
## # i 6 more variables: unidades_consumo <dbl>, renta_hogar <dbl>,  
## #   ingresos_anuales <dbl>, riesgo_pobreza <dbl>,
```

FILTRAR registros: filter()

Comparadores habituales:

- `==`, `!=` igual/distinto que
- `>`, `<` mayor/menor que
- `>=`, `<=` mayor/menor o igual que
- `%in%` los valores pertenecen a un listado
- `!is.na()` los valores no son ausentes (mejor usar `drop_na()`)
- `between(variable, val1, val2)`: si los valores (normalmente continuos) están dentro de un rango.

```
ecv19 %>%  
  filter(renta_hogar > 10000)
```

```
## # A tibble: 36,470 × 13  
##   sexo     edad nacionalidad    ccaa tipo_hogar regimen_vivienda nivel_educativo  
##   <chr>    <dbl> <chr>        <chr> <chr>           <chr>             <chr>  
## 1 Hombre      70 Espana       ES21  2 adultos... En propiedad 300  
## 2 Mujer       68 Espana       ES21  2 adultos... En propiedad 300  
## 3 Mujer       72 Espana       ES21  2 adultos... En propiedad 200  
## 4 Hombre      60 Espana       ES21  2 adultos... En propiedad 300  
## 5 Mujer       61 Espana       ES21 Monomaren... En alquiler  500  
## 6 Hombre      22 Extranjero (r... ES21 Monomaren... En alquiler 344
```

FILTRAR registros: filter()

En el caso de utilizar `%in%` dentro de `filter()`, se busca que los valores de una columna estén presentes en un conjunto de valores especificados.

```
ecv19 %>%  
  filter(edad %in% c(25:45))
```

```
## # A tibble: 9,464 × 13  
##   sexo     edad nacionalidad    ccaa tipo_hogar regimen_vivienda nivel_educativo  
##   <chr>    <dbl> <chr>        <chr> <chr>          <chr>                <chr>  
## 1 Mujer      26 Espana       ES21 Biparental... En propiedad 500  
## 2 Hombre     36 Extranjero (r... ES22 Biparental... En propiedad 500  
## 3 Mujer      30 Espana       ES22 Biparental... En propiedad 500  
## 4 Mujer      37 Espana       ES21 Unipersonal... En propiedad 500  
## 5 Mujer      40 Espana       ES21 2 adultos... En propiedad 300  
## 6 Hombre     39 Espana       ES21 2 adultos... En propiedad 300  
## 7 Hombre     43 Espana       ES21 2 adultos... En propiedad 500  
## 8 Hombre     40 Espana       ES21 Unipersonal... En propiedad 500  
## 9 Mujer      41 Espana       ES21 Biparental... En propiedad 200  
## 10 Hombre    42 Espana       ES21 Biparental... En propiedad 000  
## # i 9,454 more rows  
## # i 6 more variables: unidades_consumo <dbl>, renta hogar <dbl>,  
## #   ingresos_anuales <dbl>, riesgo_pobreza <dbl>,
```

FILTRAR registros: filter()

El operador **!** se coloca delante de una condición lógica para negarla. Esto significa que se seleccionarán las filas que no cumplan con la condición negada.

```
ecv19 %>%  
  filter(regimen_vivienda != "En propiedad")
```

```
## # A tibble: 8,644 × 13  
##   sexo     edad nacionalidad    ccaa tipo_hogar regimen_vivienda nivel_educativo  
##   <chr>    <dbl> <chr>        <chr> <chr>       <chr>           <chr>  
## 1 Mujer      61 Espana        ES21 Monomaren... En alquiler      500  
## 2 Hombre     22 Extranjero (r... ES21 Monomaren... En alquiler      344  
## 3 Mujer      43 Espana        ES21 Monomaren... En alquiler      500  
## 4 Mujer      11 <NA>          ES21 Monomaren... En alquiler    <NA>  
## 5 Hombre     56 Extranjero (r... ES21 Otros hog... En alquiler      500  
## 6 Mujer      51 Extranjero (r... ES21 Otros hog... En alquiler      300  
## 7 Hombre     30 Extranjero (r... ES21 Otros hog... En alquiler      344  
## 8 Hombre     66 Espana         ES21 Uniperson... En alquiler     100  
## 9 Hombre     38 Espana         ES42 Uniperson... En alquiler      500  
## 10 Hombre    45 Extranjero (r... ES42 2 adultos... En alquiler     100  
## # i 8,634 more rows  
## # i 6 more variables: unidades_consumo <dbl>, rentahogar <dbl>,  
## #   ingresos_anuales <dbl>, riesgo_pobreza <dbl>,
```

FILTRAR registros: filter()

Cuando es una variable continua el interés podría estar en comprobar si la variable toma valores **dentro de un intervalo continuo**.

```
ecv19 %>%  
  filter(between(edad, 25, 35))
```

```
## # A tibble: 3,897 × 13  
##   sexo     edad nacionalidad    ccaa tipo_hogar regimen_vivienda nivel_educativo  
##   <chr>   <dbl> <chr>        <chr> <chr>       <chr>           <chr>  
## 1 Mujer     26 Espana        ES21 Biparental... En propiedad 500  
## 2 Mujer     30 Espana        ES22 Biparental... En propiedad 500  
## 3 Hombre    35 Espana        ES21 Biparental... En propiedad 200  
## 4 Mujer     33 Espana        ES21 Biparental... En propiedad 500  
## 5 Mujer     26 Espana        ES21 Otros hogar... En propiedad 500  
## 6 Mujer     30 Espana        ES21 Otros hogar... En propiedad 354  
## 7 Hombre    35 Espana        ES21 2 adultos... En propiedad 344  
## 8 Hombre    30 Extranjero (r... ES21 Otros hogar... En alquiler 344  
## 9 Mujer     31 Espana        ES42 Otros hogar... En propiedad 500  
## 10 Mujer    31 Espana       ES42 Otros hogar... En propiedad 344  
## # i 3,887 more rows  
## # i 6 more variables: unidades_consumo <dbl>, renta_hogar <dbl>,  
## #   ingresos_anuales <dbl>, riesgo_pobreza <dbl>,
```

FILTRAR registros: filter()

Operadores lógicos: `&` y `|` (o)

```
ecv19 %>%  
  filter(edad > 30 | edad < 50)
```

```
## # A tibble: 39,852 × 13  
##   sexo     edad nacionalidad    ccaa tipo_hogar regimen_vivienda nivel_educativo  
##   <chr>   <dbl> <chr>        <chr> <chr>          <chr>           <chr>  
## 1 Hombre     70 Espana       ES21  2 adultos... En propiedad 300  
## 2 Mujer      68 Espana       ES21  2 adultos... En propiedad 300  
## 3 Mujer      72 Espana       ES21  2 adultos... En propiedad 200  
## 4 Hombre      60 Espana       ES21  2 adultos... En propiedad 300  
## 5 Mujer      54 Espana       ES21  Biparental... En propiedad 500  
## 6 Mujer      26 Espana       ES21  Biparental... En propiedad 500  
## 7 Mujer      23 Espana       ES21  Biparental... En propiedad 344  
## 8 Mujer      61 Espana       ES21  Monomarental... En alquiler 500  
## 9 Hombre      22 Extranjero (r... ES21  Monomarental... En alquiler 344  
## 10 Mujer     78 Espana       ES21  2 adultos... En propiedad 100  
## # i 39,842 more rows  
## # i 6 more variables: unidades_consumo <dbl>, renta_hogar <dbl>,  
## #   ingresos_anuales <dbl>, riesgo_pobreza <dbl>,  
## #   situacion_economica_infancia <chr>, pesos <dbl>
```

FILTRAR registros: filter()

Además, podemos crear un nuevo data frame a partir de las condiciones que le facilitemos a **filter()**. Por ejemplo:

```
filtered1 <- ecv19 %>%  
  filter((sexo == "Hombre" & edad %in% c(30:50)) |  
         (sexo == "Mujer" & edad %in% c(55:65)))  
filtered1
```

```
## # A tibble: 8,710 × 13  
##   sexo     edad nacionalidad    ccaa tipo_hogar regimen_vivienda nivel_educativo  
##   <chr>   <dbl> <chr>        <chr> <chr>          <chr>           <chr>  
## 1 Mujer     61 Espana       ES21 Monomaren... En alquiler      500  
## 2 Hombre    50 Espana       ES21 Biparenta... En propiedad  300  
## 3 Hombre    50 Espana       ES21 Otros hog... En propiedad  500  
## 4 Hombre    36 Extranjero (r... ES22 Biparenta... En propiedad  500  
## 5 Hombre    48 Espana       ES21 Biparenta... En propiedad  200  
## 6 Hombre    39 Espana       ES21 2 adultos... En propiedad  300  
## 7 Hombre    43 Espana       ES21 2 adultos... En propiedad  500  
## 8 Hombre    40 Espana       ES21 Uniperson... En propiedad  500  
## 9 Hombre    42 Espana       ES21 Biparenta... En propiedad  000  
## 10 Hombre   35 Espana       ES21 Biparenta... En propiedad  200  
## # i 8,700 more rows
```

FILTRAR datos ausentes

Podemos también **filtrar los registros ausentes** en alguna de sus variables con `drop_na()`. Si no especificamos, elimina todos los registros que tenga alguno de sus campos ausente.

```
ecv19 %>% drop_na()
```

Podemos indicarle que nos elimine filas con datos ausentes fijándonos solo en **alguna variable particular**.

```
ecv19 %>% drop_na(sexo, edad, ingresos_anuales)
```

```
## # A tibble: 33,376 × 13
##   sexo     edad nacionalidad    ccaa tipo_hogar regimen_vivienda nivel_educativo
##   <chr>   <dbl> <chr>        <chr> <chr>          <chr>              <chr>
## 1 Hombre     70 Espana       ES21  2 adultos... En propiedad 300
## 2 Mujer      68 Espana       ES21  2 adultos... En propiedad 300
## 3 Mujer      72 Espana       ES21  2 adultos... En propiedad 200
## 4 Hombre     60 Espana       ES21  2 adultos... En propiedad 300
## 5 Mujer      54 Espana       ES21 Biparental... En propiedad 500
## 6 Mujer      26 Espana       ES21 Biparental... En propiedad 500
## 7 Mujer      23 Espana       ES21 Biparental... En propiedad 344
## 8 Mujer      61 Espana       ES21 Monomaren... En alquiler  500
## 9 Hombre     22 Espana       ES21 Monomaren... En alquiler  244
## 10 Mujer     33 Espana      ...  ...           ...             ...
```

FILTRAR datos ausentes

`!is.na()` también se puede utilizar para filtrar un dataframe excluyendo los valores ausentes (NA) en una columna específica.

```
ecv19 %>%  
  filter(!is.na(nivel_educativo))
```

```
## # A tibble: 33,187 × 13  
##   sexo     edad nacionalidad    ccaa tipo_hogar regimen_vivienda nivel_educativo  
##   <chr>    <dbl> <chr>        <chr> <chr>          <chr>            <chr>  
## 1 Hombre     70 Espana       ES21  2 adultos... En propiedad 300  
## 2 Mujer      68 Espana       ES21  2 adultos... En propiedad 300  
## 3 Mujer      72 Espana       ES21  2 adultos... En propiedad 200  
## 4 Hombre      60 Espana       ES21  2 adultos... En propiedad 300  
## 5 Mujer      54 Espana       ES21 Biparental... En propiedad 500  
## 6 Mujer      26 Espana       ES21 Biparental... En propiedad 500  
## 7 Mujer      23 Espana       ES21 Biparental... En propiedad 344  
## 8 Mujer      61 Espana       ES21 Monomaren... En alquiler  500  
## 9 Hombre      22 Extranjero (r... ES21 Monomaren... En alquiler  344  
## 10 Mujer     78 Espana       ES21  2 adultos... En propiedad 100  
## # i 33,177 more rows  
## # i 6 more variables: unidades_consumo <dbl>, renta_hogar <dbl>,  
## #   ingresos_anuales <dbl>, riesgo_pobreza <dbl>,
```

FILTRAR registros: slice()

Normalmente filtraremos registros por alguna condición pero no siempre, a veces nos puede interesar, por ejemplo, sacar las primeras n filas. Para podemos crear **rebanadas de los datos**, seleccionando filas por su posición con `slice()`.

```
ecv19 %>% slice(1)
```

```
## # A tibble: 1 × 13
##   sexo     edad nacionalidad ccaa tipo_hogar    regimen_vivienda nivel_educativo
##   <chr>    <dbl> <chr>        <chr> <chr>          <chr>                  <chr>
## 1 Hombre     70 Espana       ES21  2 adultos si... En propiedad      300
## # i 6 more variables: unidades_consumo <dbl>, renta_hogar <dbl>,
## #   ingresos_anuales <dbl>, riesgo_pobreza <dbl>,
## #   situacion_economica_infancia <chr>, pesos <dbl>
```

FILTRAR registros: slice()

Recuerda que también podemos **extraer varias rebanadas** a la vez.

```
# filas de la 1 a la 5  
ecv19 %>% slice(1:5)
```

```
## # A tibble: 5 × 13  
##   sexo     edad nacionalidad ccaa tipo_hogar    regimen_vivienda nivel_educativo  
##   <chr>    <dbl> <chr>        <chr> <chr>          <chr>                  <chr>  
## 1 Hombre     70 Espana       ES21  2 adultos si... En propiedad      300  
## 2 Mujer      68 Espana       ES21  2 adultos si... En propiedad      300  
## 3 Mujer      72 Espana       ES21  2 adultos si... En propiedad      200  
## 4 Hombre     60 Espana       ES21  2 adultos si... En propiedad      300  
## 5 Mujer      54 Espana       ES21  Biparental c... En propiedad      500  
## # i 6 more variables: unidades_consumo <dbl>, renta hogar <dbl>,  
## #   ingresos_anuales <dbl>, riesgo_pobreza <dbl>,  
## #   situacion_economica_infancia <chr>, pesos <dbl>
```

FILTRAR registros: slice()

También podríamos usar una **secuencia de índices** a extraer.

```
# filas 1, 2, 10, 13, 27  
ecv19 %>% slice(c(1, 2, 10, 13, 27))
```

```
## # A tibble: 5 × 13  
##   sexo     edad nacionalidad ccaa tipo_hogar    regimen_vivienda nivel_educativo  
##   <chr>    <dbl> <chr>        <chr> <chr>          <chr>                  <chr>  
## 1 Hombre     70 Espana       ES21  2 adultos si... En propiedad      300  
## 2 Mujer      68 Espana       ES21  2 adultos si... En propiedad      300  
## 3 Mujer      78 Espana       ES21  2 adultos si... En propiedad      100  
## 4 Mujer      52 Espana       ES21  Biparental c... En propiedad      500  
## 5 Hombre      39 Espana       ES21  2 adultos si... En propiedad      300  
## # i 6 more variables: unidades_consumo <dbl>, renta_hogar <dbl>,  
## #   ingresos_anuales <dbl>, riesgo_pobreza <dbl>,  
## #   situacion_economica_infancia <chr>, pesos <dbl>
```

SELECCIONAR columnas: `select()`

Utilizamos `select()` para guardar sólo aquellas variables que nos interesan.

Podemos seleccionar por nombre de columna:

```
ecv19 %>%  
  select(sexo, renta_hogar)
```

```
## # A tibble: 39,852 × 2  
##   sexo    renta_hogar  
##   <chr>     <dbl>  
## 1 Hombre    31627.  
## 2 Mujer     31627.  
## 3 Mujer     42250  
## 4 Hombre    42250  
## 5 Mujer     7043  
## 6 Mujer     7043  
## 7 Mujer     7043  
## 8 Mujer     12303.  
## 9 Hombre    12303.  
## 10 Mujer    43889.  
## # i 39,842 more rows
```

SELECCIONAR columnas: `select()`

Como sucedía al filtrar, la función `select()` es bastante versatil y nos permite:

- Seleccionar **varias variables a la vez** (concatenando sus nombres).

```
ecv19 %>%  
  select(sexo:tipo_hogar)
```

```
## # A tibble: 39,852 × 5  
##   sexo     edad nacionalidad      ccaa tipo_hogar  
##   <chr>    <dbl> <chr>        <chr> <chr>  
## 1 Hombre     70 Espana       ES21  2 adultos sin niños  
## 2 Mujer      68 Espana       ES21  2 adultos sin niños  
## 3 Mujer      72 Espana       ES21  2 adultos sin niños  
## 4 Hombre     60 Espana       ES21  2 adultos sin niños  
## 5 Mujer      54 Espana       ES21 Biparental con niños/as  
## 6 Mujer      26 Espana       ES21 Biparental con niños/as  
## 7 Mujer      23 Espana       ES21 Biparental con niños/as  
## 8 Mujer      61 Espana       ES21 Monomarental con niños/as  
## 9 Hombre     22 Extranjero (resto mundo) ES21 Monomarental con niños/as  
## 10 Mujer     78 Espana       ES21 2 adultos sin niños  
## # i 39,842 more rows
```

SELECCIONAR columnas: `select()`

Por posición:

```
ecv19 %>%  
  select(1:3)
```

```
## # A tibble: 39,852 × 3  
##   sexo     edad nacionalidad  
##   <chr>    <dbl> <chr>  
## 1 Hombre    70  Espana  
## 2 Mujer     68  Espana  
## 3 Mujer     72  Espana  
## 4 Hombre    60  Espana  
## 5 Mujer     54  Espana  
## 6 Mujer     26  Espana  
## 7 Mujer     23  Espana  
## 8 Mujer     61  Espana  
## 9 Hombre    22  Extranjero (resto mundo)  
## 10 Mujer    78  Espana  
## # i 39,842 more rows
```

SELECCIONAR columnas: `select()`

Todas las columnas menos una:

```
ecv19 %>%  
  select(-ccaa)
```

```
## # A tibble: 39,852 × 12  
##   sexo     edad nacionalidad      tipo_hogar regimen_vivienda nivel_educativo  
##   <chr>    <dbl> <chr>          <chr>        <chr>                  <chr>  
## 1 Hombre     70  Espana         2 adultos... En propiedad 300  
## 2 Mujer      68  Espana         2 adultos... En propiedad 300  
## 3 Mujer      72  Espana         2 adultos... En propiedad 200  
## 4 Hombre     60  Espana         2 adultos... En propiedad 300  
## 5 Mujer      54  Espana        Biparental... En propiedad 500  
## 6 Mujer      26  Espana        Biparental... En propiedad 500  
## 7 Mujer      23  Espana        Biparental... En propiedad 344  
## 8 Mujer      61  Espana        Monomarental... En alquiler 500  
## 9 Hombre     22  Extranjero (resto m... Monomarental... En alquiler 344  
## 10 Mujer     78  Espana        2 adultos... En propiedad 100  
## # i 39,842 more rows  
## # i 6 more variables: unidades_consumo <dbl>, renta_hogar <dbl>,  
## #   ingresos_anuales <dbl>, riesgo_pobreza <dbl>,  
## #   situacion_economica_infancia <chr>, pesos <dbl>
```

SELECCIONAR columnas: `select()`

E incluso quitar varias:

```
ecv19 %>%  
  select(-c(4:8))
```

```
## # A tibble: 39,852 × 8  
##   sexo     edad nacionalidad      renta_hogar  ingresos_anuales riesgo_pobreza  
##   <chr>    <dbl> <chr>            <dbl>            <dbl>            <dbl>  
## 1 Hombre     70 Espana          31627.             0              0  
## 2 Mujer      68 Espana          31627.             0              0  
## 3 Mujer      72 Espana          42250              0              0  
## 4 Hombre     60 Espana          42250            41600             0  
## 5 Mujer      54 Espana          7043              0              1  
## 6 Mujer      26 Espana          7043            4680             1  
## 7 Mujer      23 Espana          7043            2340             1  
## 8 Mujer      61 Espana          12303.             0              1  
## 9 Hombre     22 Extranjero (resto m... 12303.            4680             1  
## 10 Mujer     78 Espana          43889.             0              0  
## # i 39,842 more rows  
## # i 2 more variables: situacion_economica_infancia <chr>, pesos <dbl>
```

SELECCIONAR columnas: `select()`

- Seleccionar columnas que **contengan** un texto (`contains()`)

```
tb <- tibble("edad" = c(30, 35, 40),
             "color_ojos" = c("azul", "amarillo", "negro"),
             "pelo_color" = c("negro", "marrón", "rubio"))

tb %>%
  select(contains("color"))
```

```
## # A tibble: 3 × 2
##   color_ojos pelo_color
##   <chr>       <chr>
## 1 azul        negro
## 2 amarillo    marrón
## 3 negro       rubio
```

RECOLOCAR columnas: `relocate()`

Para facilitar la **recolección** tenemos una función para ello, `relocate()`, indicándole en `.after` o `.before` detrás o delante de qué columnas queremos moverlas.

```
ecv19 %>%  
  relocate(nivel_educativo, .before = ccaa)
```

```
## # A tibble: 39,852 × 13  
##   sexo     edad nacionalidad  nivel_educativo ccaa tipo_hogar regimen_vivienda  
##   <chr>    <dbl> <chr>          <chr>        <chr> <chr>           <chr>  
## 1 Hombre     70 Espana         300          ES21  2 adultos... En propiedad  
## 2 Mujer      68 Espana         300          ES21  2 adultos... En propiedad  
## 3 Mujer      72 Espana         200          ES21  2 adultos... En propiedad  
## 4 Hombre     60 Espana         300          ES21  2 adultos... En propiedad  
## 5 Mujer      54 Espana         500          ES21  Biparenta... En propiedad  
## 6 Mujer      26 Espana         500          ES21  Biparenta... En propiedad  
## 7 Mujer      23 Espana         344          ES21  Biparenta... En propiedad  
## 8 Mujer      61 Espana         500          ES21  Monomaren... En alquiler  
## 9 Hombre     22 Extranjero (r... 344          ES21  Monomaren... En alquiler  
## 10 Mujer     78 Espana         100          ES21  2 adultos... En propiedad  
## # i 39,842 more rows  
## # i 6 more variables: unidades_consumo <dbl>, renta_hogar <dbl>,  
## #   ingresos_anuales <dbl>, riesgo_pobreza <dbl>,
```

RENOMBRAR columnas: `rename()`

A veces también podemos querer **modificar la «metainformación»** de los datos, **renombrando columnas**. Para ello usaremos la función `rename()` poniendo primero el nombre nuevo y luego el antiguo.

```
ecv19 %>%  
  rename(genero = sexo,  
         estructura_familiar = tipo_hogar)
```

```
## # A tibble: 39,852 × 13  
##   genero   edad nacionalidad      ccaa estructura_familiar regimen_vivienda  
##   <chr>   <dbl> <chr>        <chr> <chr>                      <chr>  
## 1 Hombre     70 Espana       ES21  2 adultos sin niños En propiedad  
## 2 Mujer      68 Espana       ES21  2 adultos sin niños En propiedad  
## 3 Mujer      72 Espana       ES21  2 adultos sin niños En propiedad  
## 4 Hombre     60 Espana       ES21  2 adultos sin niños En propiedad  
## 5 Mujer      54 Espana       ES21 Biparental con niñ... En propiedad  
## 6 Mujer      26 Espana       ES21 Biparental con niñ... En propiedad  
## 7 Mujer      23 Espana       ES21 Biparental con niñ... En propiedad  
## 8 Mujer      61 Espana       ES21 Monomarental con n... En alquiler  
## 9 Hombre     22 Extranjero (resto mu... ES21 Monomarental con n... En alquiler  
## 10 Mujer     78 Espana       ES21  2 adultos sin niños En propiedad  
## # i 39,842 more rows
```

MODIFICAR columnas: mutate()



MODIFICAR columnas: mutate()

En muchas ocasiones querremos **modificar o crear variables**. Para ello tenemos la función **mutate()**. Vamos a crear una **nueva variable** **renta_equivale** con la variable de renta dividida entre las unidades de consumo de cada hogar (escala OCDE).

```
ecv19 %>%  
  mutate(renta_equivale = renta_hogar/unidades_consumo)
```

```
## # A tibble: 39,852 × 14  
##   sexo     edad nacionalidad    ccaa tipo_hogar regimen_vivienda nivel_educativo  
##   <chr>   <dbl> <chr>        <chr> <chr>          <chr>           <chr>  
## 1 Hombre     70 Espana       ES21  2 adultos... En propiedad 300  
## 2 Mujer      68 Espana       ES21  2 adultos... En propiedad 300  
## 3 Mujer      72 Espana       ES21  2 adultos... En propiedad 200  
## 4 Hombre      60 Espana       ES21  2 adultos... En propiedad 300  
## 5 Mujer      54 Espana       ES21 Biparental... En propiedad 500  
## 6 Mujer      26 Espana       ES21 Biparental... En propiedad 500  
## 7 Mujer      23 Espana       ES21 Biparental... En propiedad 344  
## 8 Mujer      61 Espana       ES21 Monomaren... En alquiler 500  
## 9 Hombre      22 Extranjero (r... ES21 Monomaren... En alquiler 344  
## 10 Mujer     78 Espana       ES21  2 adultos... En propiedad 100  
## # i 39,842 more rows  
## # i 7 more variables: unidades_consumo <dbl>, renta_hogar <dbl>,
```

MODIFICAR columnas: `transmute()`

Otra opción es **quedarnos solo con las modificadas** (por ejemplo, para ver si hace lo que debe) con `transmute()`

```
ecv19 %>%  
  transmute(renta_equivalente = renta_hogar/unidades_consumo)
```

```
## # A tibble: 39,852 × 1  
##   renta_equivalente  
##       <dbl>  
## 1 21084.  
## 2 21084.  
## 3 28167.  
## 4 28167.  
## 5 3522.  
## 6 3522.  
## 7 3522.  
## 8 8202.  
## 9 8202.  
## 10 29259.  
## # i 39,842 more rows
```

MODIFICAR columnas: mutate()

También podemos combinarlo con la función `if_else()`, que nos puede ayudar a **recategorizaciones sencillas**.

- `if_else` (condición, verdadero, falso) nos devuelve una columna con dos valores (V/F) en función de una condición:

```
ecv19 %>%
  select(edad, renta_hogar, unidades_consumo) %>%
  mutate(renta_equiv = (renta_hogar/unidades_consumo),
         categoria_ingresos = if_else(renta_equiv <= 11500, "Bajos", "Resto"))
```

```
## # A tibble: 39,852 × 5
##       edad    renta_hogar  unidades_consumo   renta_equiv categoria_ingresos
##     <dbl>      <dbl>            <dbl>        <dbl>      <chr>
## 1     70      31627.          1.5        21084.    Resto
## 2     68      31627.          1.5        21084.    Resto
## 3     72      42250            1.5        28167.    Resto
## 4     60      42250            1.5        28167.    Resto
## 5     54      7043              2        3522.    Bajos
## 6     26      7043              2        3522.    Bajos
## 7     23      7043              2        3522.    Bajos
## 8     61     12303.           1.5        8202.    Bajos
```

MODIFICAR columnas: mutate() + case_when()

Para **recategorizaciones más complejas** tenemos a nuestra disposición **case_when()**. Supongamos por ejemplo que queremos crear una **categoría en función de la edad**.

- Si `edad <= 35` → serán "35 años o menos".
- Si `edad %in% c(36:64)` → serán "36-64 años"
- Si no se cumple lo anterior (`TRUE`) → serán "65 o más años"

```
ecv19 %>%  
  select(edad) %>%  
  mutate(edad_categoria = case_when(edad <= 35 ~ "35 o menos años",  
                                     edad %in% c(36:64) ~ "36-64 años",  
                                     TRUE ~ "65 o más años"))
```

```
## # A tibble: 39,852 × 2  
##   edad  edad_categoria  
##   <dbl> <chr>  
## 1    70  65 o más años  
## 2    68  65 o más años  
## 3    72  65 o más años  
## 4    60  36-64 años  
## 5    54  36-64 años  
## 6    26  35 o menos años
```

MODIFICAR columnas: mutate() + case_when()

dplyr::case_when()

IF ELSE...
(but you love it?)

df %>% ADD COLUMN 'danger'
mutate(danger = case_when(type == "kraken" ~ "extreme!",
TRUE ~ "high"))
OTHERWISE, danger is high.

IF type is kraken THEN danger is extreme!

danger is extreme!

danger is high!

danger is high!

danger is extreme!

danger is high!

type	age	danger
kraken	baby	extreme!
dragon	adult	high
cyclops	teen	high
kraken	adult	extreme!
dragon	teen	high

@allison_horst

Ilustración: Allison Horst

MODIFICAR columnas: `mutate()` + `case_when()`

Las condiciones de `case_when()` pueden combinar varias variables, cómo por ejemplo:

- Si tiene menos de 30 años e ingresa menos de 6000 euros --> "Joven precario"
- Si tiene entre 40 y 55 años e ingresa más de 40000 euros --> "Adulto acomodado"
- En caso contrario --> "Otros"

```
ecv19 %>%  
  select(edad,ingresos_anuales) %>%  
  mutate(edad_ingresos =  
    case_when(edad < 30 & ingresos_anuales < 6000 ~ "Joven precario",  
              edad %in% c(40:55) & ingresos_anuales > 40000 ~ "Adulto acomodado",  
              TRUE ~ "Otros"))
```

```
## # A tibble: 39,852 × 3  
##   edad  ingresos_anuales  edad_ingresos  
##   <dbl>      <dbl> <chr>  
## 1    70        0 Otros  
## 2    68        0 Otros  
## 3    72        0 Otros  
## 4    60      41600 Otros  
## 5    54        0 Otros  
## 6    26      4680 Joven precario
```

RESUMIR: group_by() + summarise()

group_by() se utiliza para agrupar un conjunto de datos según una o varias variables.

Una vez has agrupado, puedes utilizar summarise() si quieres "resumirlas".

RECUERDA hacer ungroup() si quieres volver a trabajar con los datos sin agrupar.

```
ecv19 %>%  
  group_by(sexo) %>%  
  summarise(media_ingresos = mean(ingresos_anuales, na.rm = TRUE))
```

```
## # A tibble: 2 × 2  
##   sexo    media_ingresos  
##   <chr>        <dbl>  
## 1 Hombre      9611.  
## 2 Mujer       6273.
```

RESUMIR: group_by() + summarise()

Si observamos la base de datos original, podemos observar que tenemos una variable llamada **pesos**. Esta variable es muy común encontrarla en multitud de encuestas y es MUY IMPORTANTE saber usarla cuando realizamos nuestras estimaciones e inferencias.

Si queremos aplicar los pesos poblacionales podemos usar la función **weighted.mean()**:

```
ecv19 %>%
  group_by(sexo) %>%
  summarise(media_ingresos_weighted = weighted.mean(ingresos_anuales,
                                                       w = pesos,
                                                       na.rm = TRUE))
```

```
## # A tibble: 2 × 2
##   sexo   media_ingresos_weighted
##   <chr>          <dbl>
## 1 Hombre        9895.
## 2 Mujer         6533.
```

REORDENAR filas: `arrange()`

`arrange()` ordena las observaciones en función de una o más variables. Pasándole las variables que usaremos para la ordenación (por defecto de menor a mayor), podemos invertirlo usando `desc()`.

```
ecv19 %>% arrange(sexo, desc(edad))
```

```
## # A tibble: 39,852 × 13
##   sexo     edad nacionalidad    ccaa tipo_hogar regimen_vivienda nivel_educativo
##   <chr>   <dbl> <chr>        <chr> <chr>          <chr>                <chr>
## 1 Hombre     86 Espana       ES52  Otros hog... En propiedad 000
## 2 Hombre     86 Espana       ES52  Otros hog... En propiedad 100
## 3 Hombre     86 Espana       ES61  2 adultos... En propiedad 000
## 4 Hombre     86 Espana       ES43  2 adultos... En propiedad 200
## 5 Hombre     86 Espana       ES43  Uniperson... En propiedad 000
## 6 Hombre     86 Espana       ES43  2 adultos... En propiedad 500
## 7 Hombre     86 Espana       ES43  2 adultos... En propiedad 200
## 8 Hombre     86 Espana       ES43  2 adultos... En propiedad 100
## 9 Hombre     86 Extranjero (r... ES53  Uniperson... En propiedad 500
## 10 Hombre    86 Extranjero (r... ES53  Uniperson... En alquiler 500
## # i 39,842 more rows
## # i 6 more variables: unidades_consumo <dbl>, renta_hogar <dbl>,
## #   ingresos_anuales <dbl>, riesgo_pobreza <dbl>,
```

Bonus track: transformando la estructura de los datos con {pivot_longer} y {pivot_wider} 

Datos SUCIOS: messy data

Por ejemplo, vamos a cargar la tabla `table4a` del paquete `{tidyverse}` (que ya lo tenemos cargado del entorno `{tidyverse}`).

```
table4a
```

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

¿Qué falla?

Datos SUCIOS: messy data

table4a

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

¿Qué falla?

Aunque la columna **\$country** representa una variable, las otras columnas no: **ambas son la misma variable**, solo que medida en años distintos (que debería ser a su vez otra variable), de forma que **cada fila está representando dos observaciones** (1999, 2000).

✖ Cada **variable en una columna**.

✖ Cada **observación/individuo en una fila** diferente.

✖ Cada **celda con un único valor**.

Datos SUCIOS: messy data

Lo que haremos será incluir una nueva columna llamada (por ejemplo) **year** que nos marque el año y otra llamada **cases** que nos diga el valor de la variable de interés en cada uno de esos años.

table4a

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K



country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

key value

Datos SUCIOS: messy data

Lo que haremos será incluir una nueva columna llamada (por ejemplo) **year** que nos marque el año y otra llamada **cases** que nos diga el valor de la variable de interés en cada uno de esos años.

Con la función **pivot_longer()** pivotaremos la tabla para pasarla a **formato long**:

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

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

Datos SUCIOS: messy data

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

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

- **cols**: el **nombre de las columnas a pivotar** (con comillas por ser números y no caracteres).
- **names_to**: el **nombre de la nueva columna** a la que mandamos los **nombres** de las columnas.
- **values_to**: el **nombre de la nueva columna** a la que vamos a mandar los **datos**.

Datos SUCIOS: messy data

Veamos un segundo tipo de dato sucio: vamos a cargar la tabla `table2` del paquete `{tidyverse}` (que ya lo tenemos cargado del entorno `{tidyverse}`). **¿Qué falla?**

```
table2
```

```
## # A tibble: 12 × 4
##   country     year   type     count
##   <chr>       <dbl> <chr>     <dbl>
## 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 SUCIOS: messy data

```
head(table2)
```

```
## # A tibble: 6 × 4
##   country     year   type     count
##   <chr>       <dbl> <chr>     <dbl>
## 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
```

country	year	key	value	country	year	cases	population
Afghanistan	1999	cases	745	Afghanistan	1999	7	19987071
Afghanistan	1999	population	19987071	Afghanistan	2000	2666	20595360
Afghanistan	2000	cases	2666	Brazil	1999	37737	172006362
Afghanistan	2000	population	20595360	Brazil	2000	80488	174504898
Brazil	1999	cases	37737	China	1999	212258	1272915272
Brazil	1999	population	172006362	China	2000	213766	1280428583
Brazil	2000	cases	80488				
Brazil	2000	population	174504898				
China	1999	cases	212258				
China	1999	population	1272915272				
China	2000	cases	213766				
China	2000	population	1280428583				

table2

✖ Cada **observación/individuo en una fila** diferente.

Fíjate en las cuatro primeras filas: los registros con el mismo año deberían ser el mismo, es la misma información, **debería estar en la misma fila**, pero está dividida en dos.

Datos SUCIOS: messy data

Lo que haremos será lo opuesto a antes: con `pivot_wider()` «ampliaremos» la **tabla a lo ancho**, con menos filas pero con más columnas.

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

```
## # A tibble: 6 × 4  
##   country     year   cases population  
##   <chr>       <dbl>   <dbl>      <dbl>  
## 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
```

- `names_from`: el **nombre de la columna original** de la que vamos a sacar las **nuevas columnas** que vamos a crear (`cases` y `population`).
- `values_from`: el **nombre de la columna original** de la que vamos a sacar los **datos**.

Otras funciones útiles de dplyr

- `distinct()` para seleccionar filas únicas de un marco de datos.
- `top_n()` para filtrar las n observaciones principales (se puede utilizar en combinación con `arrange()`).
- `ntile()` clasifica los valores en el número de grupos que proporcione. Útil para crear deciles, percentiles o cuartiles.
- Todas las funciones: <https://dplyr.tidyverse.org/reference/index.html>