

Introducción a R y Tidyverse

Sesión 03

Laboratorio de Innovación en Salud

2022-06-04

 @healthinnovation
 @innovalab_imt
 innovalab.info

Contenidos

- Reconocimiento de variables (función glimpse)
- Anidación de funciones con pipe (%>%)
- Cálculo de variables (introducción a mutate)
- Contabilización y ordenamiento en base a variables (uso de count y arrange).
- Explicación de tidy data
- Modificación a la estructura de los datos mediante pivot longer y wider

Exploración competencial

Función Glimpse

¿Para qué sirve? Ejemplo Glimpse

- Versión transpuesta de print
- Ayuda a visualizar la mayor cantidad de datos de muchas columnas.
- Muestra el nombre de la variable junto con una designación de tipo de variable.

Mediante la función:

```
glimpse()
```

Recordar importar el paquete tidyverse

```
library(tidyverse)
```

```
## — Attaching packages ——————
```

```
## ✓ ggplot2 3.3.6      ✓ purrr   0.3.4
## ✓ tibble  3.1.7      ✓ dplyr    1.0.9
## ✓ tidymodels 1.0.0     ✓ stringr  1.4.0
```

Función Glimpse

¿Para qué sirve?

Ejemplo

Glimpse

Así habitualmente observamos la data:

```
nycflights13::flights
```

```
## # A tibble: 336,776 × 19
##   year month   day dep_time sched_dep_time
##   <int> <int> <int>     <int>             <in
## 1 2013     1     1      517                 5
## 2 2013     1     1      533                 5
## 3 2013     1     1      542                 5
## 4 2013     1     1      544                 5
## 5 2013     1     1      554                 6
## 6 2013     1     1      554                 5
## 7 2013     1     1      555                 6
## 8 2013     1     1      557                 6
```

Con `glimpse()`, podrías tener un vistazo rápido de la estructura de los datos:

```
glimpse(nycflights13::flights)
```

```
## # Rows: 336,776
## # Columns: 19
## # $ year           <int> 2013, 2013, 2013, 2
## # $ month          <int> 1, 1, 1, 1, 1, 1, 1
## # $ day            <int> 1, 1, 1, 1, 1, 1, 1
## # $ dep_time        <int> 517, 533, 542, 544,
## # $ sched_dep_time <int> 515, 529, 540, 545,
## # $ dep_delay       <dbl> 2, 4, 2, -1, -6, -4
## # $ arr_time        <int> 830, 850, 923, 1004
## # $ sched_arr_time <int> 810, 830, 850, 1022
```

Función Glimpse

¿Para qué sirve? Ejemplo

Glimpse

Generalmente la vista de `glimpse()` es suficientemente ordenada para poder observar la estructura de la data sin problemas. Pero también se puede usar el argumento `width` para poder especificar ello.

```
glimpse(nycflights13::flights, width = 90)
```

```
## Rows: 336,776
## Columns: 19
## $ year           <int> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, ...
## $ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ day            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ dep_time        <int> 517, 533, 542, 544, 554, 554, 555, 555, 557, 557, 558, 558, 558, 558, ...
## $ sched_dep_time <int> 515, 529, 540, 545, 600, 558, 600, 600, 600, 600, 600, 600, 600, 600, ...
## $ dep_delay       <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -3, -2, -2, -2, -2, -2, -1, 0, -1, 0...
## $ arr_time        <int> 830, 850, 923, 1004, 812, 740, 913, 709, 838, 753, 849, 853, 924, ...
## $ sched_arr_time  <int> 819, 830, 850, 1022, 837, 728, 854, 723, 846, 745, 851, 856, 917, ...
```

Operador Pipe %>%

Queremos aplicar más de una función: El uso de funciones de forma anidada puede resultar ilegible o difícil de comprender.

```
tabla(formato(coeficiente(data)))
```

El operador `%>%` nos permite escribir una secuencia de operaciones de izquierda a derecha:

```
coeficiente(data) %>% formato() %>% tabla()
```

O mejor aún:

```
coeficiente(data) %>%  
formato() %>%  
tabla()
```

Útil para concatenar múltiples operaciones en dplyr

Atajo de Teclado

Cmd + Shift + M (Mac)

Ctrl + Shift + M (Windows/Linux)

Función mutate()

Con `mutate()` podemos realizar modificaciones en las variables. Por ej. sumar variables, o modificarlas de alguna manera (transformar a porcentaje, multiplicarlas por alguna constante, etc.).

Estas modificaciones pueden realizarse:

- Creando una nueva variable a partir de otras ya existentes.
- Modificar una variable existente en la misma variable.

Ejemplo:

```
library(tidyverse)  
df %>%  
  mutate(  
    New_var = var*2  
  )
```

Explicación:

En una data llamada `df` se estaría aplicando la función `mutate` creando una variable llamada `New_var` a partir de otra variable llamada `var` que está siendo multiplicada por 2.

Uso práctico de mutate()

Reconocimiento de BD

Janitor

Rename

Mutate I

Mutate II

Para esta exemplificación usaremos la base de datos del ECA sobre la erradicación de la infección por *Helicobácter Pylori* explicado en la [sesión 02](#).

```
trial_data <- readxl::read_excel("data/resea  
trial_data  
  
## # A tibble: 400 × 10  
##   `Patient number` `Baseline 13C-UBT`  
##                     <dbl> <chr>  
## 1                   1 Positive  
## 2                   2 Positive  
## 3                   3 Positive  
## 4                   4 Positive  
## 5                   5 Positive  
## 6                   6 Positive
```

```
glimpse(trial_data)  
  
## # Rows: 400  
## # Columns: 10  
## # $ `Patient number`  
## # $ `Baseline 13C-UBT`  
## # $ `Randomized group`  
## # $ `Follow-up13C-UBT (4 weeks after therapy`  
## # $ `adverse drug reactions`  
## # $ `adverse drug reactions content`  
## # $ `Adverse drug reaction classified`  
## # $ `Complete the study`  
## # $ `Uncomplated Reason`
```

Uso práctico de mutate()

Reconocimiento de BD

Janitor

Rename

Mutate I

Mutate II

En algunas ocasiones las bases de datos contienen nombres de variables muy largas o que incluso pueden contener espacios o símbolos. Una forma sencilla de solucionar ello es mediante el uso de la función `clean_names()` del paquete `janitor`.

```
trial_data <- trial_data %>%  
  janitor::clean_names()  
trial_data
```

```
## # A tibble: 400 × 10  
##   patient_number baseline_13c_ubt randomized_group follow_up13c_ubt_4_w... adverse_drug_re...  
##   <dbl> <chr>           <chr>           <chr>           <chr>  
## 1 1 Positive group B Negative No  
## 2 2 Positive group A Negative Yes  
## 3 3 Positive group B Negative No  
## 4 4 Positive group A Positive Yes
```

Uso práctico de mutate()

Reconocimiento de BD

Janitor

Rename

Mutate I

Mutate II

A pesar de que `janitor::clean_names()` nos proporciona una gran solución para el formateo de nombres de variables que tienen espacios y/o símbolos, a veces podría ser necesario modificar específicamente el nombre de algunas variables. Para ello usaremos la función `rename` de la siguiente manera:

```
trial_data <- trial_data %>%
  rename(
    follow_4_weeks = follow_up13c_ubt_4_weeks_after_therapy
  )
trial_data

## # A tibble: 400 × 10
##   patient_number baseline_13c_ubt randomized_group follow_4_weeks adverse_drug_reactions
##   <dbl> <chr>           <chr>           <chr>           <chr>
## 1 1     Positive       group B         Negative        No
## 2 2     Positive       group A         Negative        Yes
```

Uso práctico de mutate()

Reconocimiento de BD

Janitor

Rename

Mutate I

Mutate II

Una de las primeras cosas que podemos hacer con `mutate` durante el primer contacto con la data que trabajemos es:

- Configurar variables **ID** como **texto** (**character**).
- Configurar variables como **factores**.
- Configurar respuestas **NA** en caso tengan alguna otra codificación.

1. Configurar la variable **ID**

```
trial_data %>%  
  mutate(  
    patient_number = as.character(patient_number)  
  )  
  
## # A tibble: 400 × 10  
##       patient_number baseline_13c_ubit  
##             <chr>                <chr>  
##   1 1                    Positive  
##   2 2                    Positive  
##   3 3                    Positive  
##   4 4                    Positive  
##   5 5                    Positive
```

Uso práctico de mutate()

Reconocimiento de BD

Janitor

Rename

Mutate I

Mutate II

2. Configurar **fatores**

```
trial_data %>%  
  mutate(  
    randomized_group = factor(randomized_group),  
    complete_the_study = factor(complete_the_stu  
  )  
  
## # A tibble: 400 × 10  
##   patient_number baseline_13c_ubt  
##       <dbl> <chr>  
## 1 1 Positive  
## 2 2 Positive  
## 3 3 Positive  
## 4 4 Positive
```

3. Configurar respuestas NA

Para reemplazar respuestas dependiendo de una condición en particular se puede utilizar la función `case_when()` dentro de un `mutate()`

```
trial_data %>%  
  mutate(  
    Var = case_when(  
      Var == "Text" ~ "New_Text",  
      TRUE ~ Var  
    )  
  )
```

De esta manera en la variable `Var` se reemplazará todos los casos que registren el dato de `Text` por `New_Text`.

Uso de count

La función `count` es bastante sencilla y poderosa a la vez. Permite obtener una tabla en formato `tibble` que representará las frecuencias (cantidades `n`) de una o múltiples variables en específico.

```
trial_data %>%  
  count(adverse_drug_reactions)
```

```
## # A tibble: 4 × 2  
##   adverse_drug_reactions     n  
##   <chr>                  <int>  
## 1 NA                      10  
## 2 No                      295  
## 3 Yes                     94  
## 4 <NA>                     1
```

La tabla generada muestra que cantidad de personas han tenido reacciones adversas a los medicamentos ya sea en el Grupo A o B. Sin embargo, también se aprecia que hay 2 respuestas que indican la presencia de datos vacíos: `NA` y `<NA>`. El primero está codificado directamente como texto (`character`) y el segundo es un `NA` real, es decir que comunica la ausencia de un dato.

Uso de mutate() y case_when()

[Explicación previa del NA](#)
[Uso de case_when\(\)](#)
[Advertencia](#)

Al importar las bases de datos dentro de R, la gran mayoría de funciones (como `read_excel()`) interpretarán los valores en blanco o celdas vacías como reales **NA**. Sin embargo si en las celdas se ha llenado explícitamente el texto **NA** o se ha usado alguna codificación diferente, el valor de **NA** no se introducirá automáticamente y habrá que indicarlo como tal (`case_when()`).

Patient number	Randomized group	adverse drug reactions
1	group B	No
2	group A	Yes
8	group A	No
9	group A	No
10	group A	NA
11	group B	No
12	group B	NA
13	group A	No

```
trial_data %>%
  count(adverse_drug_reactions)
```

```
## # A tibble: 4 × 2
##   adverse_drug_reactions     n
##   <chr>                  <int>
## 1 NA                      10
## 2 No                      295
## 3 Yes                     94
## 4 <NA>                     1
```

Uso de mutate() y case_when()

Explicación previa del NA

Uso de case_when()

Advertencia

La función `case_when()` tiene una aplicación directa y perfecta para estos fines, en el que recodificaremos el **NA** introducido como texto a un **NA** real que sea reconocido como tal. El mismo procedimiento se utilizaría si los valores faltantes o perdidos se hubieran codificado de otra forma (ej. **777** o **999**).

Recordar que con **pipe (%>%)** podemos anidar muchas funciones en un solo bloque:

```
trial_data %>%
  mutate(
    adverse_drug_reactions = case_when(
      adverse_drug_reactions == "NA" ~ NA_character_()
      TRUE ~ adverse_drug_reactions
    )
  ) %>%
  count()
```

El código `TRUE ~ adverse_drug_reactions` significa que **todos los demás casos** mantendrán el valor original que el de la variable.

```
## # A tibble: 3 × 2
##   adverse_drug_reactions     n
##   <chr>                  <int>
## 1 No                      295
## 2 Yes                     94
## 3 <NA>                   11
```

Uso de mutate() y case_when()

Explicación previa del NA

Uso de case_when()

Advertencia

La función `case_when` requiere respetar de forma estricta el tipo de vector utilizado. Es decir que si se le pide recodificar una variable a número, y dentro de esa variable continúan habiendo textos, habrá un problema de **no-coerción**. Es por ese motivo que en el anterior ejemplo se usa `NA_character` en vez de únicamente `NA`, ya que este elemento como tal en realidad es de tipo lógico.

```
typeof(NA)
```

```
## [1] "logical"
```

```
typeof(NA_character_)
```

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

```
typeof(NA_real_)
```

```
## [1] "double"
```

Más sobre case_when()

Anteriormente vimos como usar la función `case_when` para recategorizar/recodificar variables en base a una condición de igualdad (`=`). Sin embargo, no es la única manera. También se puede recategorizar en base a múltiples condiciones, como `%in%` (comparar con múltiples valores a la vez), `>`, `≥`, `<` y `≤`, en conjunto con `&` y `|`.

Para exemplificar esto haremos una recategorización de la base `nycflights13::flights`:

```
nycflights13::flights %>%  
  count(year, month)  
  
## # A tibble: 12 × 3  
##   year month     n  
##   <int> <int> <int>  
## 1 2013     1 27004  
## 2 2013     2 24951
```

Consideraremos del mes 1 hasta el 6 como **2013-I** y a partir del mes 7, como **2013-II**.

```
nycflights13::flights %>%  
  mutate(  
    year = case_when(  
      month %in% 1:6 ~ "2013-I",  
      TRUE ~ "2013-II"  
    )  
  ) %>%  
  count(year)
```

Más sobre case_when()

Ya que la variable `month` es de tipo `integer` (**numérica**) tenemos más formas alternativas de conseguir exactamente el mismo resultado mostrado anteriormente.

En variables numéricas, podemos directamente usar `\leq` en las condiciones.

```
nycflights13::flights %>%
  mutate(
    year = case_when(
      month <= 6 ~ "2013-I",
      month > 6 ~ "2013-II"
    )
  ) %>%
  count(year)
```

```
## # A tibble: 2 × 2
##   year     n
##   <chr>   <int>
## 1 2013-I 166158
## 2 2013-II 170618
```

O directamente usar `TRUE ~ "Condicion, para todos los demás casos.`

```
nycflights13::flights %>%
  mutate(
    year = case_when(
      month <= 6 ~ "2013-I",
      TRUE ~ "2013-II"
    )
  ) %>%
  count(year)
```

```
## # A tibble: 2 × 2
##   year     n
##   <chr>   <int>
## 1 2013-I 166158
## 2 2013-II 170618
```

Más usos de count() y arrange()

Intro `arrange()`

Ya hemos visto que `count()` es sumamente útil. Además, basta con agregar más variables dentro de sus argumentos, para que estos automáticamente ingresen a generar una tabla de frecuencias. Sin embargo, a veces puede ser necesario ordenar esos resultados, para ello usaremos `arrange()`.

```
trial_data ← trial_data %>%  
  mutate(  
    follow_4_weeks = case_when(  
      follow_4_weeks = "NA" ~ NA_character_,  
      TRUE ~ follow_4_weeks  
    ),  
    adverse_drug_reactions = case_when(  
      adverse_drug_reactions = "NA" ~ NA_character_,  
      TRUE ~ adverse_drug_reactions  
    )  
  )
```

```
trial_data %>%  
  count(randomized_group,  
        follow_4_weeks,  
        adverse_drug_reactions)  
  
## # A tibble: 13 × 4  
##   randomized_group follow_4_weeks adverse_drug_reactions  
##   <chr>           <chr>          <chr>  
## 1 group A         Negative       No  
## 2 group A         Negative       Yes  
## 3 group A         Positive      No  
## 4 group B         Negative       No  
## 5 group B         Negative       Yes  
## 6 group B         Positive      No  
## 7 group C         Negative       No  
## 8 group C         Negative       Yes  
## 9 group C         Positive      No  
## 10 group D        Negative       No  
## 11 group D        Negative       Yes  
## 12 group D        Positive      No  
## 13 group E        Negative       No  
## 14 group E        Negative       Yes  
## 15 group F        Negative       No  
## 16 group F        Negative       Yes  
## 17 group G        Negative       No  
## 18 group G        Negative       Yes  
## 19 group H        Negative       No  
## 20 group H        Negative       Yes  
## 21 group I        Negative       No  
## 22 group I        Negative       Yes  
## 23 group J        Negative       No  
## 24 group J        Negative       Yes  
## 25 group K        Negative       No  
## 26 group K        Negative       Yes  
## 27 group L        Negative       No  
## 28 group L        Negative       Yes  
## 29 group M        Negative       No  
## 30 group M        Negative       Yes  
## 31 group N        Negative       No  
## 32 group N        Negative       Yes  
## 33 group O        Negative       No  
## 34 group O        Negative       Yes  
## 35 group P        Negative       No  
## 36 group P        Negative       Yes  
## 37 group Q        Negative       No  
## 38 group Q        Negative       Yes  
## 39 group R        Negative       No  
## 40 group R        Negative       Yes  
## 41 group S        Negative       No  
## 42 group S        Negative       Yes  
## 43 group T        Negative       No  
## 44 group T        Negative       Yes  
## 45 group U        Negative       No  
## 46 group U        Negative       Yes  
## 47 group V        Negative       No  
## 48 group V        Negative       Yes  
## 49 group W        Negative       No  
## 50 group W        Negative       Yes  
## 51 group X        Negative       No  
## 52 group X        Negative       Yes  
## 53 group Y        Negative       No  
## 54 group Y        Negative       Yes  
## 55 group Z        Negative       No  
## 56 group Z        Negative       Yes  
## 57 group AA       Negative      No  
## 58 group AA       Negative      Yes  
## 59 group BB       Negative      No  
## 60 group BB       Negative      Yes  
## 61 group CC       Negative      No  
## 62 group CC       Negative      Yes  
## 63 group DD       Negative      No  
## 64 group DD       Negative      Yes  
## 65 group EE       Negative      No  
## 66 group EE       Negative      Yes  
## 67 group FF       Negative      No  
## 68 group FF       Negative      Yes  
## 69 group GG       Negative      No  
## 70 group GG       Negative      Yes  
## 71 group HH       Negative      No  
## 72 group HH       Negative      Yes  
## 73 group II       Negative      No  
## 74 group II       Negative      Yes  
## 75 group JJ       Negative      No  
## 76 group JJ       Negative      Yes  
## 77 group KK       Negative      No  
## 78 group KK       Negative      Yes  
## 79 group LL       Negative      No  
## 80 group LL       Negative      Yes  
## 81 group MM       Negative      No  
## 82 group MM       Negative      Yes  
## 83 group NN       Negative      No  
## 84 group NN       Negative      Yes  
## 85 group OO       Negative      No  
## 86 group OO       Negative      Yes  
## 87 group PP       Negative      No  
## 88 group PP       Negative      Yes  
## 89 group QQ       Negative      No  
## 90 group QQ       Negative      Yes  
## 91 group RR       Negative      No  
## 92 group RR       Negative      Yes  
## 93 group SS       Negative      No  
## 94 group SS       Negative      Yes  
## 95 group TT       Negative      No  
## 96 group TT       Negative      Yes  
## 97 group UU       Negative      No  
## 98 group UU       Negative      Yes  
## 99 group VV       Negative      No  
## 100 group VV      Negative      Yes  
## 101 group WW       Negative      No  
## 102 group WW      Negative      Yes  
## 103 group XX       Negative      No  
## 104 group XX      Negative      Yes  
## 105 group YY       Negative      No  
## 106 group YY      Negative      Yes  
## 107 group ZZ       Negative      No  
## 108 group ZZ      Negative      Yes  
## 109 group AAA      Negative      No  
## 110 group AAA     Negative      Yes  
## 111 group BBB      Negative      No  
## 112 group BBB     Negative      Yes  
## 113 group CCC      Negative      No  
## 114 group CCC     Negative      Yes  
## 115 group DDD      Negative      No  
## 116 group DDD     Negative      Yes  
## 117 group EEE      Negative      No  
## 118 group EEE     Negative      Yes  
## 119 group FFF      Negative      No  
## 120 group FFF     Negative      Yes  
## 121 group GGG      Negative      No  
## 122 group GGG     Negative      Yes  
## 123 group HHH      Negative      No  
## 124 group HHH     Negative      Yes  
## 125 group III      Negative      No  
## 126 group III     Negative      Yes  
## 127 group JJJ      Negative      No  
## 128 group JJJ     Negative      Yes  
## 129 group KKK      Negative      No  
## 130 group KKK     Negative      Yes  
## 131 group LLL      Negative      No  
## 132 group LLL     Negative      Yes  
## 133 group MMM      Negative      No  
## 134 group MMM     Negative      Yes  
## 135 group NNN      Negative      No  
## 136 group NNN     Negative      Yes  
## 137 group OOO      Negative      No  
## 138 group OOO     Negative      Yes  
## 139 group PPP      Negative      No  
## 140 group PPP     Negative      Yes  
## 141 group QQQ      Negative      No  
## 142 group QQQ     Negative      Yes  
## 143 group RRR      Negative      No  
## 144 group RRR     Negative      Yes  
## 145 group SSS      Negative      No  
## 146 group SSS     Negative      Yes  
## 147 group TTT      Negative      No  
## 148 group TTT     Negative      Yes  
## 149 group UUU      Negative      No  
## 150 group UUU     Negative      Yes  
## 151 group VVV      Negative      No  
## 152 group VVV     Negative      Yes  
## 153 group WWW      Negative      No  
## 154 group WWW     Negative      Yes  
## 155 group XXX      Negative      No  
## 156 group XXX     Negative      Yes  
## 157 group YYY      Negative      No  
## 158 group YYY     Negative      Yes  
## 159 group ZZZ      Negative      No  
## 160 group ZZZ     Negative      Yes  
## 161 group ZZAA      Negative      No  
## 162 group ZZAA     Negative      Yes  
## 163 group ZZBB      Negative      No  
## 164 group ZZBB     Negative      Yes  
## 165 group ZZCC      Negative      No  
## 166 group ZZCC     Negative      Yes  
## 167 group ZZDD      Negative      No  
## 168 group ZZDD     Negative      Yes  
## 169 group ZZEE      Negative      No  
## 170 group ZZEE     Negative      Yes  
## 171 group ZZFF      Negative      No  
## 172 group ZZFF     Negative      Yes  
## 173 group ZZGG      Negative      No  
## 174 group ZZGG     Negative      Yes  
## 175 group ZZHH      Negative      No  
## 176 group ZZHH     Negative      Yes  
## 177 group ZZII      Negative      No  
## 178 group ZZII     Negative      Yes  
## 179 group ZZJJ      Negative      No  
## 180 group ZZJJ     Negative      Yes  
## 181 group ZZKK      Negative      No  
## 182 group ZZKK     Negative      Yes  
## 183 group ZZLL      Negative      No  
## 184 group ZZLL     Negative      Yes  
## 185 group ZZMM      Negative      No  
## 186 group ZZMM     Negative      Yes  
## 187 group ZZNN      Negative      No  
## 188 group ZZNN     Negative      Yes  
## 189 group ZZOO      Negative      No  
## 190 group ZZOO     Negative      Yes  
## 191 group ZZPP      Negative      No  
## 192 group ZZPP     Negative      Yes  
## 193 group ZZQQ      Negative      No  
## 194 group ZZQQ     Negative      Yes  
## 195 group ZZRR      Negative      No  
## 196 group ZZRR     Negative      Yes  
## 197 group ZZSS      Negative      No  
## 198 group ZZSS     Negative      Yes  
## 199 group ZZTT      Negative      No  
## 200 group ZZTT     Negative      Yes  
## 201 group ZZUU      Negative      No  
## 202 group ZZUU     Negative      Yes  
## 203 group ZZVV      Negative      No  
## 204 group ZZVV     Negative      Yes  
## 205 group ZZWW      Negative      No  
## 206 group ZZWW     Negative      Yes  
## 207 group ZZXX      Negative      No  
## 208 group ZZXX     Negative      Yes  
## 209 group ZZYY      Negative      No  
## 210 group ZZYY     Negative      Yes  
## 211 group ZZZZ      Negative      No  
## 212 group ZZZZ     Negative      Yes
```

Más usos de count() y arrange()

Intro

arrange()

La función `arrange()` permitirá ordenar un objeto en base a una o múltiples variables. En caso se ejecute sobre una variable numérica, se ordenará de menor a mayor por defecto, y en caso se ejecute sobre una variable texto (`character`) se ordenará de forma alfabética, tal y como ya lo hace `count()` por defecto.

Si se desea invertir el ordenamiento, se puede utilizar la función `desc()` dentro de `arrange()`.

```
## # A tibble: 13 × 4
##   randomized_group follow_4_weeks adverse
##   <chr>              <chr>          <chr>
## 1 group A            Positive        No
## 2 group A            Positive        Yes
## 3 group A            Negative       No
## 4 group A            Negative       Yes
## 5 group A            <NA>           <NA>
## 6 group B            Positive        No
## 7 group B            Positive        Yes
```

Uso de slice_max()

Anteriormente conocido como `top_n()`, la función `slice_max()`, permite seleccionar cuantos casos (filas) se especifique en base a una variable numérica.

```
trial_data ## # A tibble: 400 × 10
#>   patient_number baseline_13c_ubt randomized_group
#>   <dbl> <chr>           <chr>
#> 1 1     Positive        group B
#> 2 2     Positive        group A
#> 3 3     Positive        group B
#> 4 4     Positive        group A
#> 5 5     Positive        group A
#> 6 6     Positive        group A
#> 7 7     Positive        group B
#> 8 8     Positive        group A
#> 9 9     Positive        group A
#> 10 10    Positive       group A
#> # ... with 390 more rows, and 7 more variables:
#> #   follow_4_weeks <chr>  adverse_drug_reactions <chr>
```

Uso de slice_max()

Anteriormente conocido como `top_n()`, la función `slice_max()`, permite seleccionar cuantos casos (filas) se especifique en base a una variable numérica.

```
trial_data %>%  
  count(randomized_group,  
        follow_4_weeks)
```



```
## # A tibble: 6 × 3  
##   randomized_group follow_4_weeks     n  
##   <chr>              <chr>          <int>  
## 1 group A            Negative        155  
## 2 group A            Positive         41  
## 3 group A            <NA>             4  
## 4 group B            Negative        159  
## 5 group B            Positive         35  
## 6 group B            <NA>             6
```

Uso de slice_max()

Anteriormente conocido como `top_n()`, la función `slice_max()`, permite seleccionar cuantos casos (filas) se especifique en base a una variable numérica.

```
trial_data %>%
  count(randomized_group,
        follow_4_weeks) %>%
  slice_max(order_by = n,
            n = 2)
```

	## # A tibble: 2 × 3	## randomized_group	follow_4_weeks	n
	## <chr>	<chr>	<int>	
## 1	group B	Negative	159	
## 2	group A	Negative	155	

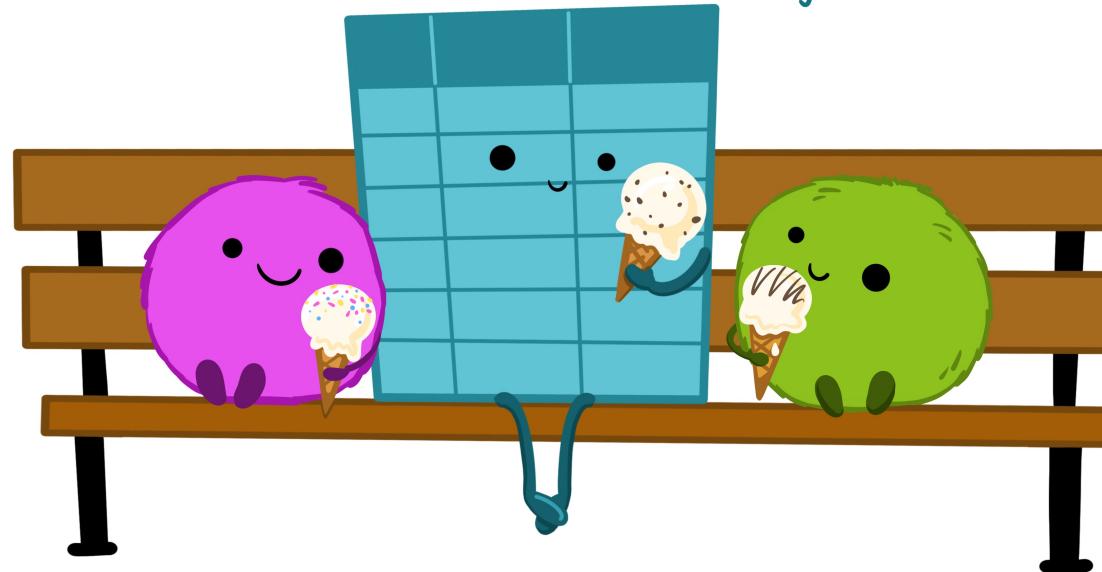
¡Hazlo tú mismo!

1. Generar una tabla de frecuencias para conocer cuantos de los que completaron el estudio, presentaron reacciones adversas al tratamiento
2. Complementar la tabla del anterior punto agregando la variable del grupo de tratamiento al que pertenece.

08 : 00

Introducción a tidydata

make friends with tidy data.



Introducción a tidydata

Tener datos ordenados (**tidydata**) significa tener una BD con estructuras adecuadas, donde cada registro representa una fila, cada variable representa una columna y cada celda contiene una simple medida (**Hadley Wickham, 2014**).

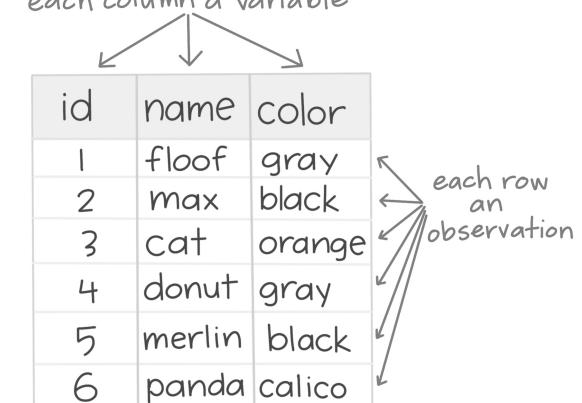
“**TIDY DATA** is a standard way of mapping the meaning of a dataset to its structure.”

—HADLEY WICKHAM

In tidy data:

- each variable forms a column
- each observation forms a row
- each cell is a single measurement

each column a variable



id	name	color
1	floof	gray
2	max	black
3	cat	orange
4	donut	gray
5	merlin	black
6	panda	calico

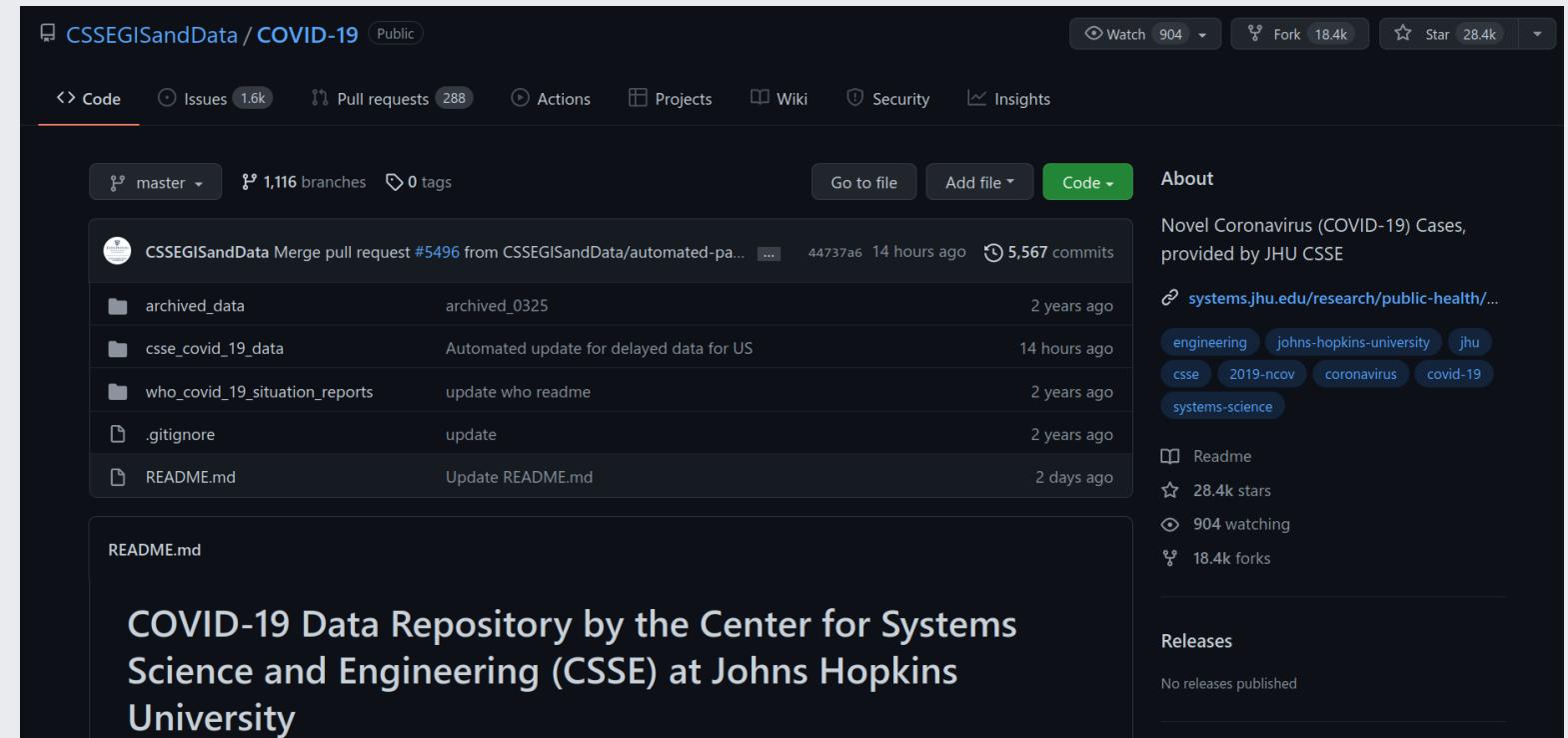
each row an observation

Wickham, H. (2014). Tidy Data. Journal of Statistical Software 59 (10). DOI: 10.18637/jss.v059.i10

Uso de pivot_longer() y pivot_wider()

Intro Importación ¿Pivot? Pivotear

Para este ejemplo usaremos una base de datos que contiene casos de personas fallecidas a causa del coranovirus a nivel de país, este dataset está alojada en el siguiente repositorio de GitHub ([click aquí](#)).



The screenshot shows the GitHub repository page for CSSEGISandData / COVID-19. The repository has 904 watchers, 18.4k forks, and 28.4k stars. It contains 1,116 branches and 0 tags. The master branch is selected. The repository description states: "Novel Coronavirus (COVID-19) Cases, provided by JHU CSSE". The README.md file contains the following text:

```
COVID-19 Data Repository by the Center for Systems  
Science and Engineering (CSSE) at Johns Hopkins  
University
```

Uso de pivot_longer() y pivot_wider()

Intro Importación ¿Pivot? Pivotear

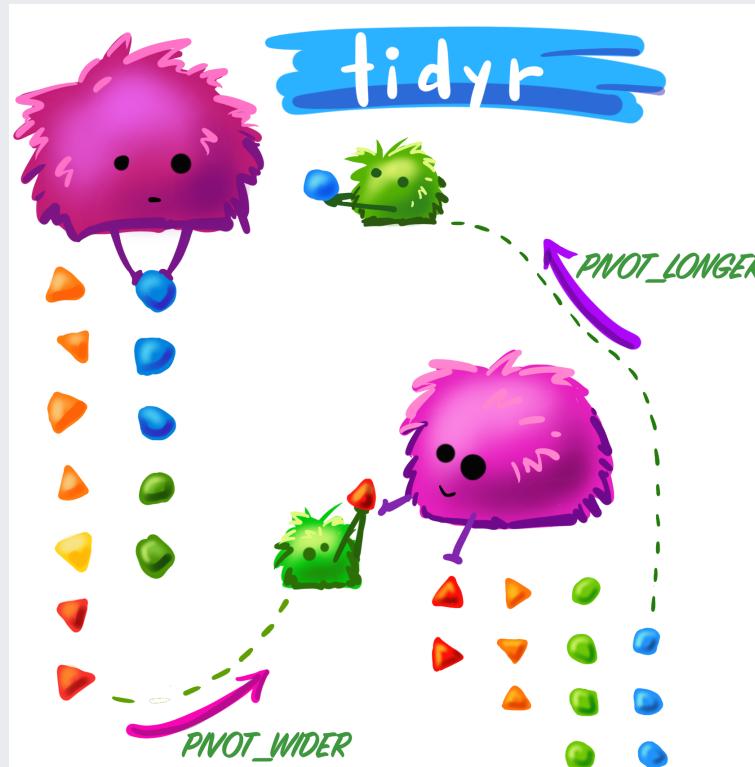
La función `read_csv` del paquete `readr` (se carga automáticamente cuando se realiza `library(tidyverse)`), permite importar archivos `csv` (data) incluso cuando el archivo está en una web.

```
covid19 ← read_csv("https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/covid19.csv")
```

	Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20	1/25/20
#	<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	<NA>	Afghanistan	33.9	67.7	0	0	0	0
2	<NA>	Albania	41.2	20.2	0	0	0	0
3	<NA>	Algeria	28.0	1.66	0	0	0	0
4	<NA>	Andorra	42.5	1.52	0	0	0	0
5	<NA>	Angola	11.2	17.0	0	0	0	0

Uso de pivot_longer() y pivot_wider()

Intro Importación ¿Pivot? Pivotear



Tenemos un conjunto de datos donde cada una de las fechas de fallecimiento por **Covid-19** se encuentran almacenadas como columnas.

Sin embargo, esta forma de organizar la información podría generar dificultades al momento de pre-procesar la información o realizar algún tipo de análisis. Tener todas las fechas y casos de muerte por COVID-19 en 2 columnas (**fecha** y **fallecidos**), podría ayudar a este proceso.

Uso de pivot_longer() y pivot_wider()

Intro Importación ¿Pivot?

Pivotear

Así, dependiendo de el formato inicial de los datos, tendremos que **pivotear** hacia la derecha (**wider**) o hacia abajo (**longer**).

wide

id	x	y	z
1	a	c	e
2	b	d	f



Uso de pivot_longer()

La función `pivot_longer` permitirá pasar todas las fechas que están a lo largo de las columnas a 2: fecha y fallecidos.

```
covid19
## # A tibble: 285 × 868
##   `Province/State` `Country/Region`   Lat   Lon
##   <chr>           <chr>          <dbl> <dbl>
## 1 <NA>             Afghanistan      33.9  67.7
## 2 <NA>             Albania        41.2  20.2
## 3 <NA>             Algeria        28.0  1.6
## 4 <NA>             Andorra        42.5  1.5
## 5 <NA>             Angola         -11.2 17.9
## 6 <NA>             Antarctica     -71.9 23.3
## 7 <NA>             Antigua and Bar... 17.1 -61.8
## 8 <NA>             Argentina      -38.4 -63.6
## 9 <NA>             Armenia        40.1  45.0
## 10 Australian Capital Te... Australia     -35.5 149.
## # ... with 275 more rows, and 864 more variables:
## #   `1/22/20` <dbl>, `1/23/20` <dbl>, `1/24/20` <dbl>, ...
```

Uso de pivot_longer()

La función `pivot_longer` permitirá pasar todas las fechas que están a lo largo de las columnas a 2: fecha y fallecidos.

```
covid19 %>%
  select(-c(Lat:Long))
```

```
## # A tibble: 285 × 866
##   `Province/State` <chr> `Country/Region` <dbl>
## 1 <NA>             <chr> 1/22/20          <dbl>
## 2 <NA>             <chr>  Afghanistan      0
## 3 <NA>             <chr>  Albania          0
## 4 <NA>             <chr>  Algeria          0
## 5 <NA>             <chr>  Andorra          0
## 6 <NA>             <chr>  Angola           0
## 7 <NA>             <chr>  Antarctica       0
## 8 <NA>             <chr>  Antigua and Bar... 0
## 9 <NA>             <chr>  Argentina        0
## 10 Australian Capital Terri... <chr> Armenia          0
## # ... with 275 more rows, and 863 more variables:
## #   `1/22/20` <dbl>, `1/23/20` <dbl>, `1/24/20` <dbl>, `1/25/20` <dbl>, ...
```

Uso de pivot_longer()

La función `pivot_longer` permitirá pasar todas las fechas que están a lo largo de las columnas a 2: fecha y fallecidos.

```
covid19 %>%
  select(-c(Lat:Long)) %>%
  pivot_longer(
    cols = -c(`Province/State`:`Count`),
    names_to = "fecha",
    values_to = "fallecidos"
  )
## # A tibble: 246,240 × 4
##   `Province/State` `Country/Region` fecha   fallecidos
##   <chr>           <chr>       <chr>     <dbl>
## 1 <NA>            Afghanistan 1/22/20      0
## 2 <NA>            Afghanistan 1/23/20      0
## 3 <NA>            Afghanistan 1/24/20      0
## 4 <NA>            Afghanistan 1/25/20      0
## 5 <NA>            Afghanistan 1/26/20      0
## 6 <NA>            Afghanistan 1/27/20      0
## 7 <NA>            Afghanistan 1/28/20      0
## 8 <NA>            Afghanistan 1/29/20      0
## 9 <NA>            Afghanistan 1/30/20      0
## 10 <NA>           Afghanistan 1/31/20      0
## # ... with 246,230 more rows
```

Uso de pivot_wider()

De forma análoga, la función `pivot_wider()` hará exactamente lo contrario, pasar de una data que se encuentre ordenada (**tidy data**), a una data ancha. Para esta función se tendrá que indicar los argumentos: `names_from` y `values_from`.

```
covid19_tidy
## # A tibble: 246,240 × 4
##   `Province/State` `Country/Region` fecha   fallecido
##   <chr>           <chr>        <chr>     <dbl>
## 1 <NA>            Afghanistan 1/22/20  
## 2 <NA>            Afghanistan 1/23/20  
## 3 <NA>            Afghanistan 1/24/20  
## 4 <NA>            Afghanistan 1/25/20  
## 5 <NA>            Afghanistan 1/26/20  
## 6 <NA>            Afghanistan 1/27/20  
## 7 <NA>            Afghanistan 1/28/20  
## 8 <NA>            Afghanistan 1/29/20  
## 9 <NA>            Afghanistan 1/30/20  
## 10 <NA>           Afghanistan 1/31/20
```

Uso de pivot_wider()

De forma análoga, la función `pivot_wider()` hará exactamente lo contrario, pasar de una data que se encuentre ordenada (**tidy data**), a una data ancha. Para esta función se tendrá que indicar los argumentos: `names_from` y `values_from`.

```
covid19_tidy %>%  
  pivot_wider(  
    names_from = "fecha",  
    values_from = "fallecidos"  
  )  
  
## # A tibble: 285 × 866  
##   `Province/State` <chr>  
##     1 <NA>  
##     2 <NA>  
##     3 <NA>  
##     4 <NA>  
##     5 <NA>  
##     6 <NA>  
##     7 <NA>  
##     8 <NA>  
##     9 <NA>  
##    10 Australian Capital Terri...   
##   `Country/Region` `1/22/20`  
##     <chr>          <dbl>  
##       Afghanistan 0  
##       Albania     0  
##       Algeria     0  
##       Andorra     0  
##       Angola      0  
##       Antarctica  0  
##       Antigua and Bar... 0  
##       Argentina   0  
##       Armenia     0  
##       Australia   0
```

¡Hazlo tú mismo!

1. Replica los procedimientos observados en las diapositivas
2. Genera un objeto tidy (`pivot_longer`)
3. Genera un objeto wider (`pivot_wider`)

08 : 00

Retroalimentación

¡Gracias!

✉ imt.innovlab@oficinas-upch.pe

🗣 [@healthinnovation](https://twitter.com/@healthinnovation)

🐦 [@innovalab_imt](https://twitter.com/@innovalab_imt)

Estas diapositivas fueron creadas mediante el paquete `xaringan` y `xaringanthemer`.