

Introducción a ciencia de datos: tidyverse

10-2022

1 Introducción

2 Tidy data

3 Magrittr pipes (tuberías)

4 Paquete tibble

5 Tidyr: Ordenar datos desordenados: tidyr

6 dplyr: Una gramática para manipular datos

7 Visualización de datos con ggplot2

Lección 1

Introducción

La librerías de tidyverse

Tidyverse es una colección de paquetes/librerías de R para ciencia de datos con diseño similar tidyverse.org

La idea principal es establecer una tecnología que aproxime el lenguaje natural a la manipulación de datos Wickham, et al. (2019)

Hadley Wickham es el director de los científicos de datos de RStudio y profesor adjunto de estadística en la Universidad de Auckland, la Universidad de Stanford y la Universidad de Rice.

Las librerías de tidyverse han venido a sustituir R base por su eficiencia y facilidad de programación para no informáticos.

Casi todas las consultas a páginas técnicas de R son o incluyen código de tidyverse.



Las librerías de tidyverse

Paquetes de tidyverse base :

- **readr**: lectura de datos
- **tibble**: una clase 'tbl_df' (el 'tibble') con una comprobación más estricta y un mejor formato que el data frame tradicional.
- **stringr**: paquete de funciones para texto
- **forcats**: paquete de funciones para factores
- **tidyverse**: arreglo y limpieza de datos
- **dplyr**: manipulación de datos
- **ggplot2**: visualización de datos (gráficos)
- **purrr**: programación funcional (pipes)

Hay muchos otros paquetes para fines especiales que se integran sin problemas, por ejemplo, lubridate (variables de tiempo), stringr (texto),forcats (factores), ...

Instalar y cargar tidyverse

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

```
-- Attaching packages ----- tidyverse 1.3.1 --
v ggplot2 3.3.5 v purrr 0.3.4
v tibble 3.1.4 v dplyr 1.0.7
v tidyr 1.1.3 v stringr 1.4.0
v readr 2.0.1 v forcats 0.5.1
-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag() masks stats::lag()
```

Se puede ver la versión del paquete tidyverse y la de los paquetes base de tidyverse.

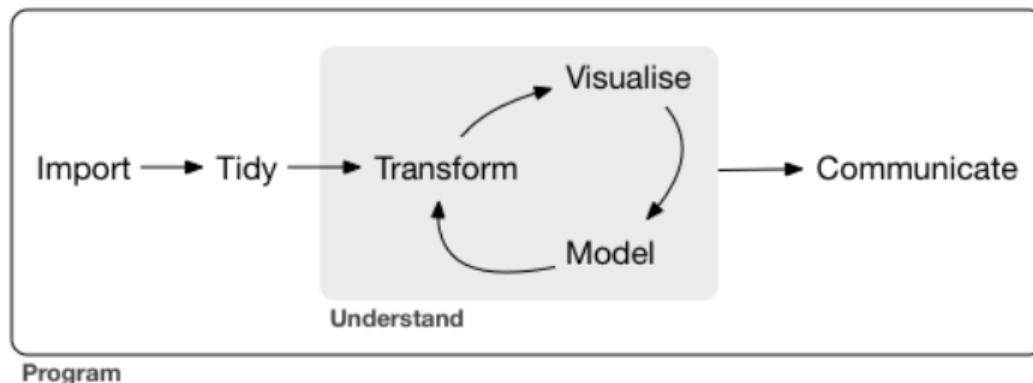
Cuidado que algunas funciones de R se sobrescriben por sus equivalentes de tidyverse.

En ocasiones es preferible indicar explícitamente el nombre de la función que deseamos utilizar, por ejemplo: `dplyr::group_by` para distinguir de `plyr::group_by` (`dplyr` es una evolución del paquete `plyr`).

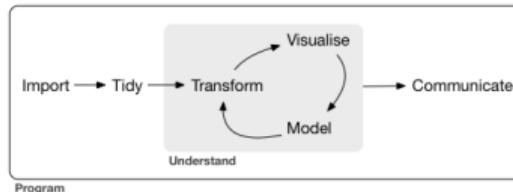
Más sobre tidyverse

Todos estos paquetes están pensados para:

- ① Tener una tecnología en la que puedan convivir desde informáticos puros, economistas, matemáticos, gestores etc. compartiendo el mismo flujo de datos. . .
- ② Facilitar el análisis y modelización de datos



Librerías para cada tarea



- **Import:** readr
- **Tidy:** tidyverse
- **Transform:** dplyr,forcats, stringr
- **Visualize:** ggplot2
- **Model:** tidymodels
- **Communicate:** rmarkdown
- **Program:** magrittr, purrr, tibble

Algunos libros:

- Wickham/Grolemund (2017).*
- Yihui Xie, J. J. Allaire, Garrett Grolemund

Más sobre tidyverse

Lo que se intenta es hacer un diseño y una gramática que sea sencilla como una API para usuarios no “tecnólogos”.

- Las tibbles como estructura de datos (superan a los data.frames y simplifican los data.table)
- El operador %>% para crear flujos de datos y funciones.
- Estandarizar la nomenclatura de las funciones,
- Establecer un orden razonable en los argumentos de las funciones (por ejemplo, fn(argumento_A = datos, argumento_B = etiquetas de las columnas, ...).

Más sobre tidyverse

La sintaxis del tidyverse puede verse como un “dialecto” de R.



Nota: Para más información, véase [Tidyverse Team \(2020\)](#) y [Wickham \(2019\)](#).

Lección 2

Tidy data

¿Qué es tidy data?

Los conjuntos de datos ordenados son todos iguales; pero cada conjunto de datos desordenado es desordenado a su manera. [Wickham/Grolemund: r4ds](#)

Si tenemos datos provenientes de distintas fuentes, seguramente tendremos que estructurarlos en una única tibble.

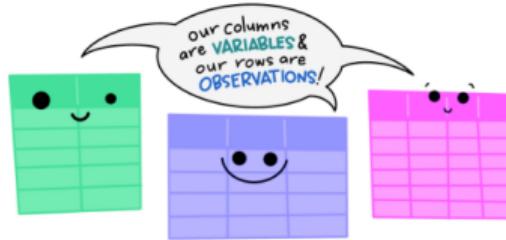
Principios de los datos estructurados:

El concepto de datos ordenados implica conjuntos de datos rectangulares y tabulares compuestos por filas y columnas:

- ① Cada variable forma una columna.
- ② Cada observación forma una fila.
- ③ Cada tipo de unidad de observación forma una tabla.

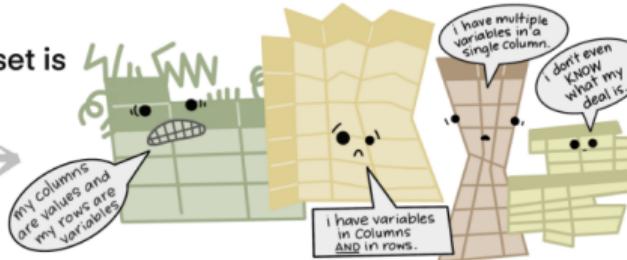
¿Qué es tidy data?

The standard structure of
tidy data means that
“tidy datasets are all alike...”



“...but every messy dataset is
messy in its own way.”

—HADLEY WICKHAM



Estructurar y ordenar datos (tidy)

Violaciones de los principios de los datos ordenados:

- ① Las cabeceras de las columnas son valores, no nombres de variables.
- ② Se almacenan múltiples variables en una columna.
- ③ Las variables se almacenan tanto en filas como en columnas.
- ④ Se almacenan múltiples tipos de unidades de observación en la misma tabla.
- ⑤ Una misma unidad de observación se almacena en varias tablas.

Estructurar y ordenar datos

Veamos ejemplos de lo anterior con unos datos de pingüinos con los que seguiremos trabajando luego.

```
#install.packages("palmerpenguins", dep=TRUE)
library("palmerpenguins")
print(penguins, width = 50)
```

```
# A tibble: 344 x 8
  species island bill_~1 bill_~2 flipp_~3 body_~4
  <fct>   <fct>    <dbl>    <dbl>    <int>    <int>
1 Adelie  Torgers  39.1     18.7     181     3750
2 Adelie  Torgers  39.5     17.4     186     3800
3 Adelie  Torgers  40.3     18       195     3250
4 Adelie  Torgers  NA        NA       NA       NA
5 Adelie  Torgers  36.7     19.3     193     3450
6 Adelie  Torgers  39.3     20.6     190     3650
7 Adelie  Torgers  38.9     17.8     181     3625
8 Adelie  Torgers  39.2     19.6     195     4675
9 Adelie  Torgers  34.1     18.1     193     3475
10 Adelie  Torgers  42       20.2     190     4250
# ... with 334 more rows, 2 more variables:
#   sex <fct>, year <int>, and abbreviated
#   variable names 1: bill_length_mm,
#   2: bill_depth_mm, 3: flipper_length_mm,
#   4: body_mass_g,
```

Estructurar y ordenar datos (tidy)

```
# A tibble: 3 x 4
  species    Biscoe   Dream  Torgersen
  <fct>      <int>    <int>     <int>
1 Adelie       44       56        52
2 Chinstrap    NA       68        NA
3 Gentoo      124      NA        NA

# A tibble: 5 x 3
  col      island   year
  <chr>    <fct>    <int>
1 Gentoo_NA  Biscoe  2007
2 Adelie_male Torgersen 2007
3 Gentoo_female Biscoe  2008
4 Chinstrap_male Dream   2008
5 Adelie_male  Torgersen 2009

# A tibble: 3 x 4
  term      bill_length_mm bill_depth_mm flipper_length_mm
  <chr>          <dbl>         <dbl>            <dbl>
1 bill_length_mm      NA        -0.235         0.656
2 bill_depth_mm      -0.235       NA        -0.584
3 flipper_length_mm   0.656      -0.584        NA
```

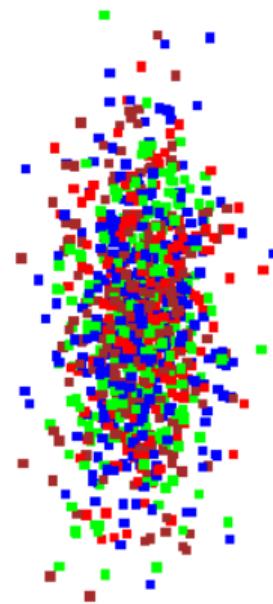
Estructurar y ordenar datos (tidy)

```
# A tibble: 6 x 6
  species   island sex     model      mpg   cyl
  <fct>     <fct> <fct>   <chr>     <dbl> <dbl>
1 Chinstrap Dream  female <NA>        NA     NA
2 Gentoo    Biscoe female <NA>        NA     NA
3 Gentoo    Biscoe male  <NA>        NA     NA
4 <NA>      <NA>   <NA>   Merc 450SLC  15.2     8
5 <NA>      <NA>   <NA> Dodge Challenger 15.5     8
6 <NA>      <NA>   <NA> Pontiac Firebird 19.2     8
```

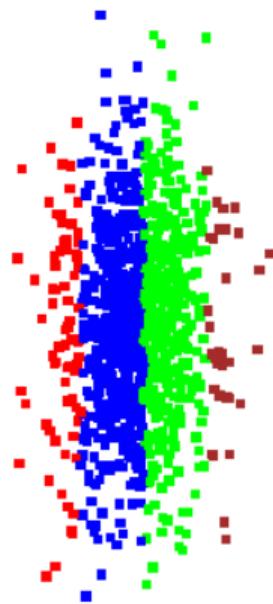
```
# A tibble: 6 x 6
  species   island sex     model      mpg   cyl
  <fct>     <fct> <fct>   <chr>     <dbl> <dbl>
1 Adelie    Dream  female <NA>        NA     NA
2 Adelie    Dream  female <NA>        NA     NA
3 Gentoo   Biscoe male  <NA>        NA     NA
4 <NA>      <NA>   <NA> Hornet Sportabout 18.7     8
5 <NA>      <NA>   <NA> Honda Civic   30.4     4
6 <NA>      <NA>   <NA> Porsche 914-2  26      4
```

Estructurar y ordenar datos (tidy)

Datos NO tidy



Datos tidy



Lección 3

Magrittr pipes (tuberías)

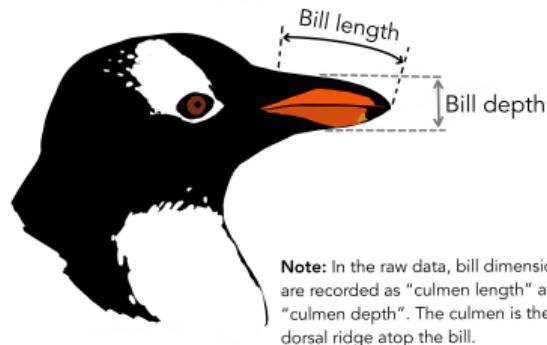
Datos de los pingüinos del Archipiélago de Palmer (Antarctica)

Como mencionamos antes, para poner ejemplos de los distintos paquetes de tidyverse utilizamos datos del paquete palmerpenguins de [Allison Horst](#).

El paquete incluye datos sobre los pingüinos observados en las islas del archipiélago Palmer, cerca de la estación Palmer, en la Antártida.



Datos de los pingüinos



Note: In the raw data, bill dimensions are recorded as "culmen length" and "culmen depth". The culmen is the dorsal ridge atop the bill.

Operadores de “tuberías” para R



Los operadores pipes de `magrittr` son:

- **Operador de tuberías:** `%>%`
- **Operador de asignación:** `%<>%`
- **Operador “T”:** `%T>%`
- **Operador de extracción (“exposition”):** `'%$%.`

Ejemplo

```
rnorm(200) %>%
matrix(ncol = 2) %T>%
plot %>% # plot no suele retornar nada
colSums
```

Operadores de “tuberías” para R

Estos operadores pretenden mejorar la legibilidad de los códigos de múltiples maneras:

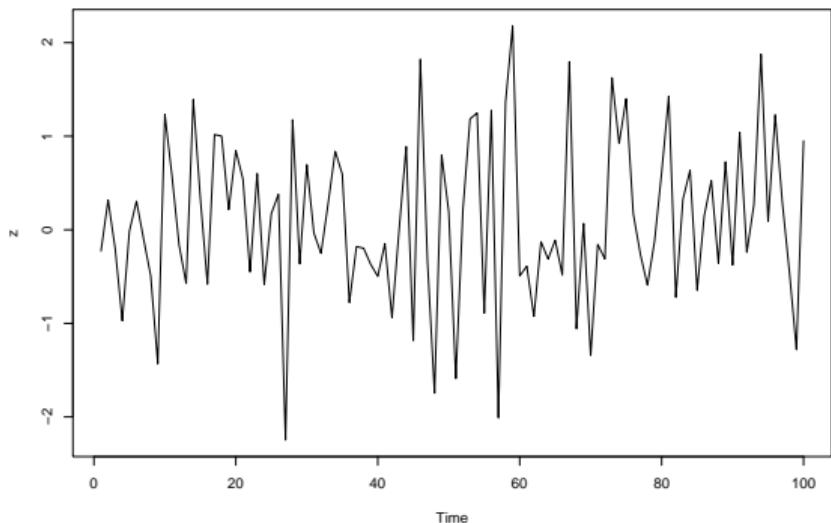
- Organizando las operaciones en una cadena de instrucciones encadenadas (de izquierda a derecha) fácilmente legible,
- Evitando las llamadas a funciones anidadas,
- Minimizando el uso de asignaciones de variables locales (<-) y definiciones de funciones,
- Añadiendo y/o eliminando fácilmente pasos del “pipeline” sin romper el código.

El operador %\$% (pasa las variables de una tibble/data.frame.

```
library(magrittr)
iris %>%
  subset(Sepal.Length > mean(Sepal.Length)) %$%
  cor(Sepal.Length, Sepal.Width)
```

Operadores de “tuberías” para R

```
data.frame(z = rnorm(100)) %$% ts.plot(z)
```



El operador pipe

pipes básicos: pasan un valor, atributo u objeto (LHS: Left Hand Side) a la siguiente llamada de función (RHS: Right Hand Side) como **primer** argumento

```
x %>% f # equivalente a: f(x)
x %>% f(y) # equivalente a: f(x, y)
x %>% f %>% g %>% h # equivalente a: h(g(f(x)))
```

El operador pipe

pipes con marcadores de posición: reenvian un valor u objeto (LHS) a la siguiente llamada de función (RHS) como **cualquier** argumento

```
x %>% f(.) # equivalente a: x %>% f
x %>% f(y, .) # equivalente a: f(y, x)
x %>% f(y, z = .) # equivalente a: f(y, z = x)
x %>% f(y = nrow(.),
           z = ncol(.)) # equivalente a: f(x, y = nrow(x), z = ncol(x))
```

Construcción de funciones con pipes

Una secuencia de código que comienza con el marcador de posición (.) devuelve una función que puede utilizarse para aplicar posteriormente la tubería a valores concretos

```
f <- . %>% cos %>% sin # equivalente a: f <- function(.) sin(cos(.))
```

```
f(20) # equivalente a: la tubería 20 %>% cos %>% sin
```

Nota: Para saber más sobre %>%, haced vignette("magrittr") en la consola de R.

Se puede obtener la cadena %>% en Rstudio desktop utilizando el atajo de teclado:

Ctrl + Shift + M.

Ejemplo con el operador pipe

Pregunta: ¿Cuál es la masa corporal media en gramos de todos los pingüinos observados en el año 2007 (tras excluir los valores perdidos)?

En un mundo sin pipes:

```
mean(subset(penguins, year == 2007)$body_mass_g, na.rm = T)
```

alternativamente:

```
peng_bm_2007 <- subset(penguins, año == 2007)$body_mass_g  
media(peng_bm_2007, na.rm = T)
```

En un mundo con pipes:

```
penguins %>%  
  subset(año == 2007) %>%  
  .$body_mass_g %>%  
  mean(na.rm = T)
```

Ventajas de usar pipes

- El estilo secuencial de las tuberías mejora la legibilidad y la lectura que las funciones anidadas.
- Hace innecesario almacenar los resultados intermedios.
- Es muy fácil añadir o eliminar pasos (empalmes de tuberías) individuales en el “pipe-line/canalización”

Las versiones recientes de R ya viene con un operador de tuberías nativo también (`|>`).

El pipe de R base

```
mtcars |> head() # es lo mismo que head(mtcars)
```

```
mpg cyl disp hp drat wt qsec vs am gear carb  
Mazda RX4     21.0   6 160 110 3.90 2.620 16.46 0 1 4 4  
Mazda RX4 Wag 21.0   6 160 110 3.90 2.875 17.02 0 1 4 4  
Datsun 710    22.8   4 108 93 3.85 2.320 18.61 1 1 4 1  
Hornet 4 Drive 21.4   6 258 110 3.08 3.215 19.44 1 0 3 1  
Hornet Sportabout 18.7   8 360 175 3.15 3.440 17.02 0 0 3 2  
Valiant       18.1   6 225 105 2.76 3.460 20.22 1 0 3 1
```

```
mtcars |> head(2) # es lo mismo que head(mtcars, 2)
```

```
mpg cyl disp hp drat wt qsec vs am gear carb  
Mazda RX4     21   6 160 110 3.9 2.620 16.46 0 1 4 4  
Mazda RX4 Wag 21   6 160 110 3.9 2.875 17.02 0 1 4 4
```

```
mtcars |> subset(cyl == 4) |> nrow()
```

```
[1] 11
```

Pipes avanzadas

Los empalmes de tuberías más avanzados como la %T% nos permiten economizar líneas de código.

- %T>% se puede utilizar para activar el efecto secundario de una función, por ejemplo, para imprimir salidas, y dejar que los datos originales pasen por alto el paso respectivo.

```
penguins[1:5, c("island", "bill_length_mm")] %T>%
  print %>% .$"bill_length_mm" %>%
  mean(na.rm=T)
```

```
# A tibble: 5 x 2
  island    bill_length_mm
  <fct>        <dbl>
1 Torgersen     39.1
2 Torgersen     39.5
3 Torgersen     40.3
4 Torgersen      NA
5 Torgersen     36.7
```

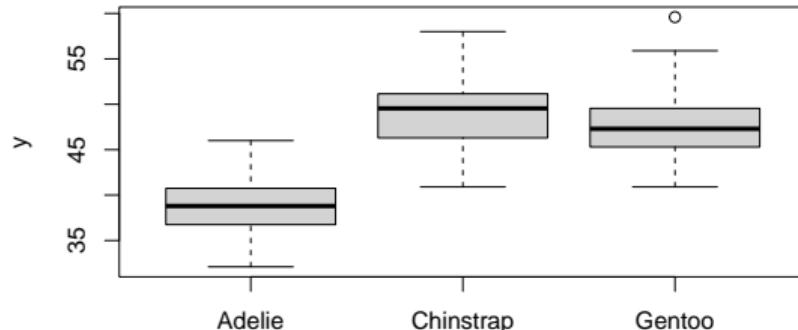
```
[1] 38.9
```

Tuberías avanzadas

- El operador `%$%` extrae las variables del objeto LHS a la expresión RHS. Es equivalente a:

```
plot(penguins$species, penguins$bill_length_mm)
```

```
penguins %$%
  plot(species,bill_length_mm)
```



Tuberías avanzadas

El operador `%<>%` se puede utilizar de forma equivalente al operador de asignación de base R (`<-`).

```
variable <- penguins$bill_length_mm  
variable %<>% mean(na.rm=T)  
variable
```

```
[1] 43.92193
```

Lección 4

Paquete tibble

Data frame avanzado: tibble

EL paquete tibble proporciona un objeto de tipo data frame mejorado: `tbl_df`. Un tibble se puede crear de cuatro maneras diferentes.

- ① A partir de vectores columna con `tibble()`.

```
tibble(  
  x = c("a", "b"),  
  y = c(1, 2),  
  z = c(T, F)  
)
```

```
# A tibble: 2 x 3  
  x     y   z  
  <chr> <dbl> <lgl>  
1 a         1 TRUE  
2 b         2 FALSE
```

Data frame avanzado: tibble

- ② Escribiendo en texto por columnas tibble, fila por fila con tribble().

```
tribble(  
  ~x, ~y, ~z,  
  "a", 1, T,  
  "b", 2, F  
)
```

```
# A tibble: 2 x 3  
  x     y   z  
  <chr> <dbl> <lgl>  
1 a        1 TRUE  
2 b        2 FALSE
```

Data frame avanzado: tibble

- ③ Creando un tibble a partir de otro objeto de las clases `matrix` o `data.frame` con `as_tibble()`.

```
data.frame(  
  x = c("a", "b"),  
  y = c(1, 2),  
  z = c(T, F)  
) %>%  
as_tibble
```

```
# A tibble: 2 x 3  
#>   x     y   z  
#>   <chr> <dbl> <lgl>  
#> 1 a       1 TRUE  
#> 2 b       2 FALSE
```

Data frame avanzado: tibble

- ④ Creando un tibble a partir de vectores con nombre con enframe().

```
c(x = "a", y = "b", z = 1) %>%
  enframe(name = "x", value = "y")
```

```
# A tibble: 3 x 2
  x     y
  <chr> <chr>
1 x     a
2 y     b
3 z     1
```

Data frame avanzado tibble

Diferencias entre tibble y data.frame

- Un tibble nunca cambia el tipo de entrada.
 - Ya no hay que preocuparse de que los caracteres se conviertan automáticamente en cadenas.
- Un tibble puede tener columnas que son listas.
- Un tibble puede tener nombres de variables no estándar.
 - Pueden empezar por un número o contener espacios.
 - Para utilizarlo se refiere a estos en un backtick: peso en Kg.
- Sólo recicla vectores de longitud 1.
- No tiene como atributo nombres de filas `row.names`.

Data frame avanzado tibble

Impresión: Por defecto, `tibble()` imprime sólo las diez primeras filas y todas las columnas que caben en la pantalla, y las clases de las columnas

penguins

```
# A tibble: 344 x 8
  species island    bill_length_mm bill_depth_mm flipper_~1 body_~2 sex     year
  <fct>   <fct>          <dbl>        <dbl>      <int>    <int> <fct> <int>
1 Adelie  Torgersen     39.1         18.7       181     3750 male   2007
2 Adelie  Torgersen     39.5         17.4       186     3800 fema-  2007
3 Adelie  Torgersen     40.3         18         195     3250 fema-  2007
4 Adelie  Torgersen     NA           NA         NA      NA <NA>  2007
5 Adelie  Torgersen     36.7         19.3       193     3450 fema-  2007
6 Adelie  Torgersen     39.3         20.6       190     3650 male   2007
7 Adelie  Torgersen     38.9         17.8       181     3625 fema-  2007
8 Adelie  Torgersen     39.2         19.6       195     4675 male   2007
9 Adelie  Torgersen     34.1         18.1       193     3475 <NA>  2007
10 Adelie Torgersen      42          20.2       190     4250 <NA>  2007
# ... with 334 more rows, and abbreviated variable names 1: flipper_length_mm,
#   2: body_mass_g
```

Data frame avanzado: tibble

Aquí se ve la diferencia con la clase `data.frame`.

```
data.frame(penguins)
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm
1	Adelie	Torgersen	39.1	18.7	181
2	Adelie	Torgersen	39.5	17.4	186
3	Adelie	Torgersen	40.3	18.0	195
4	Adelie	Torgersen	NA	NA	NA
5	Adelie	Torgersen	36.7	19.3	193
6	Adelie	Torgersen	39.3	20.6	190
7	Adelie	Torgersen	38.9	17.8	181
8	Adelie	Torgersen	39.2	19.6	195
9	Adelie	Torgersen	34.1	18.1	193
10	Adelie	Torgersen	42.0	20.2	190
11	Adelie	Torgersen	37.8	17.1	186
12	Adelie	Torgersen	37.8	17.3	180
13	Adelie	Torgersen	41.1	17.6	182
14	Adelie	Torgersen	38.6	21.2	191
15	Adelie	Torgersen	34.6	21.1	198
16	Adelie	Torgersen	36.6	17.8	185
17	Adelie	Torgersen	38.7	19.0	195
18	Adelie	Torgersen	42.5	20.7	197
19	Adelie	Torgersen	34.4	18.4	184
20	Adelie	Torgersen	46.0	21.5	194
21	Adelie	Biscoe	37.8	18.3	174
22	Adelie	Biscoe	37.7	18.7	180

Data frame avanzado: tibble

glimpse nos da la versión transpuesta de print().

```
penguins %>% glimpse
```

```
Rows: 344
Columns: 8
$ species      <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, Adelie
$ island        <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torgersen, Torgersen, Torgersen
$ bill_length_mm <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1, ~
$ bill_depth_mm  <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.1, ~
$ flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, 186
$ body_mass_g    <int> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 3475, ~
$ sex           <fct> male, female, female, NA, female, male, female, male, ~
$ year          <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007~
```

Data frame avanzado: tibble

Subconjunto: El subconjunto de un tibble ([]) siempre devuelve otro tibble y nunca un vector (en contraste con los objetos estándar data.frame).

```
data.frame(penguins) %>% .[, "species"] %>% class
```

```
[1] "factor"
```

```
penguins[, "species"] %>% class
```

```
[1] "tbl_df"     "tbl"        "data.frame"
```

Data frame avanzado: tibble

El subconjunto de un data.frame busca el nombre de variable más parecido

```
names(data.frame(penguins))
```

```
[1] "species"          "island"           "bill_length_mm"  
[4] "bill_depth_mm"   "flipper_length_mm" "body_mass_g"  
[7] "sex"              "year"
```

```
head(data.frame(penguins)$spec)
```

```
[1] Adelie Adelie Adelie Adelie Adelie  
Levels: Adelie Chinstrap Gentoo
```

'tibble' no permite la coincidencia parcial, es decir, siempre se debe proporcionar el nombre completo de la columna.

Data frame avanzado: tibble

```
head(penguins$spec)
```

Warning: Unknown or uninitialized column: 'spec'.

NULL

```
head(penguins$species)
```

[1] Adelie Adelie Adelie Adelie Adelie
Levels: Adelie Chinstrap Gentoo

- Las tibbles dan mejores mensajes Warning y Error para solucionar problemas.

Leer datos de texto rectangulares readr

El paquete **readr** proporciona funciones de lectura y escritura para múltiples formatos de archivo diferentes:

- `read_delim()`: archivos delimitados en general
- `read_csv()`: archivos separados por comas
- `read_csv2()`: archivos separados por punto y coma. En la mayoría de los países europeos, Microsoft Excel utiliza ; como delimitador común
- `read_tsv()`: archivos separados por tabulaciones
- `read_fwf()`: archivos de ancho fijo
- `read_table()`: archivos separados por espacios en blanco
- `read_log()`: archivos de registro web

Leer datos de texto rectangulares readr

- Convenientemente, las funciones `write_*`() funcionan de forma análoga.
- Se utiliza el paquete `readxl` para archivos de Excel,
- El paquete `haven` para archivos de Stata, SAS y SPSS,
- El paquete `googlesheets4` para Google Sheets
- El paquete `rvest` para archivos HTML. Paquete de referencia en el contexto de la extracción de datos de la web con R

Leer datos de texto rectangulares readr

Para ilustrar el paquete `readr`, a priori, hemos escrito un archivo csv que contiene los datos de los pingüinos, utilizando `write_csv(penguins, archivo = "datos/penguins.csv")`.

```
data <- read_csv(file = "data/penguins.csv")
```

```
New names:  
Rows: 344 Columns: 9  
-- Column specification  
----- Delimiter: "," chr  
(3): species, island, sex dbl (6): ...1, bill_length_mm, bill_depth_mm,  
flipper_length_mm, body_mass_g...  
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.  
* `` -> '...1'
```

```
data <- read_csv(file = "data/penguins.csv", col_select = c(species, island))
```

```
New names:  
Rows: 344 Columns: 2  
-- Column specification  
----- Delimiter: "," chr  
(2): species, island  
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.
```

Leer datos de texto rectangulares readr

```
data <- read_csv(file = "./data/penguins.csv",
                  col_names = paste("Var", 1:8, sep = "_"))
```

```
Rows: 345 Columns: 9
-- Column specification -----
Delimiter: ","
chr (8): Var_2, Var_3, Var_4, Var_5, Var_6, Var_7, Var_8, X9
dbl (1): Var_1

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.
```

Leer datos de texto rectangulares readr

```
data <- read_csv(file = "./data/penguins.csv", skip = 5)
```

```
Rows: 339 Columns: 9
-- Column specification ----
Delimiter: ","
chr (3): Adelie, Torgersen, female
dbl (6): 5, 36.7, 19.3, 193, 3450, 2007

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.
```

Observa que la salida de cualquier función `read_*`() es un objeto tibble.

Leer datos de texto rectangulares readr

- `readr` imprime las especificaciones de las columnas después de la importación.
- Por defecto, `readr` intenta inferir el tipo de columna (por ejemplo, `int`, `dbl`, `chr`, `fct`, `date`, `lgl`) a partir de las primeras 1.000 filas y analiza las columnas en consecuencia.
- Intenta hacer explícitas las especificaciones de las columnas. Es probable que te familiarices más con tus datos y veas advertencias si algo cambia inesperadamente.

Leer datos de texto rectangulares readr

```
read_csv(  
  archivo = "./data/penguins.csv",  
  col_types = cols(  
    species = col_character(),  
    año = col_datetime(formato = "%Y"),  
    isla = col_skip())  
)
```

Leer datos de texto rectangulares readr

Analizar sólo las primeras 1.000 filas es eficiente, pero puede llevar a conjeturas erróneas:

```
read_csv(file = "./data/penguins.csv", guess_max = 2000)
```

Nota: Encuentra más información y funciones de readr en [hoja de trucos](#).

A veces puedes tener problemas al leer datos de texto (tipo carácter): los signos especiales como ö, ä o ü pueden ser codificados de forma extraña como símbolos crípticos. En esos casos debes controlar la codificación los datos en la función read_csv (por ejemplo, UTF-8)

Leer datos de texto rectangulares readr

Supongamos que deseas dejar de utilizar los archivos .xlsx y .csv ya que no son capaces de almacenar de forma fiable los metadatos (por ejemplo, los tipos de datos).

Las funciones `write_rds()` y `read_rds()` (son warppers de `writeRDS` y `readRDS` del paquete base de R) proporcionan una buena alternativa para **serializar** tus objetos R (por ejemplo, tibbles, modelos) y almacenarlos como archivos .rds.

Más info sobre [archivos rds](#)

Leer datos de texto rectangulares readr

```
penguins %>%  
  write_rds(file = "./data/penguins.rds")
```

```
penguins <- read_rds(file = "./data/penguins.rds")
```

Nota que:

- `write_rds()` solo puede utilizarse para guardar un objeto a la vez,
- un archivo .rds cargado debe ser almacenado en una nueva variable, es decir, darle un nuevo nombre,
- `read_rds()` ¡conserva los tipos de datos!

Lección 5

Tidyr: Ordenar datos desordenados: tidyr

Tidyr: Ordenar datos desordenados: tidy



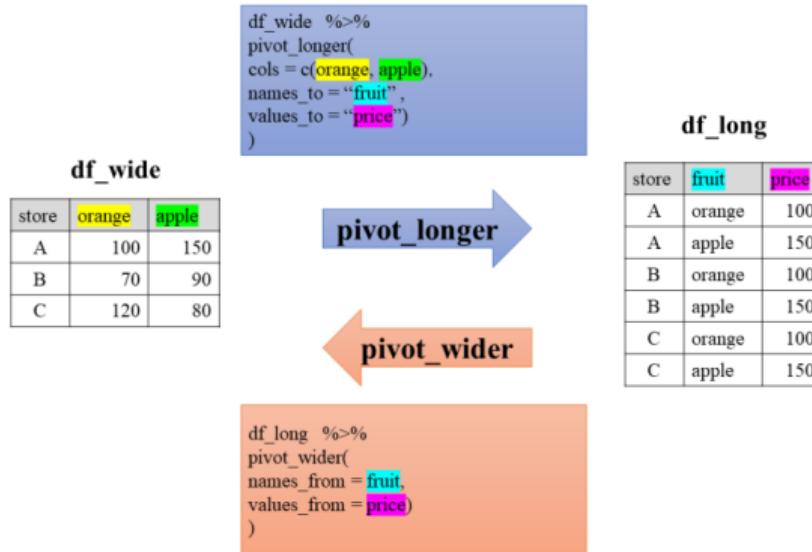
tidyR proporciona varias funciones que nos ayudarán a convertir nuestros datos en “*tidy data*” format (p.e., reestructurarlos, dividir las columnas, manejar los datos perdidos o agruparlos). De nuevo, utilizaremos los datos de los pingüinos.

penguins

```
# A tibble: 344 x 8
  species island bill_length_mm bill_depth_mm flipper_~1 body_~2 sex     year
  <fct>   <fct>      <dbl>        <dbl>       <int>    <int> <fct>    <int>
1 Adelie  Torgersen     39.1         18.7       181     3750 male    2007
2 Adelie  Torgersen     39.5         17.4       186     3800 female  2007
3 Adelie  Torgersen     40.3          18        195     3250 female  2007
4 Adelie  Torgersen      NA           NA         NA      NA <NA>    2007
5 Adelie  Torgersen     36.7         19.3       193     3450 female  2007
6 Adelie  Torgersen     39.3         20.6       190     3650 male    2007
7 Adelie  Torgersen     38.9         17.8       181     3625 female  2007
8 Adelie  Torgersen     39.2         19.6       195     4675 male    2007
9 Adelie  Torgersen     34.1         18.1       193     3475 <NA>    2007
10 Adelie Torgersen      42           20.2       190     4250 <NA>    2007
# ... with 334 more rows, and abbreviated variable names 1: flipper_length_mm,
```

Tidyr: Ordenar datos desordenados: tidyR

Pivottting: Cambiar el formato entre largo y ancho de una tabla de datos con las funciones: pivot_longer() y pivot_wider().



En nuestro ejemplo:

```
long_penguins <- penguins %>%
  pivot_longer(
    cols = c(species, island),
    names_to = "variable", values_to = "valor"
  )

long_penguins %>% glimpse
```

```
Rows: 688
Columns: 8
$ bill_length_mm      <dbl> 39.1, 39.1, 39.5, 39.5, 40.3, 40.3, NA, NA, 36.7, 36-
$ bill_depth_mm        <dbl> 18.7, 18.7, 17.4, 17.4, 18.0, 18.0, NA, NA, 19.3, 19-
$ flipper_length_mm   <int> 181, 181, 186, 186, 195, 195, NA, NA, 193, 193, 190, ~
$ body_mass_g          <int> 3750, 3750, 3800, 3800, 3250, 3250, NA, NA, 3450, 34-
$ sex                  <fct> male, male, female, female, female, female, NA, NA, ~
$ year                 <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007-
$ variable              <chr> "species", "island", "species", "island", "species", ~
$ valor                 <fct> Adelie, Torgersen, Adelie, Torgersen, Adelie, Torgers-
```

Ya no hay formato ordenado. La dim: 688 x 8

pivot_wider():

- invierte en efecto de pivot_longer()
- dim: 344 x 8

```
long_penguins %>%  
  pivot_wider(  
    names_from = "variable", values_from = "valor"  
  ) %>%  
  glimpse
```

```
Rows: 344  
Columns: 8  
$ bill_length_mm    <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1, ~  
$ bill_depth_mm     <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.1, ~  
$ flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, 186~  
$ body_mass_g       <int> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 3475, ~  
$ sex               <fct> male, female, female, NA, female, male, female, male~  
$ year              <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007~  
$ species           <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adel~  
$ island            <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torgerse~
```

Puedes encontrar más información acerca de pivot_*() en [pivoting vignette](#).

Tidyr: Ordenar datos desordenados: tidyR

Nesting: Agrupa datos similares de manera que cada grupo se convierte en una sola fila en un data frame.

```
nested_penguins <- penguins %>%
  nest(nested_data =
    c(island, bill_length_mm,
      bill_depth_mm, flipper_length_mm,
      body_mass_g, sex))
```

nested_penguins

```
# A tibble: 9 x 3
  species     year nested_data
  <fct>     <int> <list>
1 Adelie     2007 <tibble [50 x 6]>
2 Adelie     2008 <tibble [50 x 6]>
3 Adelie     2009 <tibble [52 x 6]>
4 Gentoo    2007 <tibble [34 x 6]>
5 Gentoo    2008 <tibble [46 x 6]>
6 Gentoo    2009 <tibble [44 x 6]>
7 Chinstrap  2007 <tibble [26 x 6]>
8 Chinstrap  2008 <tibble [18 x 6]>
9 Chinstrap  2009 <tibble [24 x 6]>
```

Tidyr: Ordenar datos desordenados: tidyR



- La función `nest()` genera datos anidados en un data frame con una fila por `species` y `year`.
- Los datos anidados `nested_data` por columnas contienen tibbles con seis columnas cada uno y un número de observaciones que pueden ser distintas.
- Los datos anidados son útiles si queremos aplicar funciones a cada subgrupo de datos (por ejemplo, comparar estadísticos por especie.)

Tidyr: Ordenar datos desordenados: tidyR



Rectangling: Deshace las estructuras de datos anidados (por ejemplo, JSON, HTML) y las lleva al formato de *datos ordenados*.

Extrae objetos individuales de una estructura de datos anidada mediante purrr::pluck().

```
nested_penguins %>% purrr::pluck("nested_data", 1)
```

```
# A tibble: 50 x 6
  island   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex
  <fct>     <dbl>        <dbl>          <int>       <int> <fct>
1 Torgersen    39.1         18.7           181        3750 male 
2 Torgersen    39.5         17.4           186        3800 female
3 Torgersen    40.3          18             195        3250 female
4 Torgersen     NA           NA              NA         NA <NA>
5 Torgersen    36.7         19.3           193        3450 female
6 Torgersen    39.3         20.6           190        3650 male 
7 Torgersen    38.9         17.8           181        3625 female
8 Torgersen    39.2         19.6           195        4675 male 
9 Torgersen    34.1         18.1           193        3475 <NA>
10 Torgersen    42            20.2           190        4250 <NA>
```

Tidyr: Ordenar datos desordenados: tidyR



Aplana las estructuras de datos anidados mediante tidyR::unnest().

```
nested_penguins %>% unnest(cols = c(nested_data))
```

```
# A tibble: 344 x 8
  species year island   bill_length_mm bill_depth_mm flipper_~1 body_~2 sex
  <fct>   <int> <fct>           <dbl>        <dbl>      <int>     <int> <fct>
1 Adelie    2007 Torgersen       39.1         18.7      181     3750 male 
2 Adelie    2007 Torgersen       39.5         17.4      186     3800 fema-
3 Adelie    2007 Torgersen       40.3          18       195     3250 fema-
4 Adelie    2007 Torgersen        NA           NA        NA      NA <NA>
5 Adelie    2007 Torgersen       36.7         19.3      193     3450 fema-
6 Adelie    2007 Torgersen       39.3         20.6      190     3650 male 
7 Adelie    2007 Torgersen       38.9         17.8      181     3625 fema-
8 Adelie    2007 Torgersen       39.2         19.6      195     4675 male 
9 Adelie    2007 Torgersen       34.1         18.1      193     3475 <NA>
10 Adelie   2007 Torgersen        42          20.2      190     4250 <NA>
# ... with 334 more rows, and abbreviated variable names 1: flipper_length_mm,
#   2: body_mass_g
```

Tidyr: Ordenar datos desordenados: tidyr

Extraer selectivamente componentes individuales de un objeto en una estructura de datos anidada mediante `tidyr::hoist()`.

```
nested_penguins %>% hoist(nested_data, hoisted_col = "bill_length_mm")
```

```
# A tibble: 9 x 4
  species      year hoisted_col nested_data
  <fct>     <int> <list>        <list>
1 Adelie     2007 <dbl [50]> <tibble [50 x 5]>
2 Adelie     2008 <dbl [50]> <tibble [50 x 5]>
3 Adelie     2009 <dbl [52]> <tibble [52 x 5]>
4 Gentoo    2007 <dbl [34]> <tibble [34 x 5]>
5 Gentoo    2008 <dbl [46]> <tibble [46 x 5]>
6 Gentoo    2009 <dbl [44]> <tibble [44 x 5]>
7 Chinstrap  2007 <dbl [26]> <tibble [26 x 5]>
8 Chinstrap  2008 <dbl [18]> <tibble [18 x 5]>
9 Chinstrap  2009 <dbl [24]> <tibble [24 x 5]>
```

Alternativamente, utilice `unnest_wider()` o `unnest_longer()` para tener más control sobre la operación de rectangulación.

Tidyr: Ordenar datos desordenados: tidy

Dividir y Combinar: Combina múltiples columnas en una sola columna.

```
penguins %>% unite(col = "specie_sex",
                      c(species, sex), sep = "_", remove = T)
```

```
# A tibble: 344 x 7
  specie_sex   island bill_length_mm bill_depth_mm flipper_~1 body_~2 year
  <chr>       <fct>      <dbl>        <dbl>      <int>     <int> <int>
1 Adelie_male Torgersen    39.1        18.7       181     3750  2007
2 Adelie_female Torgersen   39.5        17.4       186     3800  2007
3 Adelie_female Torgersen   40.3        18         195     3250  2007
4 Adelie_NA    Torgersen    NA          NA         NA      NA   2007
5 Adelie_female Torgersen   36.7        19.3       193     3450  2007
6 Adelie_male  Torgersen   39.3        20.6       190     3650  2007
7 Adelie_female Torgersen   38.9        17.8       181     3625  2007
8 Adelie_male  Torgersen   39.2        19.6       195     4675  2007
9 Adelie_NA    Torgersen    34.1        18.1       193     3475  2007
10 Adelie_NA   Torgersen    42          20.2       190     4250  2007
# ... with 334 more rows, and abbreviated variable names 1: flipper_length_mm,
#   2: body_mass_g
```

Tidyr: Ordenar datos desordenados: tidy



Separar una sola columna, que contiene varios valores, en varias columnas.

```
penguins %>% separate(bill_length_mm, sep = 2, into = c("cm", "mm"))
```

```
# A tibble: 344 x 9
  species island   cm     mm   bill_depth_mm flipper_len-1 body_-2 sex    year
  <fct>   <fct> <chr> <chr>      <dbl>        <int>    <int> <fct> <int>
1 Adelie  Torgersen 39   ".1"     18.7       181     3750 male   2007
2 Adelie  Torgersen 39   ".5"     17.4       186     3800 fema-  2007
3 Adelie  Torgersen 40   ".3"      18         195     3250 fema-  2007
4 Adelie  Torgersen <NA>  <NA>      NA        NA      NA <NA>  2007
5 Adelie  Torgersen 36   ".7"      19.3       193     3450 fema-  2007
6 Adelie  Torgersen 39   ".3"      20.6       190     3650 male   2007
7 Adelie  Torgersen 38   ".9"      17.8       181     3625 fema-  2007
8 Adelie  Torgersen 39   ".2"      19.6       195     4675 male   2007
9 Adelie  Torgersen 34   ".1"      18.1       193     3475 <NA>  2007
10 Adelie  Torgersen 42   ""        20.2       190     4250 <NA>  2007
# ... with 334 more rows, and abbreviated variable names 1: flipper_length_mm,
#   2: body_mass_g
```



Tidyr: Ordenar datos desordenados: tidy

Separar una sola columna, que contiene varios valores, en varias filas.

```
penguins %>% separate_rows(island, sep = "s", convert = T)
```

```
# A tibble: 564 x 8
  species island bill_length_mm bill_depth_mm flipper_len_mm body_mass_g sex     year
  <fct>   <chr>      <dbl>        <dbl>       <dbl>        <dbl> <fct>    <int>
1 Adelie  Torger       39.1         18.7        181       3750 male    2007
2 Adelie  en            39.1         18.7        181       3750 male    2007
3 Adelie  Torger       39.5         17.4        186       3800 female  2007
4 Adelie  en            39.5         17.4        186       3800 female  2007
5 Adelie  Torger       40.3          18          195       3250 female  2007
6 Adelie  en            40.3          18          195       3250 female  2007
7 Adelie  Torger       NA           NA           NA        NA <NA>    2007
8 Adelie  en            NA           NA           NA        NA <NA>    2007
9 Adelie  Torger       36.7          19.3        193       3450 female  2007
10 Adelie  en           36.7          19.3        193       3450 female  2007
# ... with 554 more rows, and abbreviated variable names 1: flipper_length_mm,
#   2: body_mass_g
```

Podemos separar basandonos en la coincidencia de caracteres

Tidyr: Ordenar datos desordenados: tidyR

Manejo de los valores perdidos: Elimina o sustituye los valores perdidos (NA).

Supongamos que tenemos una tibble y queremos hacer explícitos los valores perdidos implícitos.

incompl_penguins

```
# A tibble: 4 x 3
  species    year measurement
  <chr>     <dbl>      <dbl>
1 Adelie     2007       46.6
2 Adelie     2008       79.8
3 Gentoo    2008       31.4
4 Chinstrap  2007       NA
```

Tidyr: Ordenar datos desordenados: tidyr

Para hacer explícitos los NAs implícitos:

```
incompl_penguins %>%  
  complete(species, year, fill = list(measurement = NA))
```

```
# A tibble: 6 x 3  
  species    year measurement  
  <chr>     <dbl>      <dbl>  
1 Adelie     2007       46.6  
2 Adelie     2008       79.8  
3 Chinstrap  2007       NA  
4 Chinstrap  2008       NA  
5 Gentoo    2007       NA  
6 Gentoo    2008      31.4
```

Tidyr: Ordenar datos desordenados: tidyR

Para hacer implícitos los valores perdidos explícitos.

```
incompl_penguins %>%  
  drop_na(measurement)
```

```
# A tibble: 3 x 3  
  species   year measurement  
  <chr>     <dbl>      <dbl>  
1 Adelie    2007       46.6  
2 Adelie    2008       79.8  
3 Gentoo   2008       31.4
```

Tidyr: Ordenar datos desordenados: tidyR



Reemplaza los valores que faltan por el valor siguiente/anterior.

```
incompl_penguins %>%  
  fill(measurement, .direction = "down")
```

```
# A tibble: 4 x 3  
  species    year measurement  
  <chr>     <dbl>      <dbl>  
1 Adelie     2007       46.6  
2 Adelie     2008       79.8  
3 Gentoo    2008       31.4  
4 Chinstrap  2007       31.4
```

Tidyr: Ordenar datos desordenados: tidyR



Reemplaza los valores que faltan por un valor predefinido.

```
incompl_penguins %>%
  replace_na(replace = list(measurement = mean(. $measurement, na.rm = T)))
```

```
# A tibble: 4 x 3
  species    year measurement
  <chr>     <dbl>      <dbl>
1 Adelie    2007       46.6
2 Adelie    2008       79.8
3 Gentoo   2008       31.4
4 Chinstrap 2007       52.6
```

Tidyr: Ordenar datos desordenados: tidyr



Más información en [tidyr cheatsheet](#).

Nota: los argumentos de función precedidos por un punto en el tidyverse pueden tener una de estas dos razones:

- la función es todavía prematura, es decir, los desarrolladores aún piensan en la mejor manera de implementar y nombrar la función
- la función se aplica regularmente dentro de otra función para no confundir los argumentos de la función interna y la externa

Lección 6

dplyr: Una gramática para manipular datos

dplyr: Una gramática para manipular datos

dplyr proporciona un conjunto de funciones para manipular objetos de data frames (por ejemplo, tibbles), basándose en una gramática consistente.

Las funciones están representadas intuitivamente por “verbos” que reflejan las operaciones subyacentes y siempre dan como resultado un tibble nuevo o modificado.

Operaciones sobre las filas

- `filter()` selecciona las filas que cumplen uno o varios criterios lógicos
- `slice()` selecciona las filas en función de su ubicación en los datos
- `arrange()` cambia el orden de las filas

Ejemplos con filter()

- Filtrar todos los pingüinos de la especie “Adelie”.

```
penguins %>%  
  filter(species == "Adelie")
```

```
# A tibble: 152 x 8  
  species island   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex  year  
  <fct>   <fct>        <dbl>         <dbl>            <dbl>           <dbl>     <fct> <int>  
1 Adelie  Torgersen      39.1          18.7             181            3750 male    2007  
2 Adelie  Torgersen      39.5          17.4             186            3800 female  2007  
3 Adelie  Torgersen      40.3           18              195            3250 female  2007  
4 Adelie  Torgersen       NA            NA              NA            NA <NA>   2007  
5 Adelie  Torgersen      36.7          19.3             193            3450 female  2007  
6 Adelie  Torgersen      39.3          20.6             190            3650 male    2007  
7 Adelie  Torgersen      38.9          17.8             181            3625 female  2007  
8 Adelie  Torgersen      39.2          19.6             195            4675 male    2007  
9 Adelie  Torgersen      34.1          18.1             193            3475 <NA>   2007  
10 Adelie  Torgersen       42            20.2             190            4250 <NA>   2007  
# ... with 142 more rows, and abbreviated variable names 1: flipper_length_mm,  
#   2: body_mass_g
```

Ejemplos con filter()

- Filtrar todos los pingüinos con un valor perdido en la variable bill_length_mm.

```
penguins %>%  
  filter(is.na(bill_length_mm) == T)
```

```
# A tibble: 2 x 8  
species island    bill_length_mm bill_depth_mm flipper_l-1 body_~2 sex      year  
<fct>   <fct>           <dbl>        <dbl>       <int>     <int> <fct> <int>  
1 Adelie  Torgersen      NA          NA          NA        NA <NA>  2007  
2 Gentoo  Biscoe         NA          NA          NA        NA <NA>  2009  
# ... with abbreviated variable names 1: flipper_length_mm, 2: body_mass_g
```

```
# quitar los Nas  
#filter(!is.na(bill_length_mm) == F)
```

Ejemplos con filter()

- Filtrar todos los pingüinos observados antes del año 2008 o después del año 2008 y en los que la masa corporal está entre 3800 y 4000 gramos.

```
penguins %>%  
  filter(between(body_mass_g, 3800, 4000) & (year < 2008 | year > 2008))
```

```
# A tibble: 28 x 8  
  species island bill_length_mm bill_depth_mm flipper_-1 body_~2 sex     year  
  <fct>   <fct>      <dbl>        <dbl>       <int>    <int> <fct>    <int>  
1 Adelie  Torgersen     39.5         17.4        186    3800 female  2007  
2 Adelie  Torgersen     38.6         21.2        191    3800 male   2007  
3 Adelie  Biscoe        35.9         19.2        189    3800 female  2007  
4 Adelie  Biscoe        38.2         18.1        185    3950 male   2007  
5 Adelie  Biscoe        38.8         17.2        180    3800 male   2007  
6 Adelie  Biscoe        35.3         18.9        187    3800 female 2007  
7 Adelie  Biscoe        40.5         18.9        180    3950 male   2007  
8 Adelie  Dream          37.2         18.1        178    3900 male   2007  
9 Adelie  Dream          40.9         18.9        184    3900 male   2007  
10 Adelie  Dream         38.8          20          190    3950 male   2007  
# ... with 18 more rows, and abbreviated variable names 1: flipper_length_mm,  
#   2: body_mass_g
```

Ejemplos con slice()

- Elegir las filas de acuerdo a su índice.

```
penguins %>%  
  slice(23:27)
```

```
# A tibble: 5 x 8  
  species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex year  
  <fct>   <fct>     <dbl>        <dbl>            <dbl>       <dbl> <fct> <dbl>  
1 Adelie  Biscoe      35.9        19.2           189       3800 female  2007  
2 Adelie  Biscoe      38.2        18.1           185       3950 male   2007  
3 Adelie  Biscoe      38.8        17.2           180       3800 male   2007  
4 Adelie  Biscoe      35.3        18.9           187       3800 female  2007  
5 Adelie  Biscoe      40.6        18.6           183       3550 male   2007  
# ... with abbreviated variable names 1: flipper_length_mm, 2: body_mass_g
```

Ejemplos con slice()

- `slice_head()` selecciona las primeras n filas (viceversa para `slice_tail()`).

```
penguins %>%  
  slice_head(n = 5)
```

```
# A tibble: 5 x 8  
species island bill_length_mm bill_depth_mm flipper_l-1 body_~2 sex     year  
<fct> <fct>      <dbl>        <dbl>       <int>    <int> <fct> <int>  
1 Adelie  Torgersen     39.1        18.7       181    3750 male   2007  
2 Adelie  Torgersen     39.5        17.4       186    3800 fema~ 2007  
3 Adelie  Torgersen     40.3         18        195    3250 fema~ 2007  
4 Adelie  Torgersen      NA          NA         NA     NA <NA>  2007  
5 Adelie  Torgersen     36.7        19.3       193    3450 fema~ 2007  
# ... with abbreviated variable names 1: flipper_length_mm, 2: body_mass_g
```

```
# alternativamente:  
#slice_head(frac = 0.05)
```

Ejemplos con slice()

- Selecciona una muestra aleatoria de n filas (con o sin reemplazo).

```
penguins %>%  
  slice_sample(n = 5)
```

```
# A tibble: 5 x 8  
  species   island bill_length_mm bill_depth_mm flipper~1 body_~2 sex     year  
  <fct>     <fct>        <dbl>        <dbl>      <int>    <int> <fct> <int>  
1 Gentoo    Biscoe       43.5        15.2       213     4650 fema~  2009  
2 Gentoo    Biscoe       50.8        17.3       228     5600 male   2009  
3 Adelie    Biscoe       41.3        21.1       195     4400 male   2008  
4 Adelie    Torgersen    38.6         17        188     2900 fema~  2009  
5 Chinstrap Dream        51.3        18.2       197     3750 male   2007  
# ... with abbreviated variable names 1: flipper_length_mm, 2: body_mass_g
```

Ejemplos con slice()

- Selecciona las n filas con el valor más grande (viceversa para slice_min()).

```
penguins %>%  
  slice_max(bill_length_mm, n = 5)
```

```
# A tibble: 5 x 8  
  species   island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex   year  
  <fct>     <fct>        <dbl>        <dbl>            <dbl>       <dbl> <fct> <int>  
1 Gentoo    Biscoe      59.6         17          230       6050 male   2007  
2 Chinstrap  Dream      58           17.8        181       3700 female 2007  
3 Gentoo    Biscoe      55.9         17          228       5600 male   2009  
4 Chinstrap  Dream      55.8         19.8        207       4000 male   2009  
5 Gentoo    Biscoe      55.1         16          230       5850 male   2009  
# ... with abbreviated variable names 1: flipper_length_mm, 2: body_mass_g
```

Ejemplos con arrange()

- Selecciona los cinco pingüinos con menor masa corporal.

```
penguins %>%  
  arrange(body_mass_g) %>%  
  slice_head(n = 5)  # equivalentente a: slice_min(body_mass_g, n = 3)
```

```
# A tibble: 5 x 8  
species   island bill_length_mm bill_depth_mm flipper_le-1 body_~2 sex     year  
<fct>     <fct>      <dbl>        <dbl>       <int>    <int> <fct> <int>  
1 Chinstrap Dream        46.9        16.6       192    2700 fema-  2008  
2 Adelie    Biscoe       36.5        16.6       181    2850 fema-  2008  
3 Adelie    Biscoe       36.4        17.1       184    2850 fema-  2008  
4 Adelie    Biscoe       34.5        18.1       187    2900 fema-  2008  
5 Adelie    Dream        33.1        16.1       178    2900 fema-  2008  
# ... with abbreviated variable names 1: flipper_length_mm, 2: body_mass_g
```

- ¿Cómo seleccionarías a los 5 pingüinos con mayor masa corporal?

dplyr: Una gramática para manipular datos



Operaciones sobre las columnas

- `select()` selecciona o elimina determinadas columnas
- `rename()` cambia los nombres de las columnas
- `relocate()` cambia el orden de las columnas
- `mutate()` transforma los valores de las columnas y/o crea nuevas columnas

Ejemplos select()

- Selección por índice

```
penguins %>%  
  select(1:3) %>%  
  glimpse
```

```
Rows: 344  
Columns: 3  
$ species      <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie,~  
$ island        <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torgersen, ~  
$ bill_length_mm <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1, 42.~
```

Ejemplos select()

- Selección por nombre

```
penguins %>%  
  select(species, island, bill_length_mm) %>%  
  glimpse
```

```
Rows: 344  
Columns: 3  
$ species      <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie,~  
$ island        <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torgersen, ~  
$ bill_length_mm <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1, 42.~
```

Ejemplos select()

- Seleccionar todas las columnas

```
penguins %>%  
  select(everything()) %>%  
  glimpse
```

```
Rows: 344  
Columns: 8  
$ species      <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie,  
$ island        <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torgersen,  
$ bill_length_mm <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1, ~  
$ bill_depth_mm  <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.1, ~  
$ flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, 186~  
$ body_mass_g    <int> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 3475, ~  
$ sex            <fct> male, female, female, NA, female, male, female, male-  
$ year           <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007,
```

```
# select(last_col())
```

Ejemplos select()

- Seleccionar las columnas cuyos nombres empiezan por un patrón específico

```
penguins %>%
  select(starts_with("bill")) %>%
  glimpse
```

```
Rows: 344
Columns: 2
$ bill_length_mm <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1, 42.~
$ bill_depth_mm  <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.1, 20.~
```

```
# ends_with()
```

Ejemplos select()

```
penguins %>%  
  select(contains("e") & contains("a")) %>%  
  glimpse
```

```
Rows: 344  
Columns: 1  
$ year <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007,~
```

Ejemplos select()

Seleccionar columnas en base a una expresión regular (regex)

```
penguins %>%
  select(matches("_\\w*_mm$")) %>%
  glimpse
```

```
Rows: 344
Columns: 3
$ bill_length_mm    <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1, ~
$ bill_depth_mm     <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.1, ~
$ flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, 186~
```

Ejemplos select()

```
penguins %>%  
  select(where(is.numeric)) %>%  
  glimpse
```

```
Rows: 344  
Columns: 5  
$ bill_length_mm    <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1, ~  
$ bill_depth_mm     <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.1, ~  
$ flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, 186~  
$ body_mass_g        <int> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 3475, ~  
$ year              <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007,
```

Ejemplos select()

¿Qué columnas devuelven las siguientes consultas?

```
penguins %>%  
  select(ends_with("mm"))
```

```
penguins %>%  
  select(-contains("mm"))
```

```
penguins %>%  
  select(where(~ is.numeric(.))) %>%  # select(where(is.numeric))  
  select(where(~ mean(., na.rm = T) > 1000))
```

Ejemplos rename()

- Cambiar el nombre de la columna body_mass_g (sex) a bm (gender).

```
penguins %>% rename(bm = body_mass_g, gender = sex) %>%  
  colnames()
```

```
[1] "species"           "island"            "bill_length_mm"  
[4] "bill_depth_mm"    "flipper_length_mm" "bm"  
[7] "gender"             "year"
```

- Cambiar los nombres de las columnas que incluyen "mm" a mayúsculas.

```
penguins %>% rename_with(.fn = toupper, .cols = contains("mm")) %>%  
  colnames()
```

```
[1] "species"           "island"            "BILL_LENGTH_MM"  
[4] "BILL_DEPTH_MM"    "FLIPPER_LENGTH_MM" "body_mass_g"  
[7] "sex"                "year"
```

Ejemplos relocate()

- Cambiar el orden de las columnas en la tibble de acuerdo al siguiente esquema:

- ① colocar “especie” después de “masa corporal”.
- ② colocar “sexo” antes de “especie”.
- ③ colocar “isla” al final

```
penguins %>%  
  relocate(species, .after = body_mass_g) %>%  
  relocate(sex, .before = species) %>%  
  relocate(island, .after = last_col()) %>%  
  colnames()
```

```
[1] "bill_length_mm"      "bill_depth_mm"       "flipper_length_mm"  
[4] "body_mass_g"         "sex"                 "species"  
[7] "year"                "island"
```

Ejemplos `mutate()`

Crear una nueva variable `bm_kg` que ponga `body_mass_g` en kilogramos.

```
penguins %>%
  mutate(bm_kg = body_mass_g / 1000, .keep = "all", .after = island) %>%
  slice_head(n = 5)
```

```
# A tibble: 5 x 9
  species island   bm_kg bill_length_mm bill_dept-1 flipp-2 body_~3 sex     year
  <fct>   <fct>    <dbl>      <dbl>       <dbl>   <int> <int> <fct> <int>
1 Adelie  Torgersen 3.75      39.1       18.7     181    3750 male   2007
2 Adelie  Torgersen 3.8       39.5       17.4     186    3800 fema~  2007
3 Adelie  Torgersen 3.25      40.3       18       195    3250 fema~  2007
4 Adelie  Torgersen NA        NA         NA       NA     NA <NA>   2007
5 Adelie  Torgersen 3.45      36.7       19.3     193    3450 fema~  2007
# ... with abbreviated variable names 1: bill_depth_mm, 2: flipper_length_mm,
# 3: body_mass_g
```

- Usa `.keep` para especificar las columnas que se mantendrán después de la manipulación.
- Usa `.before/.after` para especificar la posición de la nueva columna.
- Para anular una columna dada simplemente utiliza el mismo nombre de columna.
- Para mantener sólo la nueva columna utiliza `dplyr::transmute()`.

Ejemplos `mutate()`

Codificación de una variable categórica con “C” niveles de factor en “C” dummies (a menudo en la modelización se crean “C-1” dummies).

```
penguins %>%
  mutate(
    sex_binary = case_when(
      sex == "male" ~ 1,
      sex == "female" ~ 0),
    .keep = "all", .after = island
  ) %>%
  slice_head(n = 3)
```

```
# A tibble: 3 x 9
  species island   sex_binary bill_length~1 bill_~2 flipp~3 body_~4 sex     year
  <fct>   <fct>       <dbl>        <dbl>    <dbl>    <int>    <int> <fct> <int>
1 Adelie  Torgersen     1         39.1     18.7     181     3750 male   2007
2 Adelie  Torgersen     0         39.5     17.4     186     3800 fema~  2007
3 Adelie  Torgersen     0         40.3     18       195     3250 fema~  2007
# ... with abbreviated variable names 1: bill_length_mm, 2: bill_depth_mm,
#   3: flipper length mm, 4: body mass g
```

Ejemplos `mutate()`

`case_when`: - Versión vectorizada de `if_else`

- Fórmulas de dos lados: El LHS comprueba la condición, el RHS especifica el valor de sustitución
- Para los casos no coincidentes, la función devuelve NA
- Utiliza el LHS TRUE para capturar todos los casos no especificados explícitamente de antemano

Ejemplos mutate()

- Transforma las variables de medidas a metros

```
penguins %>%
  mutate(
    across(contains("mm"), ~ . / 1000),
    .keep = "all"
  ) %>%
  slice_head(n = 3)
```

```
# A tibble: 3 x 8
  species island   bill_length_mm bill_depth_mm flipper_l-1 body_~2 sex      year
  <fct>   <fct>        <dbl>        <dbl>       <dbl>     <int> <fct> <int>
1 Adelie  Torgersen     0.0391      0.0187      0.181     3750 male   2007
2 Adelie  Torgersen     0.0395      0.0174      0.186     3800 fema~  2007
3 Adelie  Torgersen     0.0403      0.018       0.195     3250 fema~  2007
# ... with abbreviated variable names 1: flipper_length_mm, 2: body_mass_g
```

Ejemplos `mutate()`

across:

- Aplicar la misma transformación en varias columnas
- Te permite utilizar la semántica que conoces de la función `select()`.
- No requiere que se especifique explícitamente un nombre de columna, ya que sólo transforma las columnas existentes

Ejemplos `mutate()`

- Define `species`, `island` y `sex` como variables categóricas, es decir *fatores*, usando `across()`.

```
penguins %>%
  mutate(
    across(where(is.character), as.factor),
    .keep = "all"
  ) %>%
  slice_head(n = 3)
```

```
# A tibble: 3 x 8
  species island bill_length_mm bill_depth_mm flipper_l-1 body_~2 sex     year
  <fct>   <fct>      <dbl>        <dbl>       <int>     <int> <fct> <int>
1 Adelie  Torgersen     39.1         18.7       181     3750 male   2007
2 Adelie  Torgersen     39.5         17.4       186     3800 fema-  2007
3 Adelie  Torgersen     40.3          18        195     3250 fema-  2007
# ... with abbreviated variable names 1: flipper_length_mm, 2: body_mass_g
```

dplyr: Una gramática para manipular datos



Operaciones sobre datos agrupados

- `group_by()` divide los datos en función de una o varias columnas
- `summarise()` reduce un grupo de datos en una sola fila

Ejemplos group_by

- Partición de los datos en una o varias columnas

```
penguins %>% group_by(species)
```

```
# A tibble: 344 x 8
# Groups:   species [3]
  species island bill_length_mm bill_depth_mm flipper_~1 body_~2 sex     year
  <fct>   <fct>      <dbl>        <dbl>       <int>    <int> <fct> <int>
1 Adelie  Torgersen     39.1         18.7       181     3750 male   2007
2 Adelie  Torgersen     39.5         17.4       186     3800 fema~  2007
3 Adelie  Torgersen     40.3          18        195     3250 fema~  2007
4 Adelie  Torgersen      NA           NA         NA      NA <NA>  2007
5 Adelie  Torgersen     36.7         19.3       193     3450 fema~  2007
6 Adelie  Torgersen     39.3         20.6       190     3650 male   2007
7 Adelie  Torgersen     38.9         17.8       181     3625 fema~  2007
8 Adelie  Torgersen     39.2         19.6       195     4675 male   2007
9 Adelie  Torgersen     34.1         18.1       193     3475 <NA>  2007
10 Adelie Torgersen      42           20.2       190     4250 <NA>  2007
# ... with 334 more rows, and abbreviated variable names 1: flipper_length_mm,
#   2: body_mass_g
```

Utiliza `group_keys()`, `group_indices()` y `group_vars()` para acceder a las claves de agrupación, los índices de grupo por fila y las variables de agrupación.

Ejemplos group_by

`group_by()` cambia la representación del tibble y lo transforma en un data frame agrupado (`grouped_df`).

Esto nos permite operar en los subgrupos individualmente usando `summarise()`.

`summarise()` reduce un grupo de datos en una sola fila

Ejemplos group_by

```
penguins %>% group_by(species) %>%  
  summarise(count = n(), .groups = "drop")
```

```
# A tibble: 3 x 2  
  species    count  
  <fct>     <int>  
1 Adelie      152  
2 Chinstrap    68  
3 Gentoo      124
```

Ejemplos summarise()

```
penguins %>% group_by(species, sex) %>% summarise(count = n(), .groups = "drop")
```

```
# A tibble: 8 x 3
  species   sex   count
  <fct>     <fct> <int>
1 Adelie   female    73
2 Adelie   male     73
3 Adelie   <NA>      6
4 Chinstrap female   34
5 Chinstrap male    34
6 Gentoo   female   58
7 Gentoo   male     61
8 Gentoo   <NA>      5
```

Ejemplos summarise()

```
penguins %>%
  group_by(species) %>%
  summarise(
    across(contains("mm"), ~ mean(., na.rm = T), .names = "{.col}_avg"),
    .groups = "drop"
  )
```

```
# A tibble: 3 x 4
  species bill_length_mm_avg bill_depth_mm_avg flipper_length_mm_avg
  <fct>      <dbl>           <dbl>              <dbl>
1 Adelie     38.8            18.3               190.
2 Chinstrap   48.8            18.4               196.
3 Gentoo     47.5            15.0               217.
```

El uso de `group_by()`, seguido de `summarise()` y `ungroup()` refleja el paradigma **dividir-aplicar-combinar** del análisis de datos: Dividir los datos en particiones, aplicar alguna función a los datos y luego combinar los resultados.

Ejemplos summarise()

Nota: En lugar de utilizar ungroup() puedes poner el argumento .groups en summarise() igual a “drop”.

Utiliza .add = T para añadir nuevas variables de agrupación (si no, se anula la primera)

```
penguins %>%
  group_by(species) %>%
  group_by(year, .add = T) # equivalente a: group_by(species, year)
```

```
# A tibble: 344 x 8
# Groups:   species, year [9]
  species island bill_length_mm bill_depth_mm flipper_mm body_mm sex     year
  <fct>   <fct>      <dbl>        <dbl>       <int>    <int> <fct> <int>
1 Adelie  Torgersen     39.1         18.7       181     3750 male   2007
2 Adelie  Torgersen     39.5         17.4       186     3800 fema-  2007
3 Adelie  Torgersen     40.3          18        195     3250 fema-  2007
4 Adelie  Torgersen      NA           NA          NA      NA <NA>  2007
5 Adelie  Torgersen     36.7         19.3       193     3450 fema-  2007
6 Adelie  Torgersen     39.3         20.6       190     3650 male   2007
7 Adelie  Torgersen     38.9         17.8       181     3625 fema-  2007
8 Adelie  Torgersen     39.2         19.6       195     4675 male   2007
9 Adelie  Torgersen     34.1         18.1       193     3475 <NA>  2007
10 Adelie Torgersen      42           20.2       190     4250 <NA>  2007
# ... with 334 more rows, and abbreviated variable names 1: flipper_length_mm,
```

Ejemplos summarise()

```
penguins %>%
  group_by(species) %>%
  summarise(
    across(
      contains("mm"),
      list(avg = ~ mean(., na.rm = T), sd = ~ sd(., na.rm = T)),
      .names = "{.col}_{.fn}"
    ),
    .groups = "drop"
  )
```

```
# A tibble: 3 x 7
  species bill_length_mm_avg bill_length_mm_sd bill_~1 bill_~2 flipp~3 flipp~4
  <fct>          <dbl>           <dbl>   <dbl>   <dbl>   <dbl>
1 Adelie        38.8            2.66    18.3    1.22    190.    6.54
2 Chinstrap     48.8            3.34    18.4    1.14    196.    7.13
3 Gentoo        47.5            3.08    15.0    0.981   217.    6.48
# ... with abbreviated variable names 1: bill_depth_mm_avg,
#   2: bill_depth_mm_sd, 3: flipper_length_mm_avg, 4: flipper_length_mm_sd
```

Ejemplos summarise()

Cambios en el comportamiento de mutate(): Las funciones de resumen, por ejemplo, mean() o sd() operan en particiones de los datos en lugar de en los datos completos

```
penguins %>%  
  group_by(species) %>%  
  mutate(stand_bm = (body_mass_g - mean(body_mass_g, na.rm = T))  
        / sd(body_mass_g, na.rm = T)) %>%  
  glimpse
```

```
Rows: 344  
Columns: 9  
Groups: species [3]  
#> species      <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie,  
#> island       <fct> Torgersen, Torgersen, Torgersen, Torgersen,  
#> bill_length_mm <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1, ~  
#> bill_depth_mm  <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.1, ~  
#> flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, 186~  
#> body_mass_g    <int> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 3475, ~  
#> sex            <fct> male, female, female, NA, female, male, female, male~  
#> year           <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007~  
#> stand_bm       <dbl> 0.107591350, 0.216626878, -0.982763938, NA, -0.54662~
```

Ejemplos summarise()

- Calcular franjas para la masa corporal de acuerdo a la cantidad de desviaciones estándar de la media. Agrupar los datos según estos intervalos.

```
bm_breaks <- mean(penguins$body_mass_g,  
                    na.rm = T) - (-3:3) *  
sd(penguins$body_mass_g,na.rm = T)  
  
penguins %>%  
group_by(species, bm_bin = cut(body_mass_g, breaks = bm_breaks)) %>%  
summarise(count = n(), .groups = "drop")
```

Ejemplos summarise()

```
# A tibble: 12 x 3
  species   bm_bin      count
  <fct>     <fct>      <int>
1 Adelie   (2.6e+03,3.4e+03]    39
2 Adelie   (3.4e+03,4.2e+03]    87
3 Adelie   (4.2e+03,5e+03]     25
4 Adelie   <NA>                 1
5 Chinstrap (2.6e+03,3.4e+03]    11
6 Chinstrap (3.4e+03,4.2e+03]    50
7 Chinstrap (4.2e+03,5e+03]      7
8 Gentoo    (3.4e+03,4.2e+03]     6
9 Gentoo    (4.2e+03,5e+03]     56
10 Gentoo   (5e+03,5.81e+03]    52
11 Gentoo   (5.81e+03,6.61e+03]   9
12 Gentoo   <NA>                 1
```

Ejemplos summarise()

- Filtrar en las particiones en lugar de en la totalidad de los datos

```
penguins %>%  
  group_by(species, island) %>%  
  filter(flipper_length_mm == max(flipper_length_mm, na.rm = T))
```

```
# A tibble: 5 x 8  
# Groups:   species, island [5]  
  species   island   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex   year  
  <fct>     <fct>        <dbl>        <dbl>          <dbl>            <dbl> <fct> <int>  
1 Adelie    Dream       40.8        18.9          208         4300 male   2008  
2 Adelie    Biscoe      41          20            203         4725 male   2009  
3 Adelie    Torgersen   44.1        18            210         4000 male   2009  
4 Gentoo   Biscoe      54.3        15.7          231         5650 male   2008  
5 Chinstrap Dream       49          19.6          212         4300 male   2009  
# ... with abbreviated variable names 1: flipper_length_mm, 2: body_mass_g
```

Ejemplos `summarise()`

- Utilizar `group_by()` seguido de `nest()` para producir un data frame anidado

```
penguins %>%  
  group_by(species, year) %>%  
  tidyverse::nest()
```

```
# A tibble: 9 x 3  
# Groups:   species, year [9]  
  species   year data  
  <fct>     <int> <list>  
1 Adelie     2007 <tibble [50 x 6]>  
2 Adelie     2008 <tibble [50 x 6]>  
3 Adelie     2009 <tibble [52 x 6]>  
4 Gentoo    2007 <tibble [34 x 6]>  
5 Gentoo    2008 <tibble [46 x 6]>  
6 Gentoo    2009 <tibble [44 x 6]>  
7 Chinstrap  2007 <tibble [26 x 6]>  
8 Chinstrap  2008 <tibble [18 x 6]>  
9 Chinstrap  2009 <tibble [24 x 6]>
```

Puedes encontrar más información de `group_by()` ejecutando `vignette("grouping")`.

dplyr: Una gramática para manipular datos

Otras operaciones con dplyr

`distinct()` selecciona sólo filas únicas

```
penguins %>%  
  distinct(species, island)
```

```
# A tibble: 5 x 2  
  species     island  
  <fct>      <fct>  
1 Adelie     Torgersen  
2 Adelie     Biscoe  
3 Adelie     Dream  
4 Gentoo    Biscoe  
5 Chinstrap  Dream
```

`pull()` extrae columnas individuales como vectores

```
penguins %>%  
  pull(year)  # equivalente a: penguins$year
```

dplyr: Una gramática para manipular datos

if_else() sentencia if-else vectorizada.

```
penguins %>% select(species, island, body_mass_g) %>%
  mutate(penguin_size = if_else(body_mass_g < 3500,
                                "tiny penguin",
                                "big penguin"))
```

```
# A tibble: 344 x 4
  species   island body_mass_g penguin_size
  <fct>     <fct>    <int> <chr>
1 Adelie   Torgersen      3750 big penguin
2 Adelie   Torgersen      3800 big penguin
3 Adelie   Torgersen      3250 tiny penguin
4 Adelie   Torgersen        NA <NA>
5 Adelie   Torgersen      3450 tiny penguin
6 Adelie   Torgersen      3650 big penguin
7 Adelie   Torgersen      3625 big penguin
8 Adelie   Torgersen      4675 big penguin
9 Adelie   Torgersen      3475 tiny penguin
10 Adelie  Torgersen       4250 big penguin
# ... with 334 more rows
```

dplyr: Una gramática para manipular datos

lag() desplaza los valores de las columnas n hacia adelante

```
penguins %>% select(species, body_mass_g) %>%
  mutate(lagged_bm = lag(body_mass_g, n = 1))
```

```
# A tibble: 344 x 3
  species body_mass_g lagged_bm
  <fct>     <int>      <int>
1 Adelie     3750        NA
2 Adelie     3800      3750
3 Adelie     3250      3800
4 Adelie       NA      3250
5 Adelie     3450        NA
6 Adelie     3650      3450
7 Adelie     3625      3650
8 Adelie     4675      3625
9 Adelie     3475      4675
10 Adelie    4250      3475
# ... with 334 more rows
```

Lo utilizaremos cuando estudiemos series temporales

dplyr: Una gramática para manipular datos

Combinar diferentes data frames haciendo coincidir las filas en función de las claves (de forma similar a las uniones realizadas en SQL)



The diagram shows two data frames, **a** and **b**, being joined. Data frame **a** has columns **x1** and **x2** with rows A, B, and C. Data frame **b** has columns **x1** and **x3** with rows A, B, and D. A plus sign (+) indicates the join operation, and an equals sign (=) indicates the resulting combined data frame.

Mutating Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NA

dplyr::left_join(a, b, by = "x1")
Join matching rows from b to a.

x1	x3	x2
A	T	1
B	F	2
D	T	NA

dplyr::right_join(a, b, by = "x1")
Join matching rows from a to b.

x1	x2	x3
A	1	T
B	2	F

dplyr::inner_join(a, b, by = "x1")
Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F

dplyr::full_join(a, b, by = "x1")
Join data. Retain all values, all rows.

x1	x2	x3
A	1	T
B	2	F
C	3	NA
D	NA	T

Visualizar los pipes

En el enlace de [Tidydata](#)tutor puedes escribir código R y Tidyverse en tu navegador y ver cómo cambia el data frame en cada paso del pipeline que has escrito.

Aquí os dejo el enlace para visualizar las instrucciones más utilizadas de tidyverse con el ejemplo de los pingüinos:

- [arrange\(\)](#).
- [filter\(\)](#)
- [mutate\(\)](#)
- [select\(\)](#)
- [group_by\(\) %>% slice\(\)](#)
- [group_by\(\) %>% summarize\(\)](#)

Lección 7

Visualización de datos con ggplot2

Visualización de datos con ggplot2



ggplot2 es un sistema para crear gráficos de forma declarativa, basado en [The Grammar of Graphics](#). Permite producir “**gráficos elegantes para el análisis de datos**”

La sintaxis de ggplot2 ayuda a pensar en gráficos de una manera nueva y más general que R base. Facilita generar detalles importantes de los gráficos como las leyendas, los ejes, los colores, son fáciles de resolver en comparación con R base.

En [R gallery](#), [R charts](#) y en [ggplot2 extensions](#) puedes encontrar muchos ejemplos de gráficos con los códigos.

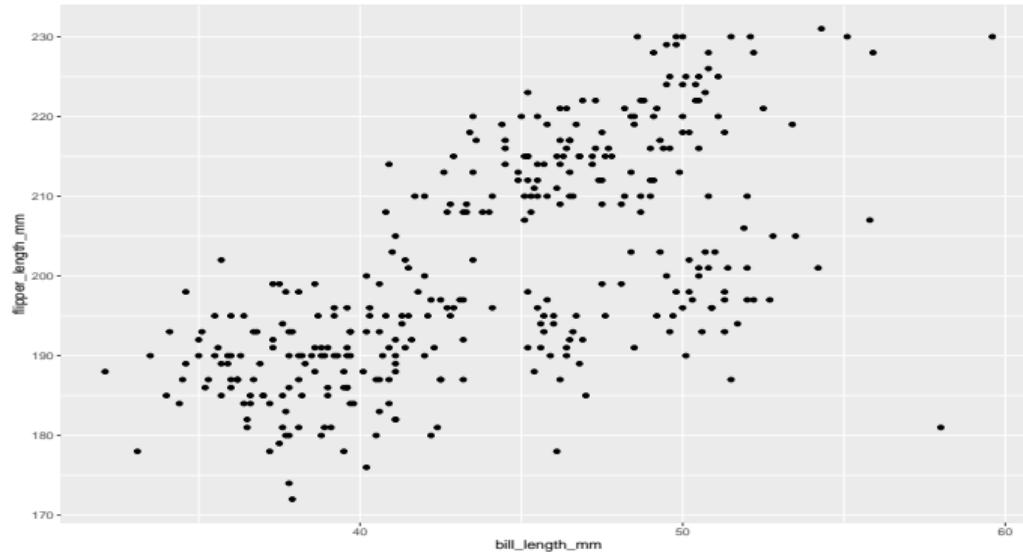
Gramática básica de ggplot2

Un ggplot necesita al menos tres cosas que hay que especificar:

- **Datos:** Normalmente un tibble del que se seleccionan las variables a visualizar. Siempre empezamos con `ggplot(data = df)` que le dice a `{ggplot2}` que vamos a trabajar con los datos `df`.
- **Estética:** Propiedades visuales que deseamos tenga nuestro gráfico. Por ejemplo, si deseamos visualizar la relación entre dos variables de `df`: `var1` en el eje x y `var2` en el eje y, debemos especificar: `aes(x = var1, y = var2)`.
- **Geometría:** Forma geométrica que deseamos utilizar para representar los datos: puntos, líneas continuas, curvas, entre otras. Se especifica con `geom_*`(). Por ejemplo, `geom_point()` para hacer un diagrama de puntos.

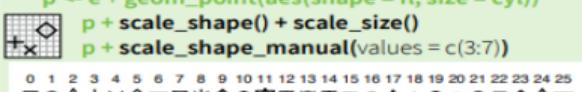
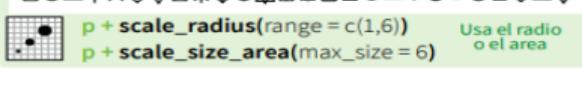
Ejemplo con la gramática básica

```
penguins %>%  
  ggplot(aes(x=bill_length_mm, y = flipper_length_mm)) +  
  geom_point(na.rm = TRUE)
```



Más elementos de la gramática ...

- **Escalas:** Permiten anular los valores por defecto de una estética para especificar otros. La sintaxis es `scale_*`(). Existen escalas para asignar valores de manera discreta, continua o manual. También escalas de localización, para color, relleno; tamaño y figuras.

<p>Escalas para todo uso Uselas con la mayoría de las estéticas</p> <p><code>scale_*_continuous()</code> - asigna valores continuos a visuales <code>scale_*_discrete()</code> - asigna valores discretos a visuales <code>scale_*_identity()</code> - crea una estética visual por cada valor <code>scale_*_manual(values = c())</code> - asigna valores específicos a valores visuales escogidos manualmente. <code>scale_*_date(date_labels = "%m/%d")</code>, <code>date_breaks = "2 weeks"</code>) - Usa los valores como fechas <code>scale_*_datetime()</code> - Usa los valores como fecha-horas igual que <code>scale_*_date</code> pero usando <code>strptime</code></p>	<p>Escalas de localización para X e Y Use con las estéticas x e y (aquí se muestra x)</p> <p><code>scale_x_log10()</code> - Usa escala logarítmica base 10 <code>scale_x_reverse()</code> - Posiciona x al revés <code>scale_x_sqrt()</code> - Usa escala raíz cuadrada</p>
<p>Escalas para Color y Relleno (Continuas)</p> <ul style="list-style-type: none"> o <- c + geom_dotplot(aes(fill = ..x..)) o + <code>scale_fill_distiller(palette = "Blues")</code> o + <code>scale_fill_gradient(low="red", high="yellow")</code> o + <code>scale_fill_gradient2(low="red", high="blue", mid = "white", midpoint = 25)</code> o + <code>scale_fill_gradientn(colours=topo.colors(6))</code> También: <code>rainbow()</code>, <code>heat.colors()</code>, <code>terrain.colors()</code>, <code>cm.colors()</code>, <code>RColorBrewer::brewer.pal()</code> 	<p>Escalas para Color y Relleno (Discretas) <code>n <- d + geom_bar(aes(fill = fl))</code></p> <p><code>n + scale_fill_brewer(palette = "Blues")</code> Ver opciones de colores: <code>RColorBrewer::display.brewer.all()</code></p> <p><code>n + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red")</code></p> 
<p>Escalas que usan tamaño y figuras</p> <p><code>p <- e + geom_point(aes(shape = fl, size = cyl))</code></p>  <p><code>p + scale_shape() + scale_size()</code> <code>p + scale_shape_manual(values = c(3:7))</code></p> <p><code>p + scale_radius(range = c(1,6))</code> Usa el radio o el área <code>p + scale_size_area(max_size = 6)</code></p> 	

Más elementos de la gramática . . .

- **Resúmenes (Stats):** Permiten construir nuevas variables de resumen para hacer un gráfico. Por ejemplo: conteo, cuantiles, proporciones, curvas ajustadas. Se especifica con `stat_*`.

`c + stat_bin(binwidth = 1, origin = 10)` **Distribución Unidimensional**
`x, y | ..count.., ..ncount.., ..density.., ..ndensity..`

`c + stat_count(width = 1)` `x, y, | ..count.., ..prop..`
`c + stat_density(adjust = 1, kernel = "gaussian")`
`x, y, | ..count.., ..density.., ..scaled..`

`e + stat_bin_2d(bins = 30, drop = T)` **Distribución Bidimensional**
`x, y, fill | ..count.., ..density..`

`e + stat_bin_hex(bins=30)` `x, y, fill | ..count.., ..density..`
`e + stat_density_2d(contour = TRUE, n = 100)`
`x, y, color, size | ..level..`
`e + stat_ellipse(level = 0.95, segments = 51, type = "t")`

`f + stat_boxplot(coef = 1.5)` **Comparativas**
`x, y | ..lower.., ..middle.., ..upper.., ..width.., ..ymin.., ..ymax..`

`f + stat_ydensity(kernel = "gaussian", scale = "area")`
`x, y | ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..`

`ggplot() + stat_function(aes(x = -3:3), n = 99,`
`fun = dnorm, args = list(sd=0.5))` `x | ..x.., ..y..`

`e + stat_identity(na.rm = TRUE)`

`ggplot() + stat_qq(aes(sample=1:100), dist = qt,`
`dparam=list(df=5))` `sample, x, y | ..sample.., ..theoretical..`

`e + stat_sum()` `x, y, size | ..n.., ..prop..`

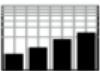
`e + stat_summary(fun.data = "mean_cl_boot")`

`h + stat_summary_bin(fun.y = "mean", geom = "bar")`

`e + stat_unique()` **Todo Uso**

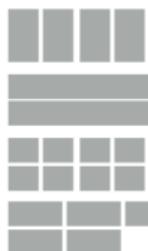
Más elementos de la gramática ...

- **Sistema de coordenadas:** Permite especificar cómo deseamos las coordenadas del gráfico. La sintaxis es `coord_*`.

	<pre>r <- d + geom_bar() r + coord_cartesian(xlim = c(0, 5)) xlim, ylim Usa coordenadas cartesianas</pre>
	<pre>r + coord_fixed(ratio = 1/2) ratio, xlim, ylim Se fija la relación de aspecto</pre>
	<pre>r + coord_flip() xlim, ylim Las coordenadas son volteadas</pre>
	<pre>r + coord_polar(theta = "x", direction=1) theta, start, direction Coordenadas polares</pre>
	<pre>r + coord_trans(ytrans = "sqrt") xtrans, ytrans, limx, limy xtrans e ytrans se asignan a funciones ventanas para transformar las coordenadas cartesianas</pre>

Más elementos de la gramática . . .

- **Facetas:** Dividen una gráfica en múltiple subgráficas en base a una o varias variables discretas. La sintaxis es `facet_*`.



`t + facet_grid(. ~ fl)`
usa fl para dividir en columnas
`t + facet_grid(year ~ .)`
usa year para dividir en líneas
`t + facet_grid(year ~ fl)`
usa los dos para dividir
`t + facet_wrap(~ fl)`
divide de una manera rectangular

Use `scales` para que dejar que el límite cambie por cada faceta

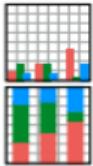
`t + facet_grid(drv ~ fl, scales = "free")`
Cada faceta tiene límites x e y independientes

- `"free_x"` - ajusta el límite del eje x
- `"free_y"` - ajusta el límite del eje y

Más elementos de la gramática . . .

- **Ajuste de las posiciones:** Permite indicar qué hacer con geoms que ocuparían la misma posición en la gráfica

```
s <- ggplot(mpg, aes(fl, fill = drv))
```



```
s + geom_bar(position = "dodge")
```

Pone los elementos a lado de cada uno

```
s + geom_bar(position = "fill")
```

Pone los elementos encima the cada uno
y usa toda la altura de la gráfica

```
e + geom_point(position = "jitter")
```

Agrega ruido a los elementos

```
e + geom_label(position = "nudge")
```

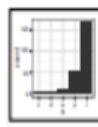
Empuja las letras para ver los puntos

```
s + geom_bar(position = "stack")
```

Pone los elementos encima the cada uno

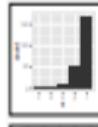
Más elementos de la gramática . . .

- **Tema:** Valores visuales generales de un gráfico, como el fondo, las cuadrículas, los ejes, el tipo de letra predeterminado, los tamaños y los colores.



`r + theme_bw()`

Fondo blanco con
cuadrícula



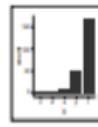
`r + theme_gray()`

Fondo gris
(tema inicial)



`r + theme_dark()`

Obscuro



`r + theme_classic()`

`r + theme_light()`



`r + theme_linedraw()`

`r + theme_minimal()`

Temas minimalisticos



`r + theme_void()`

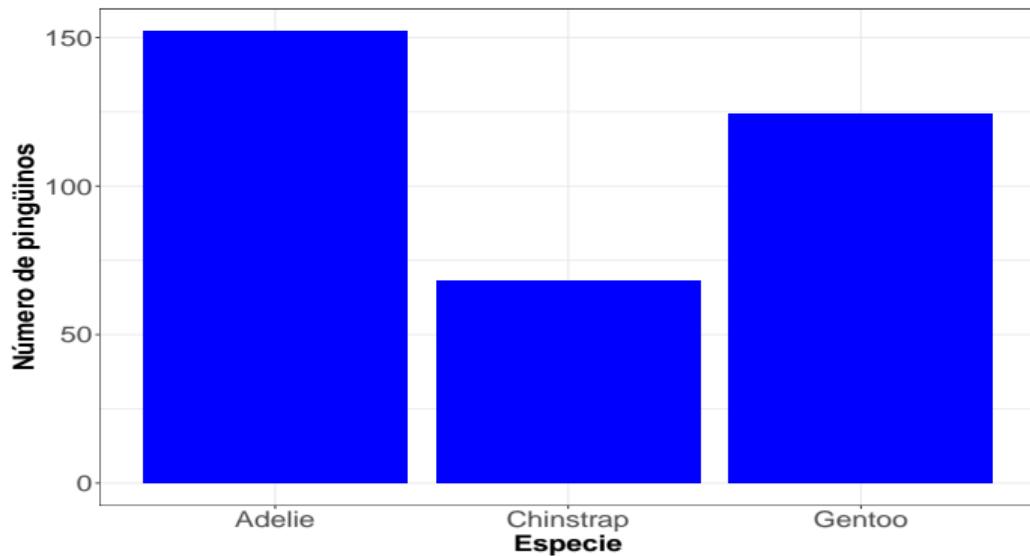
Tema vacío

Visualizando una variable cualitativa nominal

Vamos a visualizar una variable cualitativa nominal, con un tema que no es el que trae por defecto ggplot con fondo gris

```
penguins %>%
  ggplot(aes(x = species)) +
  geom_bar(fill="blue") +
  labs(x="Especie", y="Número de pingüinos") +
  theme_bw() +
  theme(axis.text = element_text(size=20),
        axis.title = element_text(size=20, face = "bold"))
```

Visualizando una variable cualitativa

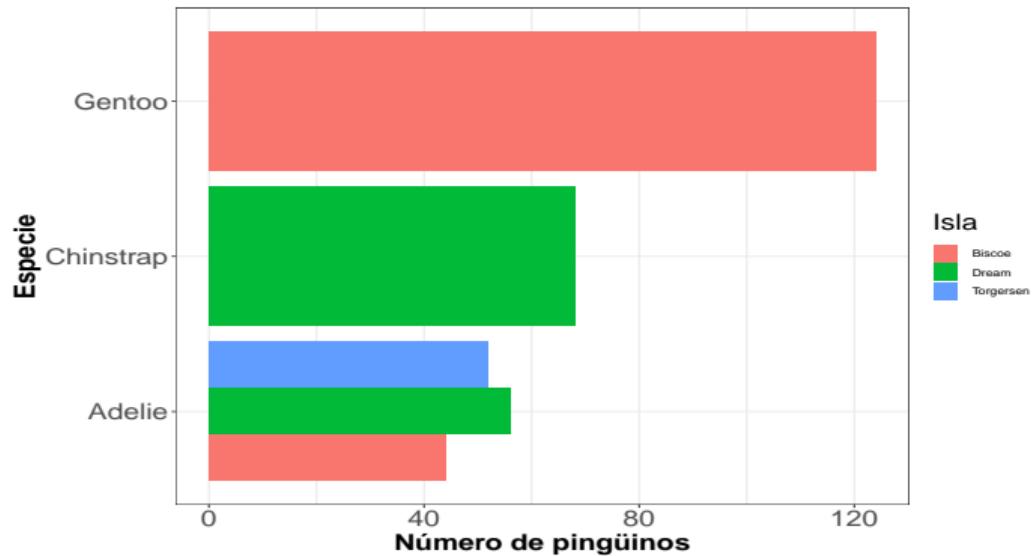


Cruzando dos variables cualitativas

Para observar la distribución conjunta de dos variables cualitativas hay que mapearlas a diferentes aspectos gráficos.

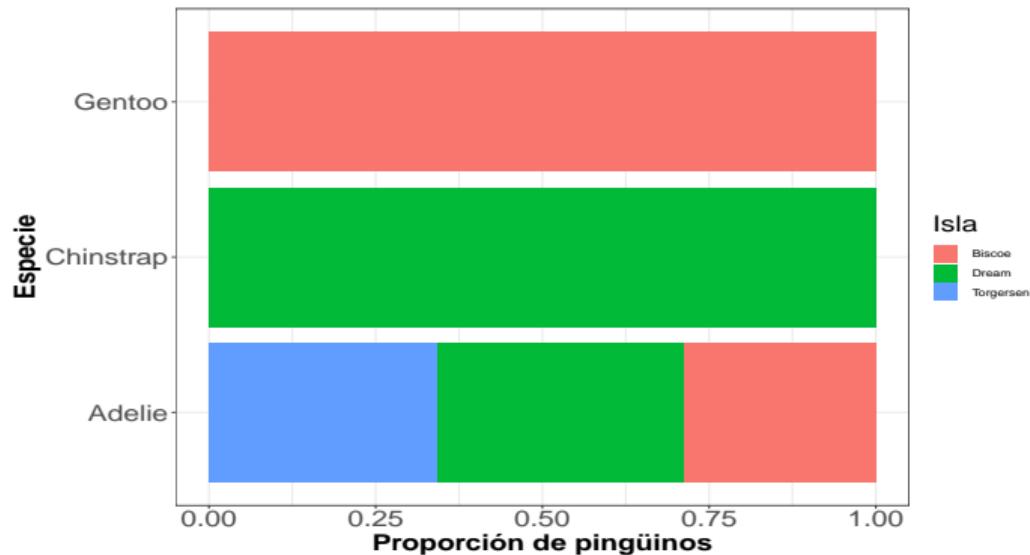
```
penguins %>% ggplot() +  
  geom_bar(aes(species, fill=island),  
           position="dodge") + coord_flip() +  
  guides(fill = guide_legend(title = "Isla")) +  
  labs(x="Número de pingüinos", y="Especie") +  
  theme_bw() +  
  theme(axis.text = element_text(size=20),  
        axis.title = element_text(size=20, face = "bold"),  
        legend.title = element_text(size=20))
```

Cruzando dos variables cualitativas



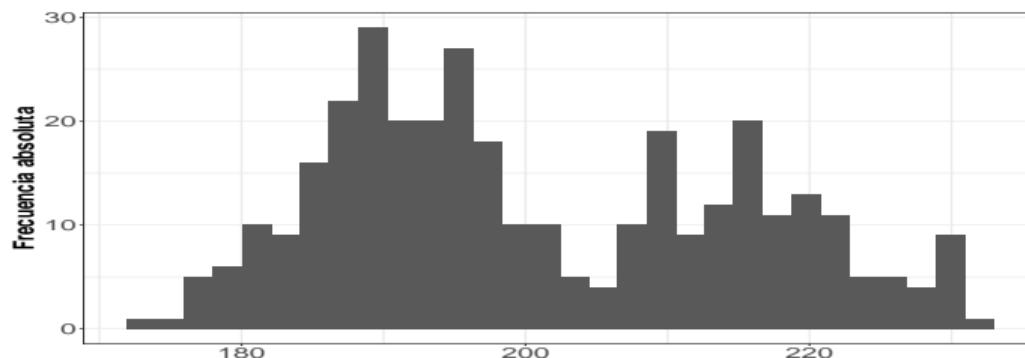
Visualizando una variable cualitativa

Para conseguir que cada barra represente el 100% de la categoría, se indica en el argumento de geom_bar.



Visualizando una variable cuantitativa

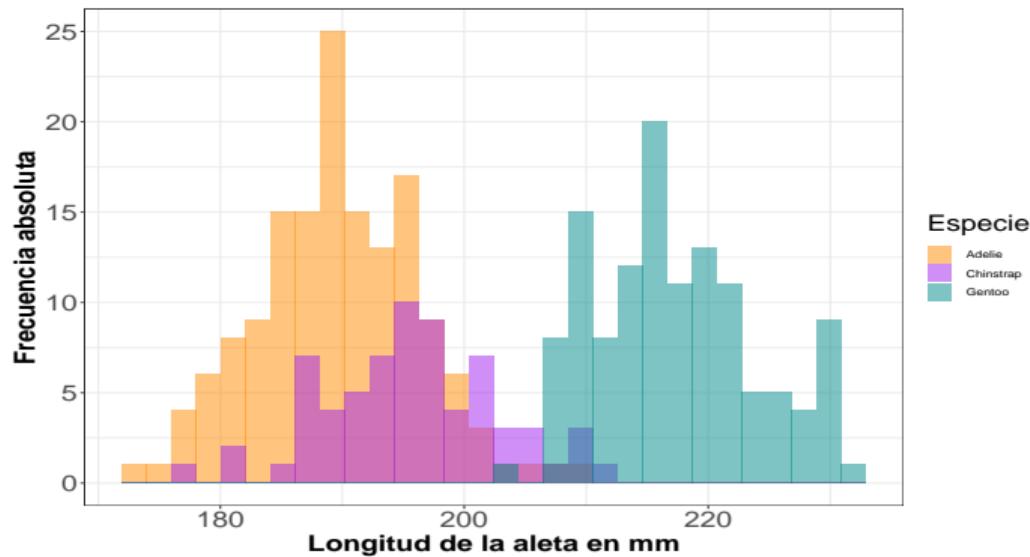
```
penguins %>%  
  ggplot(aes(x = flipper_length_mm)) +  
  geom_histogram(na.rm = TRUE) +  
  labs(x="Longitud de la aleta en mm",  
       y="Frecuencia absoluta") +  
  theme_bw() +  
  theme(axis.text = element_text(size=20),  
        axis.title = element_text(size=20, face = "bold"))
```



Cruzando una variable cuantitativa con una cualitativa

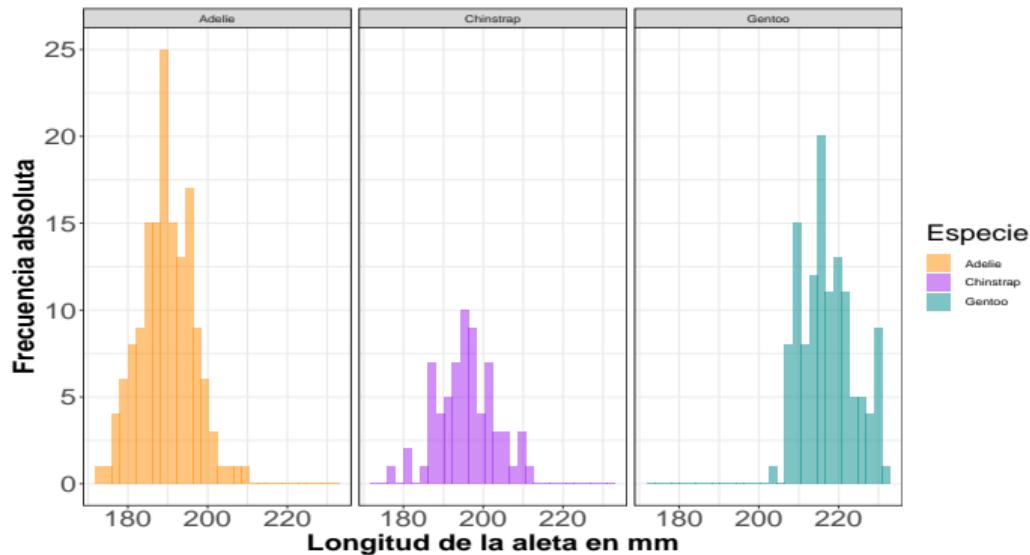
```
ggplot(data = penguins, aes(x = flipper_length_mm)) +  
  geom_histogram(aes(fill = species),  
                 alpha = 0.5,  
                 position = "identity",  
                 na.rm = TRUE) +  
  scale_fill_manual(values = c("darkorange", "purple", "cyan4")) +  
  labs(x = "Longitud de la aleta en mm",  
       y = "Frecuencia absoluta") +  
  guides(fill = guide_legend(title = "Especie")) +  
  theme_bw() +  
  theme(axis.text = element_text(size=20),  
        axis.title = element_text(size=20, face = "bold"),  
        legend.title = element_text(size=20))
```

Cruzando una variable cuantitativa con una cualitativa



Cruzando una variable cuantitativa con una cualitativa

Para que el gráfico sea más claro, se los puede separar con facetas: `facet_grid(.~species)` justo después de la capa `geom_histogram`.

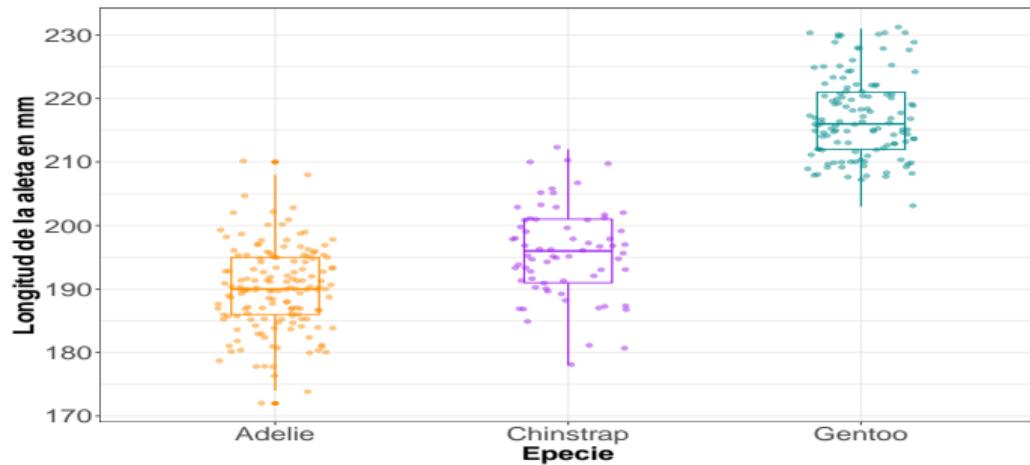


Cruzando una variable cuantitativa con una cualitativa

- La capa facet_grid admite dos variables como argumento, separadas por ~, las categorías de la primera definen las filas y las de la segunda, las columnas. Si solo se usa una, se ubica un punto en el lugar de la otra.
- Otra forma muy conveniente de cruzar una variable cuantitativa con otra cualitativa es usando boxplots.

```
ggplot(data = penguins, aes(x = species, y = flipper_length_mm)) +  
  geom_boxplot(aes(color = species), width = 0.3,  
               show.legend = FALSE) +  
  geom_jitter(aes(color = species), alpha = 0.5,  
              show.legend = FALSE,  
              position = position_jitter(width = 0.2, seed = 0)) +  
  scale_color_manual(values = c("darkorange","purple","cyan4")) +  
  labs(x = "Especie", y = "Longitud de la aleta en mm")
```

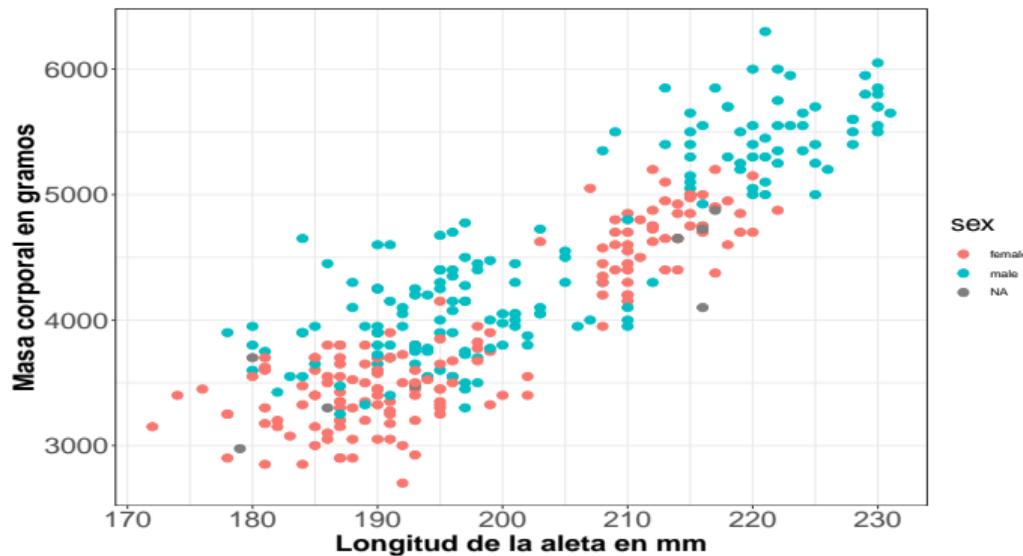
Cruzando una variable cuantitativa con una cualitativa



Cruzando dos variables cuantitativas

```
ggplot(penguins) +  
  geom_point(mapping = aes(x = flipper_length_mm,  
                            y = body_mass_g,  
                            color = sex), size=3) + theme_bw() +  
  theme(axis.text = element_text(size=20),  
        axis.title = element_text(size=20, face = "bold"),  
        legend.title = element_text(size=20)) +  
  guides(fill = guide_legend(title = "Sexo"))
```

Cruzando dos variables cuantitativas



Códigos de color html

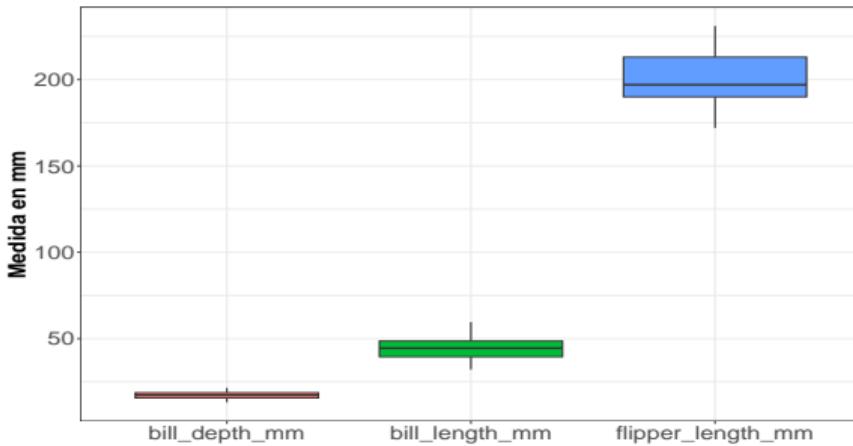
- Para obtener colores realmente personalizados, puede consultar los [códigos de color HTML](#) (también llamados *códigos hexadecimales*, por ejemplo, #ff0000 para el rojo) en lugar de especificar los colores por su [nombre predefinido](#) en R.]
- Hexcodes: códigos que especifican el nivel de intensidad del color rojo (dos primeros), verde (dos segundos) y azul (dos últimos dígitos)
- La familia de funciones `scale_colour_*`() permite ajustar los valores de la estética `color` (por ejemplo, `scale_colour_brewer()` selecciona una paleta del famoso proyecto [ColorBrewer](#)).

Integrando las herramientas de tidyverse . . .

```
penguins_long <- penguins %>%
  tidyr::pivot_longer(
    cols = contains("mm"),
    names_to = "var", values_to = "val") %>%
  tidyr::drop_na()
```

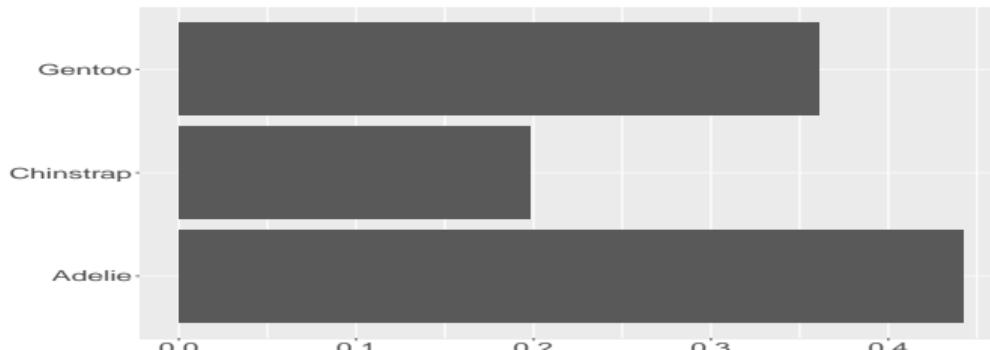
```
penguins_long %>%
  ggplot(aes(x = var, y = val, fill=var)) +
  geom_boxplot() +
  theme_bw() +
  theme(legend.position="none") +
  labs(x="", y="Medida en mm")
```

Integrando las herramientas de tidyverse . . .



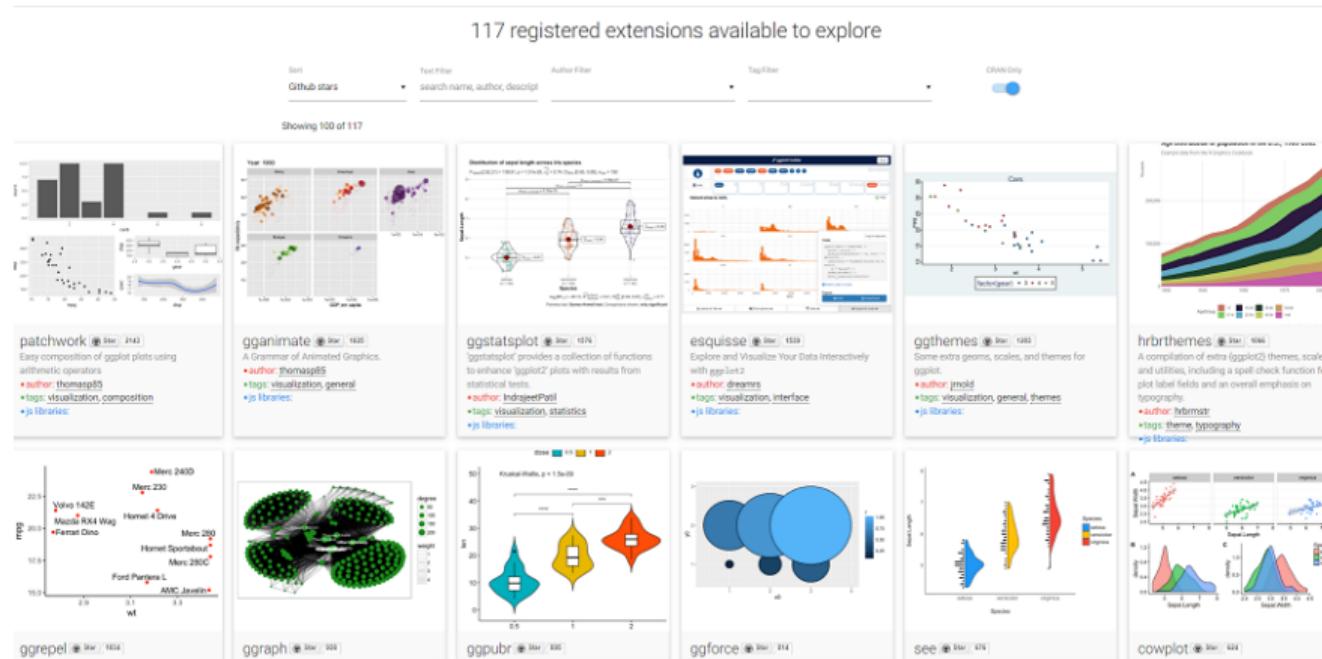
Integrando las herramientas de tidyverse . . .

```
penguins %>%  
  dplyr::count(species) %>%  
  dplyr::mutate(prop = n / sum(n)) %>%  
  ggplot() + geom_col(aes(x = prop, y = species)) +  
  labs(x="Proporción", y="") +  
  theme(axis.text = element_text(size=20),  
axis.title = element_text(size=20, face = "bold"),  
legend.title = element_text(size=20))
```



Extensiones de ggplot

Hay muchas extensiones de ggplot. Ver <https://exts.ggplot2.tidyverse.org/gallery/>

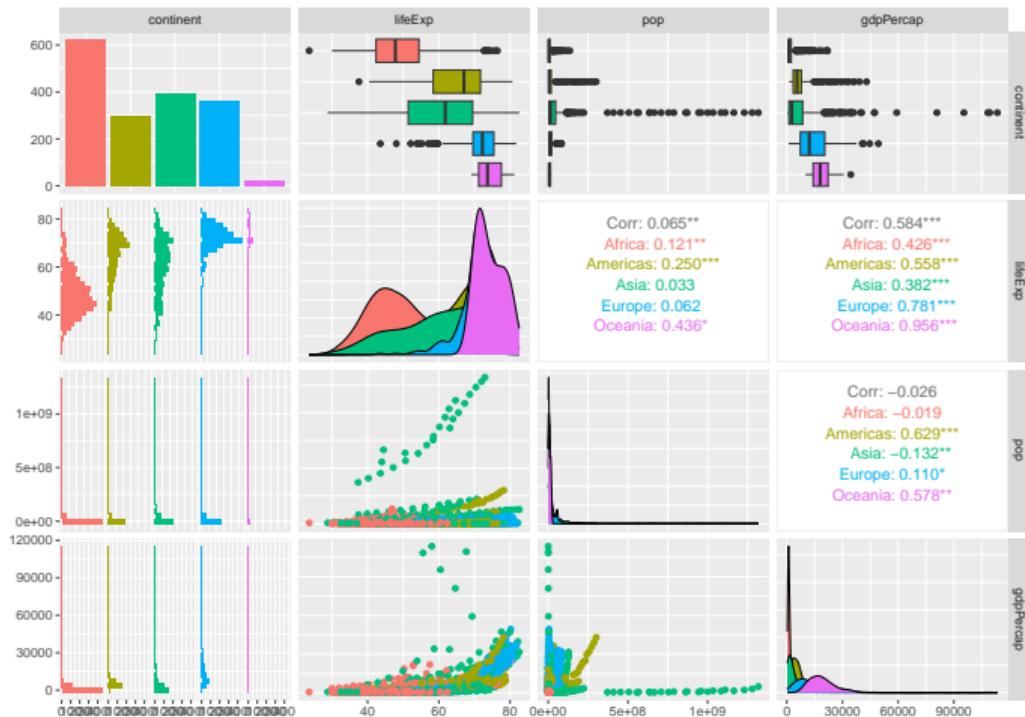


Extensiones de ggplot

[GGally](#) contiene extensiones de ggplot2 para datos multivariantes

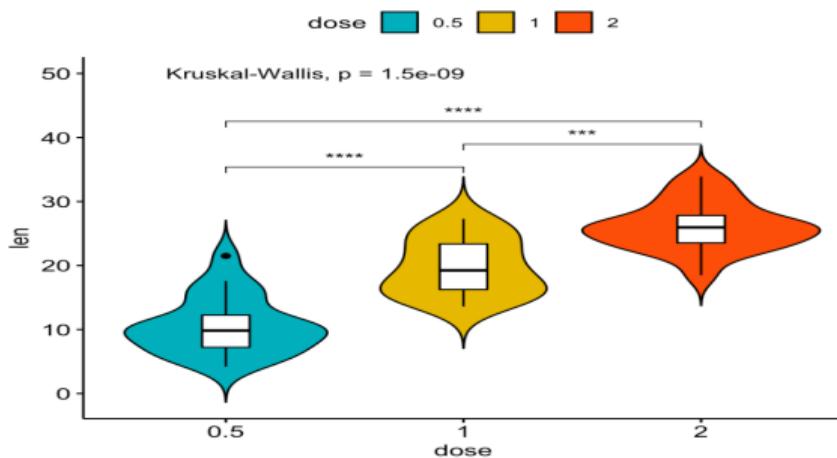
```
library(GGally)
library(gapminder)
gapminder %>% select(-country,-year) %>%
  ggpairs(aes(color=continent))
```

Extensiones de ggplot



Extensiones de ggplot

`ggpubr` proporciona algunas funciones fáciles de usar para crear y personalizar gráficos útiles para una publicación.



Puedes ver más ejemplos en <https://rpkgs.datanovia.com/ggpubr/index.html>

Para cerrar . . .

En <https://allisonhorst.github.io/palmerpenguins/articles/examples.html> puedes encontrar otros ejemplos de gráficos usando los datos de los pingüinos.

En un documento aparte, con extensión html, trataremos cómo hacer gráficos interactivos.