

powered by

R programming

Introducción general al uso de R - Clase 03 -

minolicnp@gmail.com

Ignacio Minoli *pwp*

Observatorio de Biodiversidad del Bosque Atlántico (OBBA)
Instituto de Biología Subtropical (IBS) - CONICET - UNaM.



Pipes: Historia

- Son operadores que permiten expresar múltiples operaciones o funciones en forma secuencial.
- En 2012 `chain()`. Hadley Wickham en 2013 pipe `%.%` . Incluido en `library(dplyr)`.
- En 2013 se publica el paquete `magrittr` ya con el pipe `%>%` . Fusión en 2014 con los paquetes de Hadley Wickham.
- En 2014 pipe::`%>>%`. En 2018 wrapr::`%.>%`.
- En 2016 se trató de usar el R Base para poder tener un pipe nativo.
- En 2016 el paquete `magrittr` comenzó a desarrollar su pipe `%>%` en C para aumentar la velocidad y converger hacia un pipe nativo.
- A fines del 2020 nació el pipe nativo de R Base `|>` , pero recién llegó a ser incluido en la versión de Mayo del 2021.

Diferencias pipes

- Utilizando `%>%`

```
library(magrittr)
1:10 %>% mean()
```

```
[1] 5.5
```

```
1:10 %>% mean
```

```
[1] 5.5
```

- Utilizando `|>` ... es equivalente a esto?

```
1:10 |> mean()
1:10 |> mean
```

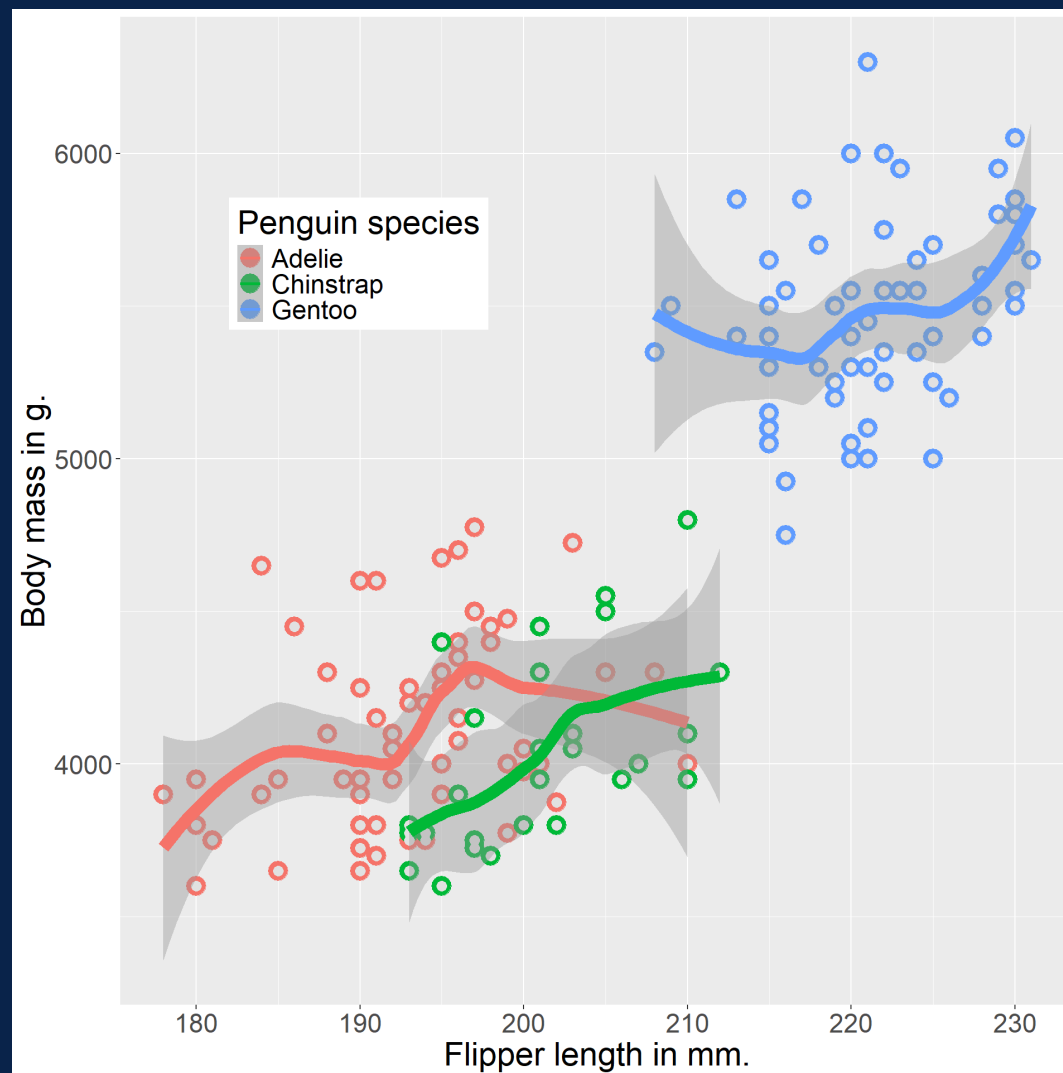
```
Error: The pipe operator requires a function call as RHS (<text>:2:9)
```

- Hay algunas diferencias. Los Pipes pasan elementos de la izquierda (lhs. left-hand side) como el 1er argumento a la derecha (rhs. right-hand side).

Diferencias pipes

```
# Con pipes %>%
library(dplyr)
library(ggplot2)

fig <- palmerpenguins::penguins %>%
  filter(sex == "male") %>%
  filter(body_mass_g > 3550) %>%
  ggplot(aes(flipper_length_mm,
             body_mass_g, size=I(4),
             color = factor(species))) +
  geom_point(aes(colour = factor(species),
                 size = 6)) +
  geom_point(colour = "grey90",
             size = 3) +
  geom_smooth(formula = y ~ x,
             method = "loess") +
  guides(color = guide_legend(
    title = "Penguin species")) +
  theme(legend.position = "inside",
        legend.position.inside = c(.25, .
        text = element_text(
          size = 22)) +
  labs(x = "Flipper length in mm.",
       y = "Body mass in g.")
```

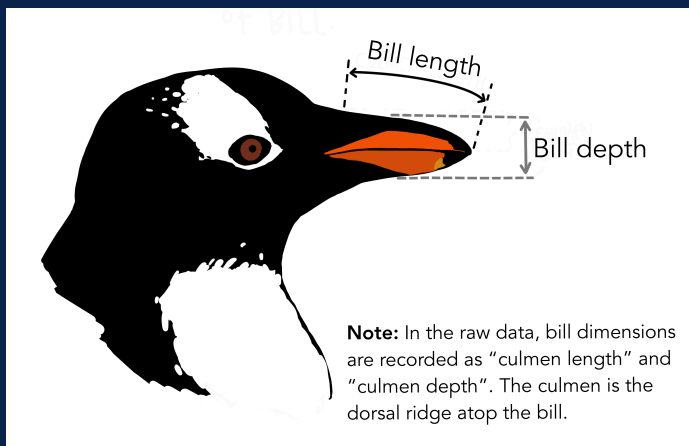


Diferencias pipes

- ¿Qué ocurre? ¿Error?

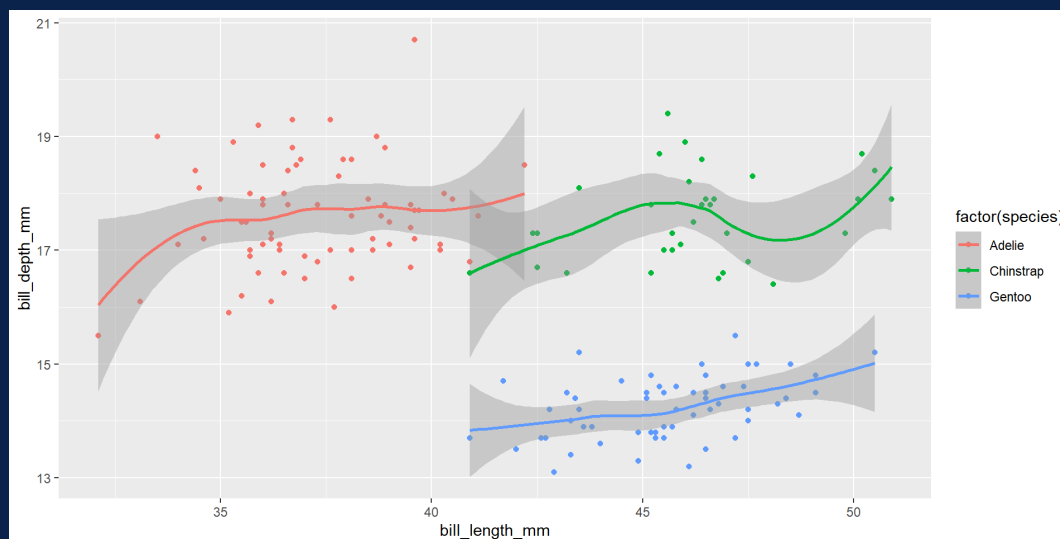
```
palmerpenguins::penguins |>
  dplyr::filter(sex == "female") |>
  dplyr::filter(bill_length_mm < 55) |>
  ggplot2::qplot(x = bill_length_mm,
                 y = bill_depth_mm,
                 geom=c("point"),
                 color = factor(species),
                 data = .) +
  geom_smooth(formula = y ~ x,
              method = "loess")
```

Error in eval(expr, envir, enclos): object '.' not found



- ¿Cual es la diferencia?

```
palmerpenguins::penguins %>%
  dplyr::filter(sex == "female") %>%
  dplyr::filter(bill_length_mm < 55) %>%
  ggplot2::qplot(x = bill_length_mm,
                 y = bill_depth_mm,
                 geom=c("point"),
                 color = factor(species),
                 data = .) +
  geom_smooth(formula = y ~ x,
              method = "loess")
```



Opciones en Pipes

- Utilizar los pipes puede reducir el número de objetos en la sesión y disminuye el uso de recursos en el workspace.
- Uso más frecuente:
 - * Subsets
 - * Exploración de objetos
 - * Estadísticas de resumen
- Manipulación de datos:
 - * Reordenar set de datos
 - * Cambio de formatos
 - * Creación de nuevas variables
- Unión de data frames o tibbles:
 - * Filas
 - * Columnas
 - * Id
 - * Direccionalmente

Exploración de objetos

```
# Funcion de R base
str(iris)
```

```
'data.frame': 150 obs. of 5 variables:
 $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...
```

```
library(dplyr)
```

```
iris %>%
  glimpse()
```

```
Rows: 150
```

```
Columns: 5
```

```
$ Sepal.Length <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9, 5.4, 4....
$ Sepal.Width <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7, 3....
$ Petal.Length <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5, 1.5, 1....
$ Petal.Width <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.2, 0....
$ Species <fct> setosa, setosa, setosa, setosa, setosa, setosa, setosa, s...
```

Ejemplos resumen

- ¿Qué es un resumen en un data frame o tibble?

```
# Con R base
summary(iris)
```

```

Sepal.Length    Sepal.Width    Petal.Length    Petal.Width
Min.      :4.300    Min.      :2.000    Min.      :1.000    Min.      :0.100
1st Qu.:5.100    1st Qu.:2.800    1st Qu.:1.600    1st Qu.:0.300
Median :5.800    Median :3.000    Median :4.350    Median :1.300
Mean    :5.843    Mean    :3.057    Mean    :3.758    Mean    :1.199
3rd Qu.:6.400    3rd Qu.:3.300    3rd Qu.:5.100    3rd Qu.:1.800
Max.    :7.900    Max.    :4.400    Max.    :6.900    Max.    :2.500

Species
setosa      :50
versicolor:50
virginica   :50

```


Ejemplos resumen

```
library(dplyr)

iris %>%
  group_by(Species) %>%
  summarize(
    mean_size = mean(Petal.Length, na.rm = TRUE),
    n()
  )
```

```
# A tibble: 3 × 3
  Species    mean_size `n()`
  <fct>      <dbl> <int>
1 setosa      1.46     50
2 versicolor  4.26     50
3 virginica   5.55     50
```

```
aggregate(data = iris, Petal.Length~Species, FUN = mean)
```

```
  Species Petal.Length
1  setosa      1.462
2 versicolor  4.260
3 virginica   5.552
```

Ejemplos reordenamiento. R base

- Me permite acomodar las filas o columnas de un data frame en un orden deseado.

```
attach(ToothGrowth)

# R base. Orden ascendente a partir de la variable len

new_ToothGrowth <- ToothGrowth[order(len),]
head(ToothGrowth, 3)
```

	len	supp	dose
1	4.2	VC	0.5
2	11.5	VC	0.5
3	7.3	VC	0.5

```
head(new_ToothGrowth, 3)
```

	len	supp	dose
1	4.2	VC	0.5
9	5.2	VC	0.5
4	5.8	VC	0.5

```
detach(ToothGrowth)
```

Ejemplos reordenamiento. R Base

```
attach(CO2)

# R base. Orden descendente a partir de la variable uptake

new_CO2_inv <- CO2[order(-uptake),]
head(CO2, 3)
```

Grouped Data: uptake ~ conc | Plant

	Plant	Type	Treatment	conc	uptake
1	Qn1	Quebec	nonchilled	95	16.0
2	Qn1	Quebec	nonchilled	175	30.4
3	Qn1	Quebec	nonchilled	250	34.8

```
head(new_CO2_inv, 3)
```

Grouped Data: uptake ~ conc | Plant

	Plant	Type	Treatment	conc	uptake
21	Qn3	Quebec	nonchilled	1000	45.5
14	Qn2	Quebec	nonchilled	1000	44.3
20	Qn3	Quebec	nonchilled	675	43.9

```
detach(CO2)
```

Ejemplos reordenamiento. dplyr

```
# R dplyr. Orden ascendente a partir de la variable weight
attach(ChickWeight)
library(dplyr)

new_ChickWeight <- ChickWeight %>%
  arrange(weight)
head(ChickWeight, 3) ; head(new_ChickWeight, 3)
```

Grouped Data: weight ~ Time | Chick

	weight	Time	Chick	Diet
1	42	0	1	1
2	51	2	1	1
3	59	4	1	1

Grouped Data: weight ~ Time | Chick

	weight	Time	Chick	Diet
1	35	2	18	1
2	39	2	3	1
3	39	0	18	1

```
new_ChickWeight_inv <- ChickWeight %>%
  arrange(desc(weight))
head(ChickWeight, 3) ; head(new_ChickWeight_inv, 3)
```

Grouped Data: weight ~ Time | Chick

	weight	Time	Chick	Diet
1	42	0	1	1
2	51	2	1	1
3	59	4	1	1

Grouped Data: weight ~ Time | Chick

	weight	Time	Chick	Diet
1	373	21	35	3
2	361	20	35	3
3	341	21	34	3

Ejemplos subsets

- Me permite segmentar los datos de acuerdo a lo necesitado.
- Puede realizarse con una selección con operadores por números, por caracteres, por filas, por columnas o con múltiples combinaciones anteriores.
- R base. Puedo 1- seleccionar y/o 2- excluir: variables o filas.
- Estas selecciones con R base pueden hacerse por posición o por nombre.

```
# Selección variables con R base.
glimpse(stackloss)
```

```
Rows: 21
Columns: 4
$ Air.Flow    <dbl> 80, 80, 75, 62, 62, 62, 62, 62, 58, 58, 58, 58, 58, 58, 50,...
$ Water.Temp  <dbl> 27, 27, 25, 24, 22, 23, 24, 24, 23, 18, 18, 17, 18, 19, 18,...
$ Acid.Conc.  <dbl> 89, 88, 90, 87, 87, 87, 93, 93, 87, 80, 89, 88, 82, 93, 89,...
$ stack.loss  <dbl> 42, 37, 37, 28, 18, 18, 19, 20, 15, 14, 14, 13, 11, 12, 8, ...
```

```
selec_data <- stackloss[c("Water.Temp", "Acid.Conc.")]
head(selec_data, 4)
```

	Water.Temp	Acid.Conc.
1	27	89
2	27	88
3	25	90
4	24	87

Ejemplos subsets

- En la familia tidyverse hay básicamente 2 operaciones: **filter** para subsets de filas y **select** para subsets de columnas o variables.
- Con la función filter hay dos opciones principales group_by() y ungroup().
- Las funciones y operadores más comunes son: ==, >, >=, &, |, !, xor(), is.na(), between(), near().

```
# Subset de filas con R familia tidyverse SIN group_by().
starwars %>%
  filter(mass > mean(mass, na.rm = TRUE)) %>%
  print(n = 3)
```

```
# A tibble: 10 × 14
  name      height  mass hair_color skin_color eye_color birth_year sex  gender
<chr>      <int> <dbl> <chr>      <chr>      <chr>      <dbl> <chr> <chr>
1 Darth Va...   202   136 none       white       yellow      41.9  male  mascu...
2 Owen Lars    178   120 brown, gr... light       blue        52    male  mascu...
3 Chewbacca    228   112 brown      unknown    blue        200   male  mascu...
# i 7 more rows
# i 5 more variables: homeworld <chr>, species <chr>, films <list>,
#   vehicles <list>, starships <list>
```

Ejemplos subsets

```
# Subset de filas con R familia tidyverse CON group_by().
starwars %>%
  group_by(gender) %>%
  filter(mass > mean(mass, na.rm = TRUE)) %>%
  print(n = 10)
```

```
# A tibble: 15 × 14
# Groups:   gender [3]
  name      height  mass hair_color skin_color eye_color birth_year sex  gender
  <chr>      <int> <dbl> <chr>      <chr>      <chr>      <dbl> <chr> <chr>
1 Darth V...    202   136 none       white      yellow      41.9 male  mascu...
2 Owen La...    178   120 brown, gr... light      blue        52  male  mascu...
3 Beru Wh...    165    75 brown      light      blue        47  fema... femin...
4 Chewbac...    228   112 brown      unknown    blue       200  male  mascu...
5 Jabba D...    175  1358 <NA>        green-tan... orange     600  herm... mascu...
6 Jek Ton...    180   110 brown      fair       blue        NA  <NA>  <NA>
7 IG-88        200   140 none       metal      red         15  none  mascu...
8 Bossk        190   113 none       green      red         53  male  mascu...
9 Ayla Se...    178    55 none       blue       hazel       48  fema... femin...
10 Gregar ...    185    85 black      dark       brown       NA  <NA>  <NA>
# i 5 more rows
# i 5 more variables: homeworld <chr>, species <chr>, films <list>,
#   vehicles <list>, starships <list>
```

Ejemplos cambio de clases

- Puede hacerse con R base: `class()`, `as.numeric()`, `as.factor()`, `as.character()`.

```
df <- data.frame(var1 = LETTERS[5:10],  
                 var2 = as.character(5:10),  
                 var3 = as.character(seq(0, 1, 0.2))  
                 )  
  
df
```

	var1	var2	var3
1	E	5	0
2	F	6	0.2
3	G	7	0.4
4	H	8	0.6
5	I	9	0.8
6	J	10	1

```
sapply(df, class)
```

	var1	var2	var3
	"character"	"character"	"character"

Ejemplos cambio de clases

- Con R base

```
df <- data.frame(clase = LETTERS[5:8],
                 nota = as.character(5:8),
                 indice = as.character(seq(0, 1, 0.33))
                 )

df
```

```
  clase nota indice
1     E    5      0
2     F    6  0.33
3     G    7  0.66
4     H    8  0.99
```

```
str(df)
```

```
'data.frame':  4 obs. of  3 variables:
 $ clase : chr  "E" "F" "G" "H"
 $ nota  : chr  "5" "6" "7" "8"
 $ indice: chr  "0" "0.33" "0.66" "0.99"
```

```
df$indice <- as.numeric(df$indice)
str(df)
```

```
'data.frame':  4 obs. of  3 variables:
 $ clase : chr  "E" "F" "G" "H"
 $ nota  : chr  "5" "6" "7" "8"
 $ indice: num  0 0.33 0.66 0.99
```

Dialecto tidyverse

- Seleccionar variables con operadores:
 - : para seleccionar un rango consecutivo de variables. e.g. **1:4**
 - c()** para combinar selecciones. e.g. **c("a", "b")**
 - para descartar variables (negación). e.g. **-c("var1", "var2")**
 - !** para seleccionar un set de variables (no es verdadero). e.g. **!var1**
 - &** y **|** para seleccionar intersección o unión de un set de 2 variables. Son operadores condicionales **"y"** **"o"**
- Se complementa con asistentes:
 - everything()**: para todas las variables (específico de columnas).
 - last_col()**: selecciona la última variables (específico de columnas).
 - starts_with()**: comienza con un prefijo (match de patrones en los nombres).
 - ends_with()**: termina con un sufijo (match de patrones en los nombres).
 - contains()**: contiene un string literal (match de patrones en los nombres).
 - matches()**: una expresión regular (match de patrones en los nombres).
 - num_range()**: un rango numérico (match de patrones en los nombres).

Dialecto tidyverse

```
library(dplyr)
names(starwars)
```

```
[1] "name"      "height"    "mass"      "hair_color" "skin_color"
[6] "eye_color" "birth_year" "sex"       "gender"     "homeworld"
[11] "species"   "films"     "vehicles"  "starships"
```

```
starwars %>%
  select(name:mass) %>%
  head(3)
```

```
# A tibble: 3 × 3
  name      height  mass
<chr>      <int> <dbl>
1 Luke Skywalker    172    77
2 C-3PO             167    75
3 R2-D2              96    32
```

```
names(iris)
```

```
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

```
iris %>%
  select(starts_with("Petal") | ends_with("Width")) %>%
  head(3)
```

```
Petal.Length Petal.Width Sepal.Width
1           1.4          0.2          3.5
2           1.4          0.2          3.0
3           1.3          0.2          3.2
```

Seccionar y concatenar

```
library(dplyr)
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 %>% slice(1)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21	6	160	110	3.9	2.62	16.46	0	1	4	4

```
slice(mtcars, -(1:2)) %>%
  head(., 4)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
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

Seccionar y concatenar

```
tail(mtcars, 2)
```

```
      mpg cyl disp  hp drat   wt  qsec vs am gear carb
Maserati Bora 15.0   8  301 335 3.54 3.57 14.6  0  1    5    8
Volvo 142E    21.4   4  121 109 4.11 2.78 18.6  1  1    4    2
```

```
nrow(mtcars)
```

```
[1] 32
```

```
mtcars %>%
  slice(31:n())
```

```
      mpg cyl disp  hp drat   wt  qsec vs am gear carb
Maserati Bora 15.0   8  301 335 3.54 3.57 14.6  0  1    5    8
Volvo 142E    21.4   4  121 109 4.11 2.78 18.6  1  1    4    2
```

```
palmerpenguins::penguins %>%
  group_by(species) %>%
  arrange(desc(body_mass_g)) %>%
  slice(1:2)
```

```
# A tibble: 6 × 8
# Groups:   species [3]
  species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
  <fct>     <fct>         <dbl>         <dbl>         <int>         <int>
1 Adelie   Biscoe          43.2           19           197          4775
2 Adelie   Biscoe          41            20           203          4725
3 Chinstrap Dream         52            20.7         210          4800
4 Chinstrap Dream        52.8            20           205          4550
5 Gentoo   Biscoe          49.2           15.2         221          6300
6 Gentoo   Biscoe        59.6            17           230          6050
# i 2 more variables: sex <fct>, year <int>
```

Creación de nuevas variables

- Con R base se pueden crear a partir de la aritmética de vectores.

```
hospital <- c("Garrahan", "Perrando")
pacientes <- c(150, 350)
cost_indv <- c(3.1, 2.5)
df <- data.frame(hospital, pacientes, cost_indv)
df
```

```
  hospital pacientes cost_indv
1 Garrahan      150      3.1
2 Perrando      350      2.5
```

```
# Nueva variable
df$tot_cost <- df$pacientes * df$cost_indv
df$tot_cost
```

```
[1] 465 875
```

```
df
```

```
  hospital pacientes cost_indv tot_cost
1 Garrahan      150      3.1      465
2 Perrando      350      2.5      875
```

Creación de nuevas variables

- Con la familia tidyverse se pueden crear nuevas variables utilizando **mutate()**.

```
tratamiento <- c("Organico", "Agroquimico")
h_planta <- c(150, 350)
cost_indv <- c(3.1, 2.5)
df <- tibble(tratamiento, h_planta, cost_indv)
df
```

```
# A tibble: 2 × 3
  tratamiento h_planta cost_indv
  <chr>      <dbl>    <dbl>
1 Organico    150      3.1
2 Agroquimico 350      2.5
```

```
# Nuevas variables
df_new <- df %>% mutate(
  tot_cost = h_planta * cost_indv,
  tot_cost_3y = tot_cost * 3,
  subsidio = case_when(
    tot_cost_3y > 1500 ~ "Si Corresponde",
    tot_cost_3y < 1500 ~ "No Corresponde"
  )
)
df_new
```

```
# A tibble: 2 × 6
  tratamiento h_planta cost_indv tot_cost tot_cost_3y subsidio
  <chr>      <dbl>    <dbl>    <dbl>    <dbl> <chr>
1 Organico    150      3.1      465      1395 No Corresponde
2 Agroquimico 350      2.5      875      2625 Si Corresponde
```

Unir data frames o tibbles

- Con R base hay varias funciones `rbind()`, `cbind()`, `merge()`, `join()`.

```
Tasa_Disper_A <- c(6.6, 6.7, 8.2)
Camb_Urb <- c(0.2, 0.1, 0.5)
df_habitat <- cbind(Tasa_Disper_A, Camb_Urb)
df_habitat
```

```
      Tasa_Disper_A Camb_Urb
[1,]           6.6      0.2
[2,]           6.7      0.1
[3,]           8.2      0.5
```

```
Tasa_Disper_B <- c(10, 15, 22)
Agr_Fam <- c(100, 150, 300)
df_agr <- cbind(Tasa_Disper_B, Agr_Fam)
df_agr
```

```
      Tasa_Disper_B Agr_Fam
[1,]             10     100
[2,]             15     150
[3,]             22     300
```

```
# Uno las filas del data frame
df_compuesto <- rbind(df_habitat, df_agr)
# No es requisito tener el mismo nro de columnas
df_compuesto %>% head(4)
```

```
      Tasa_Disper_A Camb_Urb
[1,]           6.6      0.2
[2,]           6.7      0.1
[3,]           8.2      0.5
[4,]          10.0     100.0
```


Unir data frames o tibbles

- Con tidyverse **bind_rows()**, **bind_cols()**, **join()**.

```
a <- data.frame(a = 1:2, b = 3:4, c = 5:6)
b <- data.frame(a = 7:8, b = 2:3, c = 3:4, d = 8:9)
# R Base
rbind(a, b)
```

Error in rbind(deparse.level, ...): numbers of columns of arguments do not match

- ¿Qué pasó? ¿A qué se debe el error? ¿Ideas?
- Otra función que fue muy utilizada es **merge()**.
- Reemplazada por **library(dplyr)** con:
 - left_join()**
 - right_join()**
 - inner_join()**
 - full_join()**
 - semi_join()**
 - anti_join()**

Unir data frames o tibbles

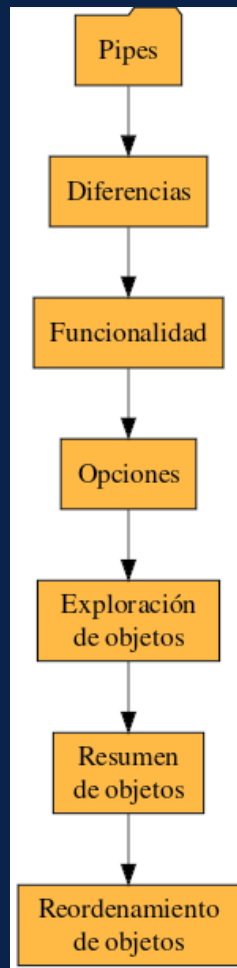
```
# Tidyverse
library(dplyr)
# Une los dos elementos
bind_rows(a, b)
```

	a	b	c	d
1	1	3	5	NA
2	2	4	6	NA
3	7	2	3	8
4	8	3	4	9

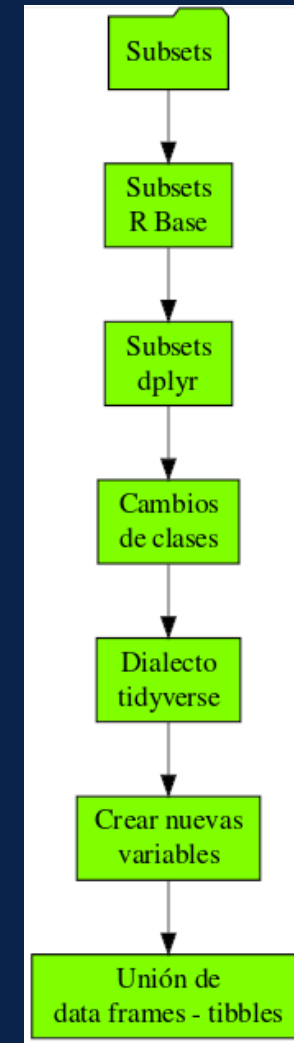
```
# Con este argumento puedo ver de que elemento viene cada fila que fue unida
bind_rows(list(a, b), .id = "id")
```

	id	a	b	c	d
1	1	1	3	5	NA
2	1	2	4	6	NA
3	2	7	2	3	8
4	2	8	3	4	9

Estructura, Pipes, Indexación



Subsets, Nuevos df, Nuevas vars



QR - presentación

PDF Presentación: Clase 03 - Teoría



Fin. Clase 03 - Teoría

!!! Muchas gracias !!!

¿ Preguntas ? ... ¿ Consultas ?

