Hola

Enfoque Estadístico del Aprendizaje

Programación funcional y ajuste por optimización

Bernardo L. Español · 12 de Septiembre

Programación funcional

PROGRAMACIÓN FUNCIONAL

Qué es `purrr`

PROGRAMACIÓN FUNCIONAL

Qué es `purrr`

`purrr` es la librería de `tidyverse` con funciones y métodos de **programación funcional** para trabajar con listas y vectores.

map()

La función *principal* de la programación funcional es `map()`. **Aplica una función a cada elemento** de una estructura de datos.

```
map(.x, .f, ...)
```

map()

La función *principal* de la programación funcional es `map()`. **Aplica una función a cada elemento** de una estructura de datos.

```
map(.x, .f, ...)
```

- `.x`: vector o lista de entrada
- `.f`: función que se quiere aplicar
- `...`: otros argumentos que pueda necesitar la función

PROGRAMACIÓN FUNCIONAL

map()

La función *principal* de la programación funcional es `map()`. **Aplica una función a cada elemento** de una estructura de datos.

```
map(.x, .f, ...)
```

- `.x`: vector o lista de entrada
- `.f`: función que se quiere aplicar
- `...`: otros argumentos que pueda necesitar la función

No es más que un loop que recorre cada elemento, le aplica la función y guarda el resultado.

La familia de `map`

Hay muchas funciones de *mapeo*, todas con **tipado estable**: siempre devuelven un tipo concreto.

La familia de `map`

Hay muchas funciones de *mapeo*, todas con **tipado estable**: siempre devuelven un tipo concreto.

- `map()` siempre devuelve una lista.

La familia de `map`

Hay muchas funciones de *mapeo*, todas con **tipado estable**: siempre devuelven un tipo concreto.

- `map()` siempre devuelve una lista.
- `map_lgl()`, `map_int()`, `map_dbl()` y `map_chr()` devuelven un vector del tipo que indica el sufijo.

PROGRAMACIÓN FUNCIONAL

La familia de **map**

```
l <- c(1, 2, 3)
pow_f <- function(x, p) x^p
```

La familia de `map`

```
l <- c(1, 2, 3)
pow_f <- function(x, p) x^p
```

Este loop

```
out <- numeric(length(l))
for (i in seq_along(out)) {
  out[i] <- pow_f(l[i], p = 2)
}
out
```

```
[1] 1 4 9
```

La familia de `map`

```
l <- c(1, 2, 3)
pow_f <- function(x, p) x^p
```

Este loop

```
out <- numeric(length(l))
for (i in seq_along(out)) {
  out[i] <- pow_f(l[i], p = 2)
}
out
```

```
[1] 1 4 9
```

Se puede escribir de cualquiera de estas formas

La familia de `map`

```
l <- c(1, 2, 3)
pow_f <- function(x, p) x^p
```

Este loop

```
out <- numeric(length(l))
for (i in seq_along(out)) {
  out[i] <- pow_f(l[i], p = 2)
}
out
```

```
[1] 1 4 9
```

Se puede escribir de cualquiera de estas formas

```
map_dbl(l, pow_f, p = 2)
```

La familia de `map`

```
l <- c(1, 2, 3)
pow_f <- function(x, p) x^p
```

Este loop

```
out <- numeric(length(l))
for (i in seq_along(out)) {
  out[i] <- pow_f(l[i], p = 2)
}
out
```

```
[1] 1 4 9
```

Se puede escribir de cualquiera de estas formas

```
map_dbl(l, pow_f, p = 2)
l %>% map_dbl(pow_f, p = 2)
```

La familia de `map`

```
l <- c(1, 2, 3)
pow_f <- function(x, p) x^p
```

Este loop

```
out <- numeric(length(l))
for (i in seq_along(out)) {
  out[i] <- pow_f(l[i], p = 2)
}
out
```

```
[1] 1 4 9
```

Se puede escribir de cualquiera de estas formas

```
map_dbl(l, pow_f, p = 2)
l %>% map_dbl(pow_f, p = 2)
l |> map_dbl(pow_f, p = 2)
```

La familia de `map`

Y si en vez de un vector queremos la lista

```
map(l, pow_f, p = 2)
```

```
[[1]]
[1] 1

[[2]]
[1] 4

[[3]]
[1] 9
```

La familia de `map`

`map2_*` aplica una función usando elementos de **dos** listas. Por ejemplo, elevar cada número de una lista a la potencia que dice la otra:

```
l <- c(1, 2, 3, 4)
p <- c(2, 3, 2, 5)

map2_dbl(l, p, pow_f)
```

```
[1]    1    8    9 1024
```

Funciones anónimas

Si la función tiene más argumentos que los que `map2` recorre, la envolvemos en una anónima:

```
shifted_pow_f <- function(x, x0, p) (x - x0)^p
```

Funciones anónimas

Si la función tiene más argumentos que los que `map2` recorre, la envolvemos en una anónima:

```
shifted_pow_f <- function(x, x0, p) (x - x0)^p

map2_dbl(l, p, \(x, y) shifted_pow_f(x, x0 = 1, p = y))
```

```
[1]   0   1   4 243
```

Funciones anónimas

Si la función tiene más argumentos que los que `map2` recorre, la envolvemos en una anónima:

```
shifted_pow_f <- function(x, x0, p) (x - x0)^p

map2_dbl(l, p, \(x, y) shifted_pow_f(x, x0 = 1, p = y))
```

```
[1]   0   1   4 243
```

También se puede hacer así, pero no lo hagan:

```
map2_dbl(l, p, shifted_pow_f, x0 = 1)
```

Otro comando: `split()`

`split()` corta un dataframe en una lista, con un dataframe por grupo:

```
d <- tibble(g = c("a", "a", "b"), x = c(1, 2, 10))
```

Otro comando: `split()`

`split()` corta un dataframe en una lista, con un dataframe por grupo:

```
d <- tibble(g = c("a", "a", "b"), x = c(1, 2, 10))

split(d, ~g)
```

```
$a
# A tibble: 2 × 2
  g         x
  <chr> <dbl>
1 a         1
2 a         2

$b
# A tibble: 1 × 2
  g         x
  <chr> <dbl>
1 b        10
```

split() + map()

Cada elemento de la lista es un dataframe, y `map_*` lo recorre como a cualquier otro:

```
split(d, ~g) %>% map_dbl(\(part) mean(part$x))
```

```
   a    b
 1.5 10.0
```

split() + map_dfr()

Si queremos tener el resultado en una tabla podemos hacer algo así:

```
split(d, ~g) %>% map_dfr(\(part) tibble(media = mean(part$x)), .id = "g")
```

```
# A tibble: 2 × 2
  g     media
  <chr> <dbl>
1 a       1.5
2 b      10
```

Ejercicio de la clase pasada

EJERCICIO DE LA CLASE PASADA

Los datos

```
library(tidyverse)

df <- read_csv("global-energy-substitution.csv")
```

EJERCICIO DE LA CLASE PASADA

Los datos

```
library(tidyverse)

df <- read_csv("global-energy-substitution.csv")
```

```
df <- df %>%
  rename(
    "Other renewables" = "Other renewables (TWh, substituted energy)",
    "Biofuels" = "Biofuels (TWh, substituted energy)",
    "Solar" = "Solar (TWh, substituted energy)",
    "Wind" = "Wind (TWh, substituted energy)",
  ) %>%
  select("Year", "Biofuels", "Solar", "Wind", "Other renewables")
```

EJERCICIO DE LA CLASE PASADA

Los datos

Year	Biofuels	Solar	Wind	Other renewables
2019	1127	1749	3520	2038
2020	1068	2117	3940	2152
2021	1128	2593	4583	2318
2022	1193	3250	5185	2379
2023	1319	4027	5665	2411
2024	1367	5151	6124	2476

EJERCICIO DE LA CLASE PASADA

De ancho a largo

`ggplot` necesita **una fila por año y fuente**, no una columna por fuente.

```
df_long <- df %>%
  pivot_longer(
    cols = -Year,
    names_to = "Source",
    values_to = "TWh"
  )
```

EJERCICIO DE LA CLASE PASADA

De ancho a largo

`ggplot` necesita **una fila por año y fuente**, no una columna por fuente.

```
df_long <- df %>%
  pivot_longer(
    cols = -Year,
    names_to = "Source",
    values_to = "TWh"
  )
```

Year	Source	TWh
2024	Biofuels	1367
2024	Solar	5151
2024	Wind	6124
2024	Other renewables	2476

EJERCICIO DE LA CLASE PASADA

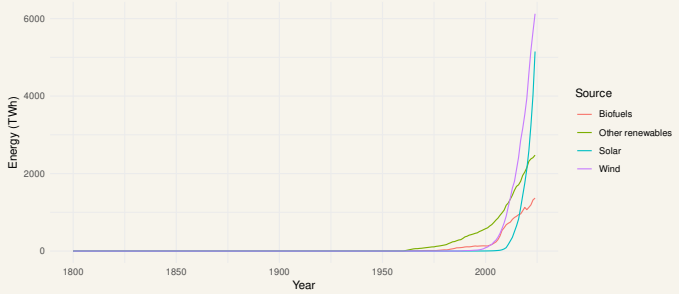
Las cuatro fuentes

```
ggplot(df_long, aes(x = Year, y = TWh, color = Source)) +
  geom_line() +
  labs(y = "Energy (TWh)", x = "Year") +
  theme_minimal()
```

EJERCICIO DE LA CLASE PASADA

Las cuatro fuentes

```
ggplot(df_long, aes(x = Year, y = TWh, color = Source)) +
  geom_line() +
  labs(y = "Energy (TWh)", x = "Year") +
  theme_minimal()
```



EJERCICIO DE LA CLASE PASADA

Una fuente

```
one_source_df <- df_long %>%
  filter(Source == "Other renewables", TWh > 0)
```

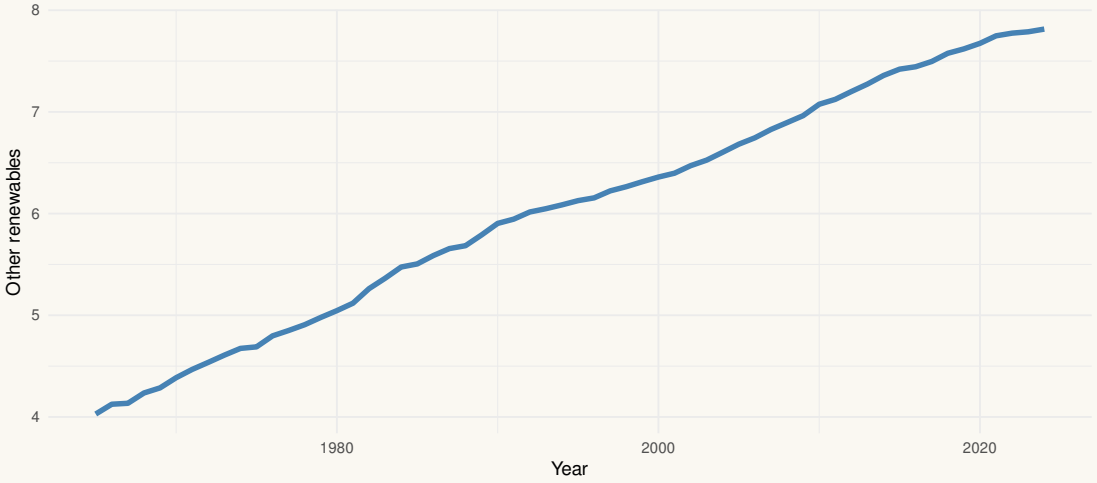
EJERCICIO DE LA CLASE PASADA

Una fuente

```
one_source_df <- df_long %>%
  filter(Source == "Other renewables", TWh > 0)

ggplot(data = one_source_df, aes(x = Year, y = log(TWh))) +
  geom_line(color = "steelblue", linewidth = 1.5) +
  labs(x = "Year", y = "Other renewables") +
  theme_minimal()
```

EJERCICIO DE LA CLASE PASADA

Una fuente



EJERCICIO DE LA CLASE PASADA

Ajuste exponencial

```
fit_exp <- function(d) {
  coefs <- coef(lm(log(TWh) ~ Year, data = d))
```

EJERCICIO DE LA CLASE PASADA

Ajuste exponencial

```
fit_exp <- function(d) {
  coefs <- coef(lm(log(TWh) ~ Year, data = d))

  tibble(a = exp(coefs[[1]]), b = coefs[[2]])
}
```

EJERCICIO DE LA CLASE PASADA

La salida de `fit_exp`

```
fit <- fit_exp(one_source_df)
a <- fit$a
b <- fit$b

fit
```

```
# A tibble: 1 × 2
         a      b
     <dbl>  <dbl>
1 1.07e-54 0.0653
```

EJERCICIO DE LA CLASE PASADA

Plot en semi-logy

```
ggplot(data = one_source_df, aes(x = Year, y = log(TWh))) +
  geom_line(color = "steelblue", linewidth = 1.5) +
```

EJERCICIO DE LA CLASE PASADA

Plot en semi-logy

```
ggplot(data = one_source_df, aes(x = Year, y = log(TWh))) +
  geom_line(color = "steelblue", linewidth = 1.5) +
  stat_function(fun = \(x) log(a) + b * x,
                color = "black", linetype = "dashed",
                linewidth = 1.2) +
```

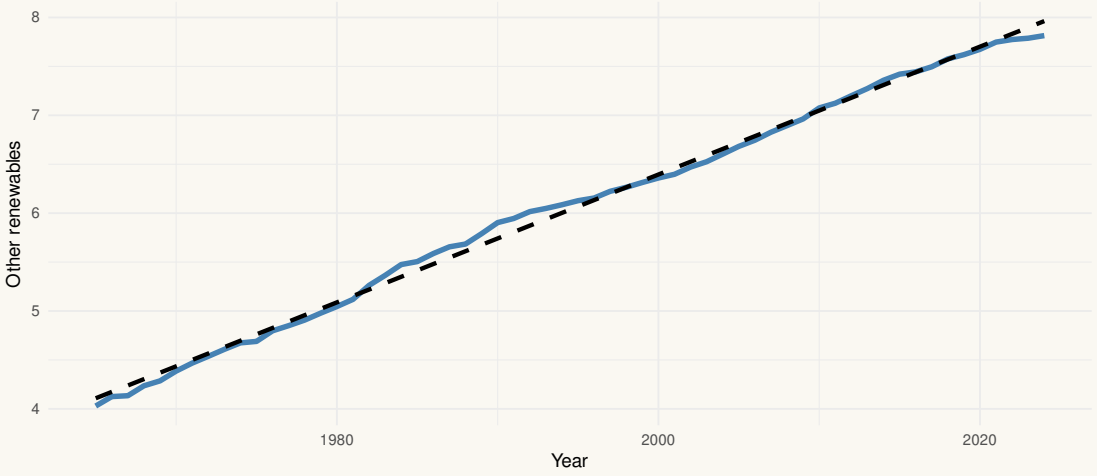
EJERCICIO DE LA CLASE PASADA

Plot en semi-logy

```
ggplot(data = one_source_df, aes(x = Year, y = log(TWh))) +
  geom_line(color = "steelblue", linewidth = 1.5) +
  stat_function(fun = \(x) log(a) + b * x,
                color = "black", linetype = "dashed",
                linewidth = 1.2) +
  labs(x = "Year", y = "Other renewables") +
  theme_minimal()
```

EJERCICIO DE LA CLASE PASADA

Plot en semi-logy



EJERCICIO DE LA CLASE PASADA

Plot en cords. originales

```
ggplot(data = one_source_df, aes(x = Year, y = TWh)) +
  geom_line(color = "steelblue", linewidth = 1.5) +
```

EJERCICIO DE LA CLASE PASADA

Plot en cords. originales

```
ggplot(data = one_source_df, aes(x = Year, y = TWh)) +
  geom_line(color = "steelblue", linewidth = 1.5) +
  stat_function(fun = \(x) a * exp(b * x),
                color = "black", linetype = "dashed",
                linewidth = 1.2) +
```

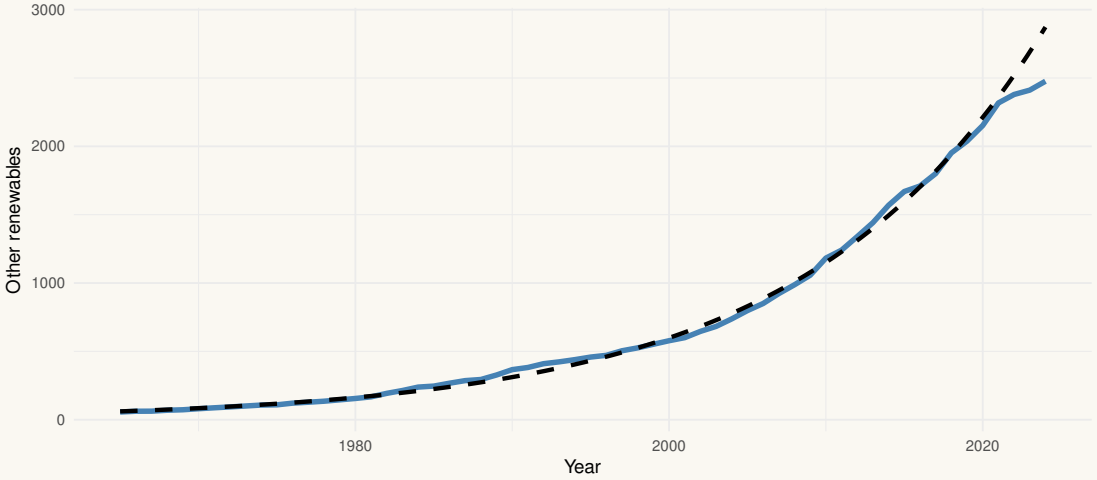
EJERCICIO DE LA CLASE PASADA

Plot en cords. originales

```
ggplot(data = one_source_df, aes(x = Year, y = TWh)) +
  geom_line(color = "steelblue", linewidth = 1.5) +
  stat_function(fun = \(x) a * exp(b * x),
                color = "black", linetype = "dashed",
                linewidth = 1.2) +
  labs(x = "Year", y = "Other renewables") +
  theme_minimal()
```

EJERCICIO DE LA CLASE PASADA

Plot en cords. originales



EJERCICIO DE LA CLASE PASADA

Todas las fuentes

EJERCICIO DE LA CLASE PASADA

Todas las fuentes

```
coef_table <- df_long %>%
```

EJERCICIO DE LA CLASE PASADA

Todas las fuentes

```
coef_table <- df_long %>%
  filter(TWh > 0) %>%
```

EJERCICIO DE LA CLASE PASADA

Todas las fuentes

```
coef_table <- df_long %>%
  filter(TWh > 0) %>%
  split(~Source) %>%
  map_dfr(fit_exp, .id = "Source")

coef_table
```

EJERCICIO DE LA CLASE PASADA

Todas las fuentes

```
coef_table <- df_long %>%
  filter(TWh > 0) %>%
  split(~Source) %>%
  map_dfr(fit_exp, .id = "Source")

coef_table
```

Source	a	b
Biofuels	5.323e−78	0.09160
Other renewables	1.070e−54	0.06533
Solar	3.741e−265	0.30520
Wind	9.006e−247	0.28500

EJERCICIO DE LA CLASE PASADA

Todas las fuentes

```
coef_table <- df_long %>%
  filter(TWh > 0) %>%
  split(~Source) %>%
  map_dfr(fit_exp, .id = "Source")

coef_table
```

Source	a	b
Biofuels	5.323e−78	0.09160
Other renewables	1.070e−54	0.06533
Solar	3.741e−265	0.30520
Wind	9.006e−247	0.28500

Notar que con veinte fuentes sería exactamente igual.

EJERCICIO DE LA CLASE PASADA

Un comentario: el tiempo de duplicación

EJERCICIO DE LA CLASE PASADA

Un comentario: el tiempo de duplicación

Con el ajuste $Y(X) = ae^{bX}$, buscamos el ΔX que hace que Y valga el doble:

$$Y(X + \Delta X) = 2Y(X)$$

EJERCICIO DE LA CLASE PASADA

Un comentario: el tiempo de duplicación

Con el ajuste $Y(X) = ae^{bX}$, buscamos el ΔX que hace que Y valga el doble:

$$Y(X + \Delta X) = 2Y(X)$$

$$ae^{b(X+\Delta X)} = 2ae^{bX} \quad \Rightarrow \quad e^{b\Delta X} = 2$$

EJERCICIO DE LA CLASE PASADA

Un comentario: el tiempo de duplicación

Con el ajuste $Y(X) = ae^{bX}$, buscamos el ΔX que hace que Y valga el doble:

$$Y(X + \Delta X) = 2Y(X)$$

$$ae^{b(X+\Delta X)} = 2ae^{bX} \quad \Rightarrow \quad e^{b\Delta X} = 2$$

$$\Delta X = \frac{\log(2)}{b}$$

EJERCICIO DE LA CLASE PASADA

Un comentario: el tiempo de duplicación

Con el ajuste $Y(X) = ae^{bX}$, buscamos el ΔX que hace que Y valga el doble:

$$Y(X + \Delta X) = 2Y(X)$$

$$ae^{b(X+\Delta X)} = 2ae^{bX} \quad \Rightarrow \quad e^{b\Delta X} = 2$$

$$\Delta X = \frac{\log(2)}{b}$$

Entonces $\frac{\log(2)}{b}$ nos dice cuánto tiene que variar X para que Y se duplique.
Notar que en esta expresión no aparecen X ni a, así que la duplicación tarda lo mismo en toda la curva.

EJERCICIO DE LA CLASE PASADA

Qué dice b

Lo calculamos para las cuatro fuentes:

```
coef_table %>%
  mutate(t2 = log(2) / b) %>%
  arrange(t2)
```

EJERCICIO DE LA CLASE PASADA

Qué dice b

Lo calculamos para las cuatro fuentes:

```
coef_table %>%
  mutate(t2 = log(2) / b) %>%
  arrange(t2)
```

Source	a	b	t2
Solar	3.741e−265	0.30520	2.271
Wind	9.006e−247	0.28500	2.432
Biofuels	5.323e−78	0.09160	7.567
Other renewables	1.070e−54	0.06533	10.610

EJERCICIO DE LA CLASE PASADA

Qué dice b

Lo calculamos para las cuatro fuentes:

```
coef_table %>%
  mutate(t2 = log(2) / b) %>%
  arrange(t2)
```

Source	a	b	t2
Solar	3.741e−265	0.30520	2.271
Wind	9.006e−247	0.28500	2.432
Biofuels	5.323e−78	0.09160	7.567
Other renewables	1.070e−54	0.06533	10.610

La energía solar y heólica duplican su consumo cada dos años y medio; los biocombustibles, cada siete y medio.

EJERCICIO DE LA CLASE PASADA

Los cuatro ajustes de una

EJERCICIO DE LA CLASE PASADA

Los cuatro ajustes de una

```
fits <- df_long %>%
  filter(TWh > 0) %>%
```

EJERCICIO DE LA CLASE PASADA

Los cuatro ajustes de una

```
fits <- df_long %>%
  filter(TWh > 0) %>%
  left_join(coef_table, by = "Source") %>%
```

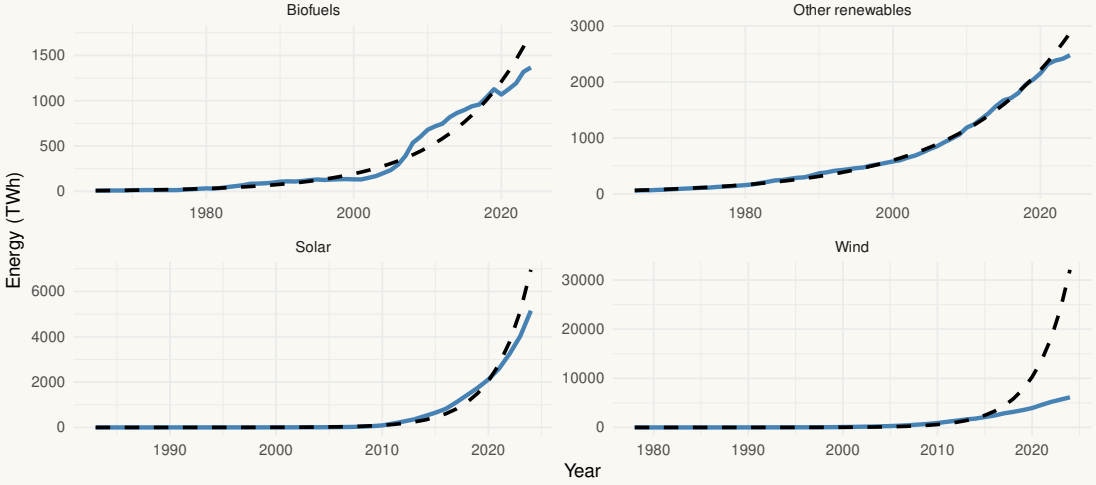
EJERCICIO DE LA CLASE PASADA

Los cuatro ajustes de una

```
fits <- df_long %>%
  filter(TWh > 0) %>%
  left_join(coef_table, by = "Source") %>%
  mutate(fit = a * exp(b * Year))

ggplot(fits, aes(x = Year, y = TWh)) +
  geom_line(color = "steelblue", linewidth = 1.2) +
  geom_line(aes(y = fit), color = "black", linetype = "dashed") +
  facet_wrap(~ Source, scales = "free") +
  labs(x = "Year", y = "Energy (TWh)") +
  theme_minimal()
```

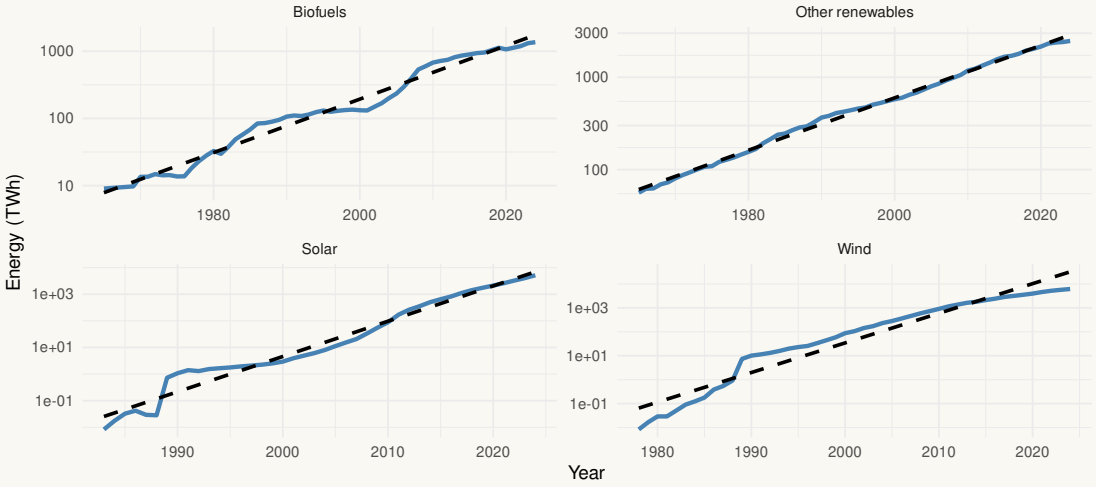
EJERCICIO DE LA CLASE PASADA

Los cuatro ajustes de una



EJERCICIO DE LA CLASE PASADA

Los cuatro ajustes en el `log`



EJERCICIO DE LA CLASE PASADA

¿Es razonable el modelo?

EJERCICIO DE LA CLASE PASADA

¿Es razonable el modelo?

```
resid_table <- df_long %>%
  filter(TWh > 0) %>%
```

EJERCICIO DE LA CLASE PASADA

¿Es razonable el modelo?

```
resid_table <- df_long %>%
  filter(TWh > 0) %>%
  split(~Source) %>%
```

EJERCICIO DE LA CLASE PASADA

¿Es razonable el modelo?

```
resid_table <- df_long %>%
  filter(TWh > 0) %>%
  split(~Source) %>%
  map_dfr(\(d) augment(lm(log(TWh) ~ Year, data = d)), .id = "Source")
```

EJERCICIO DE LA CLASE PASADA

¿Es razonable el modelo?

```
resid_table <- df_long %>%
  filter(TWh > 0) %>%
  split(~Source) %>%
  map_dfr(\(d) augment(lm(log(TWh) ~ Year, data = d)), .id = "Source")

resid_table %>% select(Source, Year, .fitted, .resid) %>% head(4)
```

```
# A tibble: 4 × 4
  Source    Year .fitted   .resid
  <chr>    <int>   <dbl>    <dbl>
1 Biofuels  1965    2.06  0.139
2 Biofuels  1966    2.15  0.0672
3 Biofuels  1967    2.24 -0.00570
4 Biofuels  1968    2.34 -0.0793
```

EJERCICIO DE LA CLASE PASADA

Residuos Q-Q

```
ggplot(resid_table, aes(sample = .std.resid)) +
  stat_qq(alpha = 0.7, size = 0.9) +
```

EJERCICIO DE LA CLASE PASADA

Residuos Q-Q

```
ggplot(resid_table, aes(sample = .std.resid)) +
  stat_qq(alpha = 0.7, size = 0.9) +
  geom_abline(col = "#C95E61", linetype = "dashed") +
```

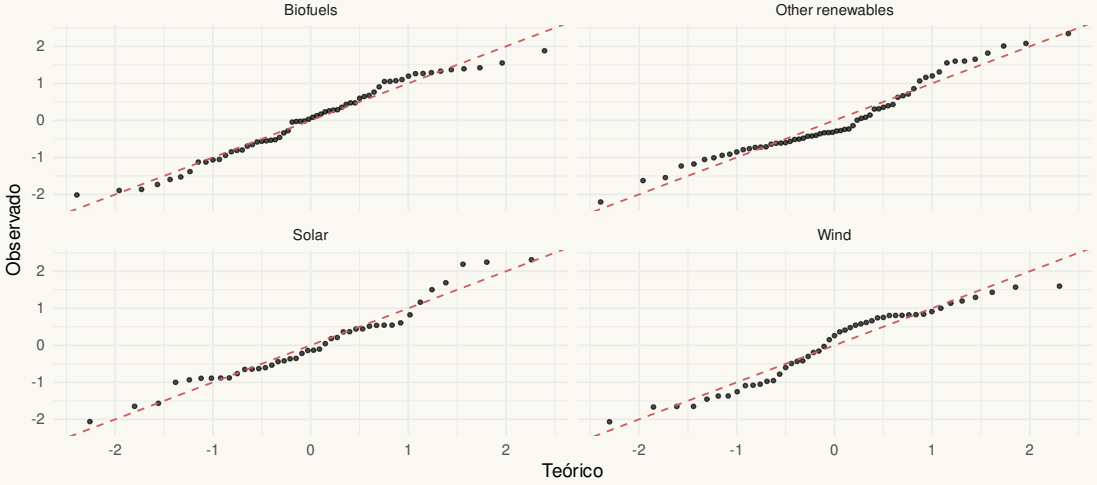
EJERCICIO DE LA CLASE PASADA

Residuos Q-Q

```
ggplot(resid_table, aes(sample = .std.resid)) +
  stat_qq(alpha = 0.7, size = 0.9) +
  geom_abline(col = "#C95E61", linetype = "dashed") +
  facet_wrap(~ Source) +
  labs(x = "Teórico", y = "Observado") +
  theme_minimal()
```

EJERCICIO DE LA CLASE PASADA

Residuos Q-Q



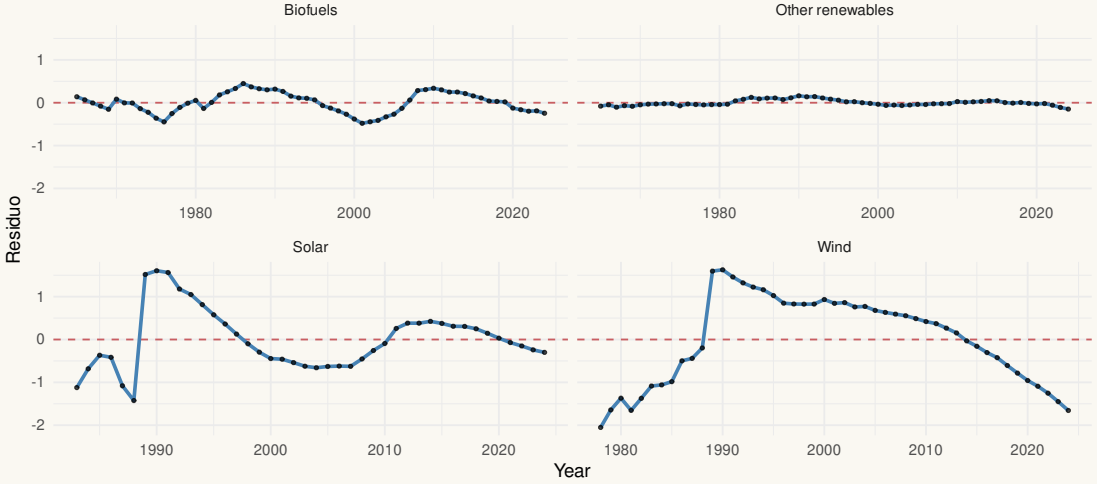
EJERCICIO DE LA CLASE PASADA

Residuos contra el año

```
ggplot(resid_table, aes(x = Year, y = .resid)) +
  geom_hline(yintercept = 0, linetype = "dashed", col = "#C95E61") +
  geom_line(color = "steelblue", linewidth = 1) +
  geom_point(size = 0.8, alpha = 0.7) +
  facet_wrap(~ Source, scales = "free_x") +
  labs(x = "Year", y = "Residuo") +
  theme_minimal()
```

EJERCICIO DE LA CLASE PASADA

Residuos contra el año



Regresión lineal a lo ML

EL PROBLEMA

Los datos de siempre

```
library(tidyverse)
```

EL PROBLEMA

Los datos de siempre

```
library(tidyverse)
```

```
# Igual que las clases pasadas
df <- read_delim("usu_individual_T126.txt", delim = ";") %>%
```

EL PROBLEMA

Los datos de siempre

```
library(tidyverse)
```

```
# Igual que las clases pasadas
df <- read_delim("usu_individual_T126.txt", delim = ";") %>%
  rename(age = CH06, salary = P21) %>%
```

EL PROBLEMA

Los datos de siempre

```
library(tidyverse)
```

```
# Igual que las clases pasadas
df <- read_delim("usu_individual_T126.txt", delim = ";") %>%
  rename(age = CH06, salary = P21) %>%
  filter(salary > 0, ESTADO == 1)
```

EL PROBLEMA

Los datos de siempre

```
library(tidyverse)
```

```
# Igual que las clases pasadas
df <- read_delim("usu_individual_T126.txt", delim = ";") %>%
  rename(age = CH06, salary = P21) %>%
  filter(salary > 0, ESTADO == 1)

age_salary_18_40 <- df %>%
  filter(between(age, 18, 40)) %>%
```

EL PROBLEMA

Los datos de siempre

```
library(tidyverse)
```

```
# Igual que las clases pasadas
df <- read_delim("usu_individual_T126.txt", delim = ";") %>%
  rename(age = CH06, salary = P21) %>%
  filter(salary > 0, ESTADO == 1)

age_salary_18_40 <- df %>%
  filter(between(age, 18, 40)) %>%
  group_by(age) %>%
  summarise(mean_salary = mean(salary)) %>%
```

EL PROBLEMA

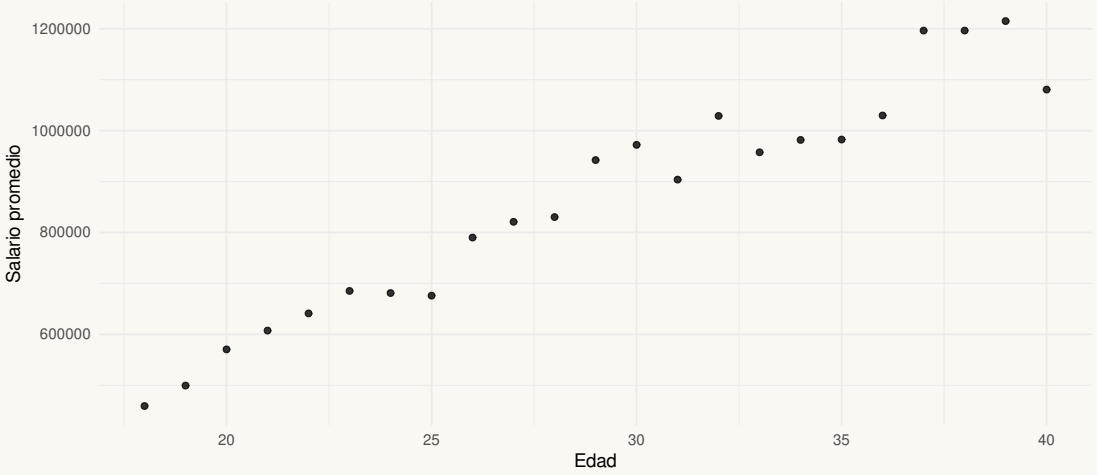
Los datos de siempre

```
library(tidyverse)
```

```
# Igual que las clases pasadas
df <- read_delim("usu_individual_T126.txt", delim = ";") %>%
  rename(age = CH06, salary = P21) %>%
  filter(salary > 0, ESTADO == 1)

age_salary_18_40 <- df %>%
  filter(between(age, 18, 40)) %>%
  group_by(age) %>%
  summarise(mean_salary = mean(salary)) %>%
  ungroup() %>%
  arrange(age)
```

EL PROBLEMA

Los datos de siempre



EL PROBLEMA

La recta que ya sabemos ajustar

En la clase 3 vimos que estos estimadores

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n}(x_i - \overline{x})^2}, \qquad \hat{\beta}_0 = \overline{y} - \hat{\beta}_1\,\overline{x}$$

son los *mejores* que se pueden construir, y que `lm()` los calcula por nosotros.

EL PROBLEMA

La recta que ya sabemos ajustar

En la clase 3 vimos que estos estimadores

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n}(x_i - \overline{x})^2}, \qquad \hat{\beta}_0 = \overline{y} - \hat{\beta}_1\,\overline{x}$$

son los *mejores* que se pueden construir, y que `lm()` los calcula por nosotros.

Veamos si podemos llegar a la misma recta sin usar la fórmula analítica.

EL PROBLEMA

El MSE

Para decidir qué tan buena es una recta usamos el **error cuadrático medio**:

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left(\hat{Y}_i - Y_i\right)^2 = \mathrm{E}\Big(\left(\hat{\mathbf{Y}} - \mathbf{Y}\right)^2\Big)$$

EL PROBLEMA

El MSE

Para decidir qué tan buena es una recta usamos el **error cuadrático medio**:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left(\hat{Y}_i - Y_i\right)^2 = \text{E}\Big(\left(\hat{\mathbf{Y}} - \mathbf{Y}\right)^2\Big)$$

Nuestro objetivo es encontrar los parámetros (β_0, β_1) que minimicen

$$\text{MSE}(\beta_0, \beta_1 \mid \mathbf{X}, \mathbf{Y}) = \text{E}\Big((\beta_0 + \beta_1\mathbf{X} - \mathbf{Y})^2\Big)$$

EL PROBLEMA

El MSE en código

```
compute_mse_lm <- function(par, x_data, y_data) {
  b0 <- par[1]
  b1 <- par[2]
```

EL PROBLEMA

El MSE en código

```
compute_mse_lm <- function(par, x_data, y_data) {
  b0 <- par[1]
  b1 <- par[2]

  y_hat <- b0 + b1 * x_data
```

EL PROBLEMA

El MSE en código

```
compute_mse_lm <- function(par, x_data, y_data) {
  b0 <- par[1]
  b1 <- par[2]

  y_hat <- b0 + b1 * x_data

  diff <- y_hat - y_data
  mean(diff^2)
}
```

EL PROBLEMA

El MSE en código

```
x_data <- age_salary_18_40$age
y_data <- age_salary_18_40$mean_salary

# el MSE de la recta que da lm()
compute_mse_lm(c(-72557.64, 32110.37), x_data, y_data)
```

```
[1] 2846611934
```

Estrategia 1 - Rectas al azar

Muchas rectas al azar

ESTRATEGIA 1 - RECTAS AL AZAR

Muchas rectas al azar

La primera estrategia consiste en **sortear muchas rectas y quedarnos con la de menor MSE**.

ESTRATEGIA 1 - RECTAS AL AZAR

Muchas rectas al azar

La primera estrategia consiste en **sortear muchas rectas y quedarnos con la de menor MSE**.

- Partimos de la recta que une el primer punto del dataset con el último. No es un buen ajuste, pero nos da el orden de magnitud de (β_0, β_1).

ESTRATEGIA 1 - RECTAS AL AZAR

Muchas rectas al azar

La primera estrategia consiste en **sortear muchas rectas y quedarnos con la de menor MSE**.

- Partimos de la recta que une el primer punto del dataset con el último. No es un buen ajuste, pero nos da el orden de magnitud de (β_0, β_1).
- Sorteamos muchos pares (β_0, β_1) alrededor de esa base.

ESTRATEGIA 1 - RECTAS AL AZAR

Muchas rectas al azar

La primera estrategia consiste en **sortear muchas rectas y quedarnos con la de menor MSE**.

- Partimos de la recta que une el primer punto del dataset con el último. No es un buen ajuste, pero nos da el orden de magnitud de (β_0, β_1).
- Sorteamos muchos pares (β_0, β_1) alrededor de esa base.
- Calculamos el MSE de cada recta y nos quedamos con el mínimo.

La recta base

```
n <- nrow(age_salary_18_40)
```

La recta base

```
n <- nrow(age_salary_18_40)

# pendiente entre el primer punto y el último
b1_guess <- (y_data[n] - y_data[1]) / (x_data[n] - x_data[1])
```

La recta base

```
n <- nrow(age_salary_18_40)

# pendiente entre el primer punto y el último
b1_guess <- (y_data[n] - y_data[1]) / (x_data[n] - x_data[1])

# y la ordenada que le corresponde
b0_guess <- y_data[1] - b1_guess * x_data[1]
```

La recta base

```
n <- nrow(age_salary_18_40)

# pendiente entre el primer punto y el último
b1_guess <- (y_data[n] - y_data[1]) / (x_data[n] - x_data[1])

# y la ordenada que le corresponde
b0_guess <- y_data[1] - b1_guess * x_data[1]

cat(b0_guess, b1_guess)
```

```
-49111.19 28243.75
```

ESTRATEGIA 1 - RECTAS AL AZAR

Mil rectas

```
n_lines <- 1000

random_lines <- tibble(
  b0 = runif(n_lines, b0_guess - 1.5 * abs(b0_guess),
                      b0_guess + 1.5 * abs(b0_guess)),
```

ESTRATEGIA 1 - RECTAS AL AZAR

Mil rectas

```
n_lines <- 1000

random_lines <- tibble(
  b0 = runif(n_lines, b0_guess - 1.5 * abs(b0_guess),
                      b0_guess + 1.5 * abs(b0_guess)),
  b1 = runif(n_lines, b1_guess - 1.5 * abs(b1_guess),
                      b1_guess + 1.5 * abs(b1_guess))
)
```

ESTRATEGIA 1 - RECTAS AL AZAR

Mil rectas

```
n_lines <- 1000

random_lines <- tibble(
  b0 = runif(n_lines, b0_guess - 1.5 * abs(b0_guess),
                      b0_guess + 1.5 * abs(b0_guess)),
  b1 = runif(n_lines, b1_guess - 1.5 * abs(b1_guess),
                      b1_guess + 1.5 * abs(b1_guess))
)
head(random_lines)
```

```
# A tibble: 6 × 2
      b0     b1
   <dbl>  <dbl>
1   9336.  8377.
2 -83660. 30854.
3 -67952. 43907.
4 -38377. 18354.
5  11032. 66795.
6 -93063. -4093.
```

Mil rectas

```
ggplot(age_salary_18_40, aes(age, mean_salary)) +
  # todas las rectas sorteadas
  geom_abline(data = random_lines, aes(intercept = b0, slope = b1),
              alpha = 0.15) +
```

ESTRATEGIA 1 - RECTAS AL AZAR

Mil rectas

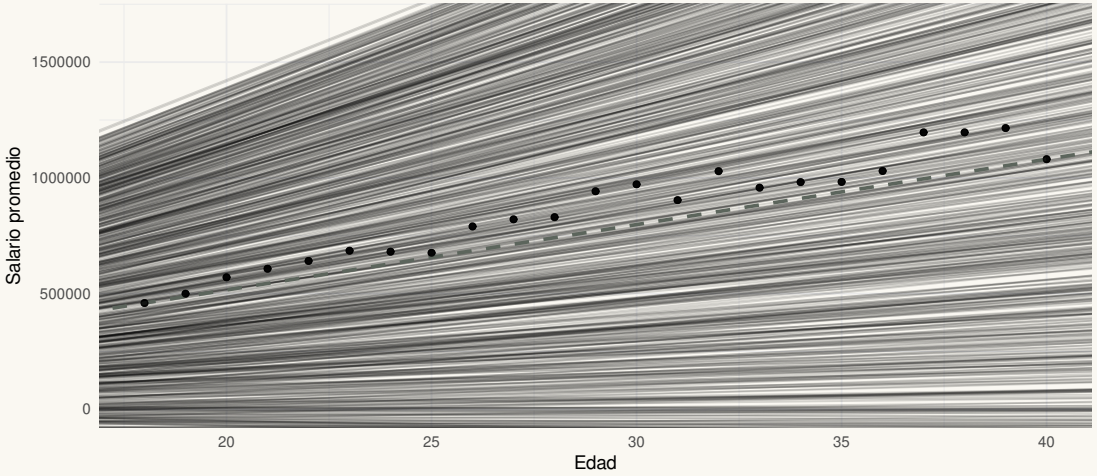
```
ggplot(age_salary_18_40, aes(age, mean_salary)) +
  # todas las rectas sorteadas
  geom_abline(data = random_lines, aes(intercept = b0, slope = b1),
              alpha = 0.15) +

  # la recta base
  geom_abline(intercept = b0_guess, slope = b1_guess,
              linewidth = 1, col = "#5C655C", linetype = "dashed") +
```

Mil rectas

```
ggplot(age_salary_18_40, aes(age, mean_salary)) +
  # todas las rectas sorteadas
  geom_abline(data = random_lines, aes(intercept = b0, slope = b1),
              alpha = 0.15) +

  # la recta base
  geom_abline(intercept = b0_guess, slope = b1_guess,
              linewidth = 1, col = "#5C655C", linetype = "dashed") +

  # los puntos al final, para que queden arriba de las rectas
  geom_point(alpha = 0.8) +
  theme_minimal()
```

ESTRATEGIA 1 - RECTAS AL AZAR

Mil rectas



El MSE de cada una

Queremos aplicar `compute_mse_lm` a cada fila de `random_lines`:

```
random_lines <- random_lines %>%
  mutate(
    mse = map2_dbl(b0, b1, \(b0, b1) compute_mse_lm(c(b0, b1), x_data, y_data))
  )
```

ESTRATEGIA 1 - RECTAS AL AZAR

El MSE de cada una

Queremos aplicar `compute_mse_lm` a cada fila de `random_lines`:

```
random_lines <- random_lines %>%
  mutate(
    mse = map2_dbl(b0, b1, \(b0, b1) compute_mse_lm(c(b0, b1), x_data, y_data))
  )

head(random_lines)
```

```
# A tibble: 6 × 3
      b0     b1     mse
   <dbl>  <dbl>   <dbl>
1   9336.  8377. 3.95e11
2 -83660. 30854. 5.18e 9
3 -67952. 43907. 1.29e11
4 -38377. 18354. 1.44e11
5  11032. 66795. 1.24e12
6 -93063. -4093. 1.21e12
```

ESTRATEGIA 1 - RECTAS AL AZAR

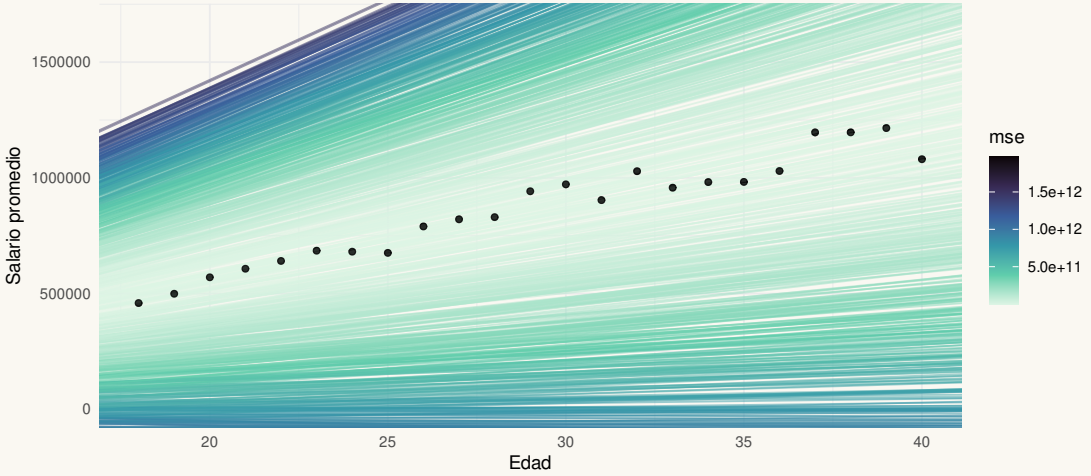
Pintemos las rectas

Usamos el MSE para pintar cada recta:

```
ggplot(age_salary_18_40, aes(age, mean_salary)) +
  geom_abline(data = random_lines,
              aes(intercept = b0, slope = b1, color = mse),
              alpha = 0.5) +
  geom_point(alpha = 0.8) +
  scale_color_viridis_c(option = "mako", direction = -1) +
  theme_minimal()
```

ESTRATEGIA 1 - RECTAS AL AZAR

Pintemos las rectas



ESTRATEGIA 1 - RECTAS AL AZAR

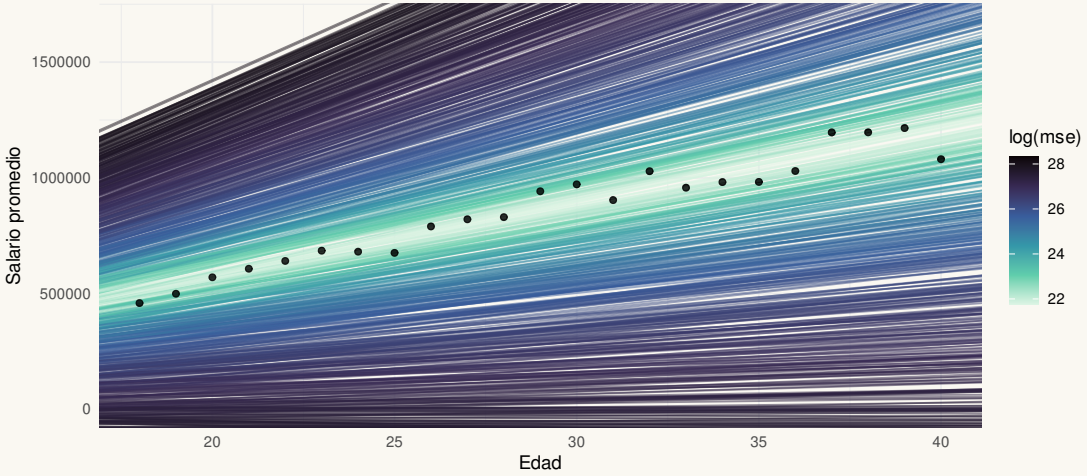
Pintemos las rectas (en log)

El MSE se reparte sobre varios órdenes de magnitud. Lo ponemos en log-scale:

```
                aes(intercept = b0, slope = b1, color = log(mse)),
```

ESTRATEGIA 1 - RECTAS AL AZAR

Pintemos las rectas (en log)



ESTRATEGIA 1 - RECTAS AL AZAR

La recta que minimiza el MSE

```
best_line <- random_lines %>% slice_min(order_by = mse, n = 1)
best_line
```

```
# A tibble: 1 × 3
       b0     b1         mse
    <dbl>  <dbl>       <dbl>
1 -91448. 32861. 2879689461.
```

ESTRATEGIA 1 - RECTAS AL AZAR

La recta que minimiza el MSE

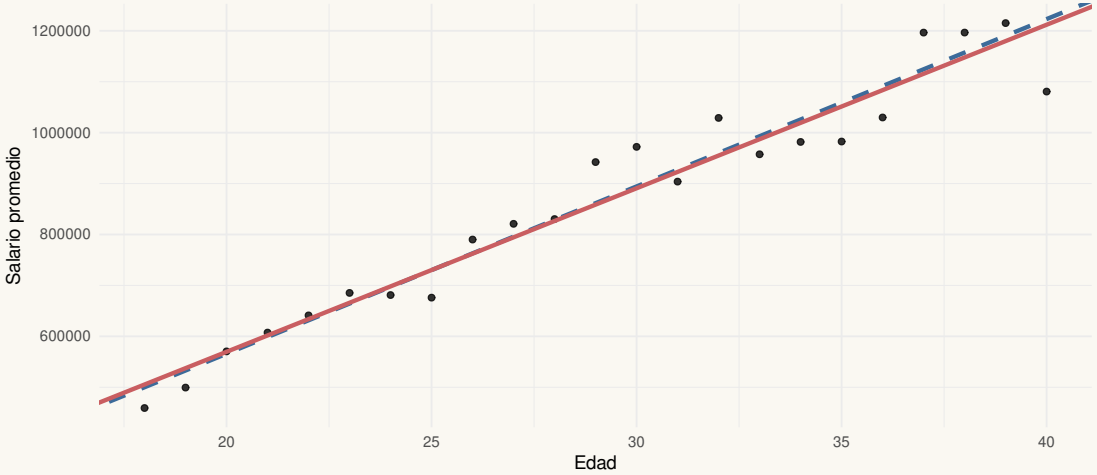
```
best_line <- random_lines %>% slice_min(order_by = mse, n = 1)
best_line
```

```
# A tibble: 1 × 3
       b0     b1         mse
    <dbl>  <dbl>       <dbl>
1 -91448. 32861. 2879689461.
```

```
mod <- lm(formula = mean_salary ~ age, data = age_salary_18_40)
tidy(mod)
```

term	estimate	std.error	statistic	p.value
(Intercept)	−72560	52220	−1.39	1.792e−01
age	32110	1755	18.29	2.212e−14

La recta que minimiza el MSE



Al azar: $b_0 = -91448, b_1 = 32861$. Analítica: $b_0 = -72558, b_1 = 32110$.

Estrategia 2 - Gradiente a mano

ESTRATEGIA 2 - GRADIENTE A MANO

Descenso por el gradiente

ESTRATEGIA 2 - GRADIENTE A MANO

Descenso por el gradiente

En lugar de evaluar rectas sueltas, calculamos **cómo cambiar** los parámetros para que el MSE baje, y los ajustamos en esa dirección.

ESTRATEGIA 2 - GRADIENTE A MANO

Descenso por el gradiente

En lugar de evaluar rectas sueltas, calculamos **cómo cambiar** los parámetros para que el MSE baje, y los ajustamos en esa dirección.

Para eso necesitamos el gradiente de $\mathrm{MSE}(\beta_0, \beta_1)$.

ESTRATEGIA 2 - GRADIENTE A MANO

Descenso por el gradiente

En lugar de evaluar rectas sueltas, calculamos **cómo cambiar** los parámetros para que el MSE baje, y los ajustamos en esa dirección.

Para eso necesitamos el gradiente de $\mathrm{MSE}(\beta_0, \beta_1)$. Como en general no tenemos su fórmula exacta, lo estimamos por diferencias finitas:

$$[\nabla f(\mathbf{x})]_i \approx \frac{f(\mathbf{x} + h\,\mathbf{e}_i) - f(\mathbf{x})}{h}, \quad i = 1, \ldots, n$$

ESTRATEGIA 2 - GRADIENTE A MANO

El gradiente numérico

```
numerical_grad <- function(fn, x, h = 1e-6) {
  n <- length(x)
  g <- numeric(n)
  fn_x <- fn(x)
```

ESTRATEGIA 2 - GRADIENTE A MANO

El gradiente numérico

```
numerical_grad <- function(fn, x, h = 1e-6) {
  n <- length(x)
  g <- numeric(n)
  fn_x <- fn(x)

  for (i in seq_len(n)) {
    x_p_h <- x
    x_p_h[i] <- x_p_h[i] + h
```

ESTRATEGIA 2 - GRADIENTE A MANO

El gradiente numérico

```
numerical_grad <- function(fn, x, h = 1e-6) {
  n <- length(x)
  g <- numeric(n)
  fn_x <- fn(x)

  for (i in seq_len(n)) {
    x_p_h <- x
    x_p_h[i] <- x_p_h[i] + h

    g[i] <- (fn(x_p_h) - fn_x) / h
  }
```

ESTRATEGIA 2 - GRADIENTE A MANO

El gradiente numérico

```
numerical_grad <- function(fn, x, h = 1e-6) {
  n <- length(x)
  g <- numeric(n)
  fn_x <- fn(x)

  for (i in seq_len(n)) {
    x_p_h <- x
    x_p_h[i] <- x_p_h[i] + h

    g[i] <- (fn(x_p_h) - fn_x) / h
  }

  g
}
```

ESTRATEGIA 2 - GRADIENTE A MANO

El gradiente numérico

Nos hace falta una función objetivo que dependa **solo** de los parámetros:

```
obj_fn <- function(par) compute_mse_lm(par, x_data, y_data)
```

ESTRATEGIA 2 - GRADIENTE A MANO

El gradiente numérico

Nos hace falta una función objetivo que dependa **solo** de los parámetros:

```
obj_fn <- function(par) compute_mse_lm(par, x_data, y_data)

numerical_grad(obj_fn, c(0, 0))
```

```
[1]  -1717285 -52627075
```

ESTRATEGIA 2 - GRADIENTE A MANO

El gradiente numérico

Nos hace falta una función objetivo que dependa **solo** de los parámetros:

```
obj_fn <- function(par) compute_mse_lm(par, x_data, y_data)

numerical_grad(obj_fn, c(0, 0))
```

```
[1]  -1717285 -52627075
```

En realidad en este este caso, y en muchos otros, el gradiente se puede calcular a mano, pero vamos a hacer como si no lo conociéramos para ver un caso más general.

El algoritmo

Partimos de un punto inicial $\mathbf{x}_0$ (en nuestro caso la tupla (β_0, β_1)) y actualizamos los parámetros en la dirección **opuesta** al gradiente:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta \nabla f(\mathbf{x}_k)$$

El algoritmo

Partimos de un punto inicial $\mathbf{x}_0$ (en nuestro caso la tupla (β_0, β_1)) y actualizamos los parámetros en la dirección **opuesta** al gradiente:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta \nabla f(\mathbf{x}_k)$$

$\nabla f(\mathbf{x}_k)$ es la dirección de mayor crecimiento, y η, el *learning rate*, fija el tamaño del paso.

ESTRATEGIA 2 - GRADIENTE A MANO

El algoritmo en código

```
gradient_descent <- function(fn, par0, learning_rate, n_iter) {
  par <- par0
  f_prev <- fn(par)
  path <- tibble(b0 = par[1], b1 = par[2], mse = f_prev)
```

ESTRATEGIA 2 - GRADIENTE A MANO

El algoritmo en código

```
gradient_descent <- function(fn, par0, learning_rate, n_iter) {
  par <- par0
  f_prev <- fn(par)
  path <- tibble(b0 = par[1], b1 = par[2], mse = f_prev)

  for (k in seq_len(n_iter)) {
    g <- numerical_grad(fn, par)
    new_par <- par - learning_rate * g
    f_new <- fn(new_par)
```

ESTRATEGIA 2 - GRADIENTE A MANO

El algoritmo en código

```
gradient_descent <- function(fn, par0, learning_rate, n_iter) {
  par <- par0
  f_prev <- fn(par)
  path <- tibble(b0 = par[1], b1 = par[2], mse = f_prev)

  for (k in seq_len(n_iter)) {
    g <- numerical_grad(fn, par)
    new_par <- par - learning_rate * g
    f_new <- fn(new_par)

    # solo actualizo los parámetros si el MSE bajó
    if (f_new < f_prev) {
      par <- new_par
      f_prev <- f_new
    }
```

ESTRATEGIA 2 - GRADIENTE A MANO

El algoritmo en código

```
gradient_descent <- function(fn, par0, learning_rate, n_iter) {
  par <- par0
  f_prev <- fn(par)
  path <- tibble(b0 = par[1], b1 = par[2], mse = f_prev)

  for (k in seq_len(n_iter)) {
    g <- numerical_grad(fn, par)
    new_par <- par - learning_rate * g
    f_new <- fn(new_par)

    # solo actualizo los parámetros si el MSE bajó
    if (f_new < f_prev) {
      par <- new_par
      f_prev <- f_new
    }

    path <- rbind(path, tibble(b0 = par[1], b1 = par[2], mse = f_prev))
  }
```

ESTRATEGIA 2 - GRADIENTE A MANO

El algoritmo en código

```
gradient_descent <- function(fn, par0, learning_rate, n_iter) {
  par <- par0
  f_prev <- fn(par)
  path <- tibble(b0 = par[1], b1 = par[2], mse = f_prev)

  for (k in seq_len(n_iter)) {
    g <- numerical_grad(fn, par)
    new_par <- par - learning_rate * g
    f_new <- fn(new_par)

    # solo actualizo los parámetros si el MSE bajó
    if (f_new < f_prev) {
      par <- new_par
      f_prev <- f_new
    }

    path <- rbind(path, tibble(b0 = par[1], b1 = par[2], mse = f_prev))
  }

  list(par = par, value = f_prev, path = path)
}
```

La primera corrida

```
res <- gradient_descent(
  fn = obj_fn,
  par0 = c(0, 0),
  learning_rate = 1e-4,
  n_iter = 5000
)
res$par
```

```
[1] -2589.394 29817.500
```

La primera corrida

```
res <- gradient_descent(
  fn = obj_fn,
  par0 = c(0, 0),
  learning_rate = 1e-4,
  n_iter = 5000
)
res$par
```

```
[1] -2589.394 29817.500
```

```
coef(mod)
```

```
(Intercept)         age
  -72557.64    32110.37
```

ESTRATEGIA 2 - GRADIENTE A MANO

La primera corrida

```
res <- gradient_descent(
  fn = obj_fn,
  par0 = c(0, 0),
  learning_rate = 1e-4,
  n_iter = 5000
)
res$par
```

```
[1] -2589.394 29817.500
```

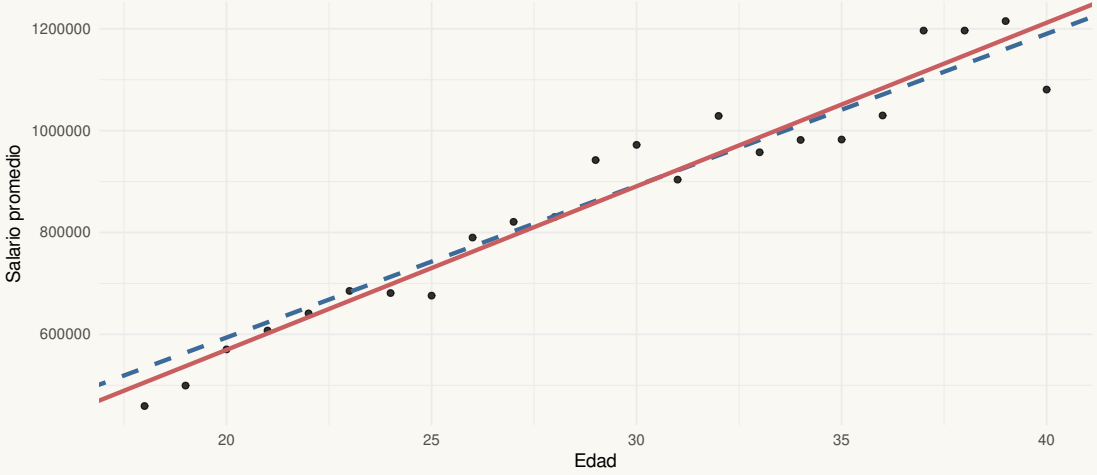
```
coef(mod)
```

```
(Intercept)         age
  -72557.64    32110.37
```

La pendiente no está tan mal; la ordenada al origen está muy lejos.

ESTRATEGIA 2 - GRADIENTE A MANO

La primera corrida



La superficie del MSE

Miremos la **superficie** del MSE alrededor de la trayectoria junto al camino del algoritmo.

ESTRATEGIA 2 - GRADIENTE A MANO

La superficie del MSE

Miremos la **superficie** del MSE alrededor de la trayectoria junto al camino del algoritmo.

```
library(plotly)
```

La superficie del MSE

Miremos la **superficie** del MSE alrededor de la trayectoria junto al camino del algoritmo.

```
library(plotly)

# grilla alrededor de la trayectoria, con un 20% de margen
pad_b0 <- 0.2 * diff(range(res$path$b0))
pad_b1 <- 0.2 * diff(range(res$path$b1))
```

ESTRATEGIA 2 - GRADIENTE A MANO

La superficie del MSE

Miremos la **superficie** del MSE alrededor de la trayectoria junto al camino del algoritmo.

```
library(plotly)

# grilla alrededor de la trayectoria, con un 20% de margen
pad_b0 <- 0.2 * diff(range(res$path$b0))
pad_b1 <- 0.2 * diff(range(res$path$b1))

b0_seq <- seq(min(res$path$b0) - pad_b0, max(res$path$b0) + pad_b0,
              length.out = 120)
b1_seq <- seq(min(res$path$b1) - pad_b1, max(res$path$b1) + pad_b1,
              length.out = 120)
```

ESTRATEGIA 2 - GRADIENTE A MANO

La superficie del MSE

Miremos la **superficie** del MSE alrededor de la trayectoria junto al camino del algoritmo.

```
library(plotly)

# grilla alrededor de la trayectoria, con un 20% de margen
pad_b0 <- 0.2 * diff(range(res$path$b0))
pad_b1 <- 0.2 * diff(range(res$path$b1))

b0_seq <- seq(min(res$path$b0) - pad_b0, max(res$path$b0) + pad_b0,
              length.out = 120)
b1_seq <- seq(min(res$path$b1) - pad_b1, max(res$path$b1) + pad_b1,
              length.out = 120)

# plotly indexa z como [y, x], de ahí la transpuesta
z <- t(outer(b0_seq, b1_seq, Vectorize(\(b0, b1) obj_fn(c(b0, b1)))))
```

La superficie del MSE

```
plot_ly(x = ~b0_seq, y = ~b1_seq, z = ~z) %>%
  add_surface(opacity = 0.9, showscale = FALSE) %>%
```

ESTRATEGIA 2 - GRADIENTE A MANO

La superficie del MSE

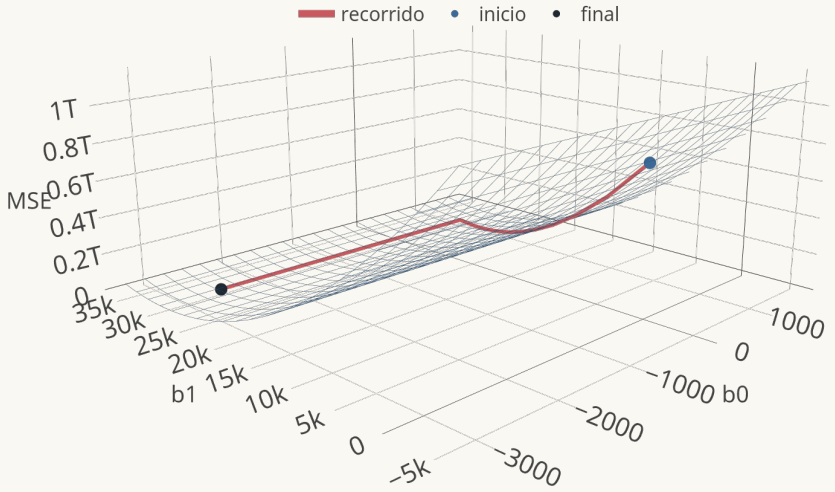
```
plot_ly(x = ~b0_seq, y = ~b1_seq, z = ~z) %>%
  add_surface(opacity = 0.9, showscale = FALSE) %>%

  # la trayectoria
  add_trace(x = res$path$b0, y = res$path$b1, z = res$path$mse,
            type = "scatter3d", mode = "lines",
            line = list(color = "#C95E61", width = 6)) %>%
```

La superficie del MSE

```
plot_ly(x = ~b0_seq, y = ~b1_seq, z = ~z) %>%
  add_surface(opacity = 0.9, showscale = FALSE) %>%

  # la trayectoria
  add_trace(x = res$path$b0, y = res$path$b1, z = res$path$mse,
            type = "scatter3d", mode = "lines",
            line = list(color = "#C95E61", width = 6)) %>%

  # el punto inicial y el final
  add_markers(x = res$path$b0[1], y = res$path$b1[1], z = res$path$mse[1],
              name = "inicio") %>%
  add_markers(x = tail(res$path$b0, 1), y = tail(res$path$b1, 1),
              z = tail(res$path$mse, 1), name = "final")
```

ESTRATEGIA 2 - GRADIENTE A MANO

El camino del algoritmo



El efecto del learning rate

Problemos darle un learning rate más grande:

El efecto del learning rate

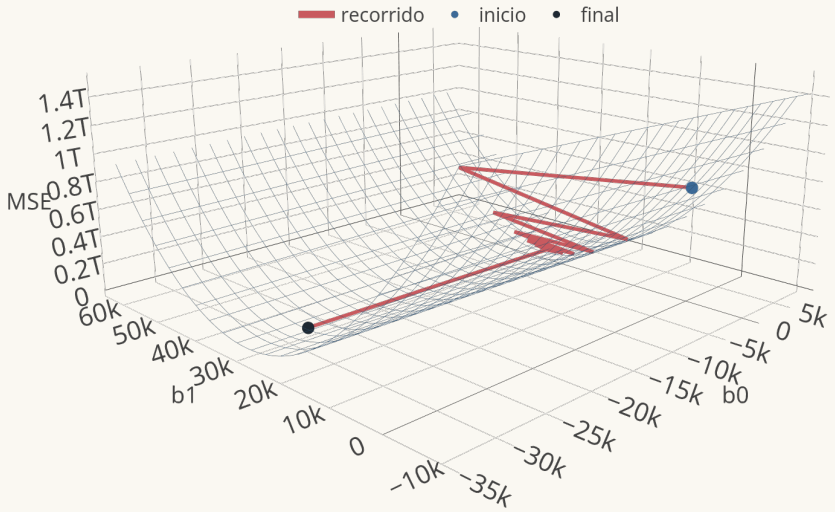
Problemos darle un learning rate más grande:

```
res_large <- gradient_descent(obj_fn, par0 = c(0, 0),
                              learning_rate = 1e-3, n_iter = 5000)
res_large$par
```

```
[1] -27810.07  30643.98
```

ESTRATEGIA 2 - GRADIENTE A MANO

El efecto del learning rate



ESTRATEGIA 2 - GRADIENTE A MANO

Un paso más grande todavía

```
res_too_large <- gradient_descent(obj_fn, par0 = c(0, 0),
                                  learning_rate = 2e-3, n_iter = 5000)
res_too_large$par
```

```
[1] 0 0
```

ESTRATEGIA 2 - GRADIENTE A MANO

Un paso más grande todavía

```
res_too_large <- gradient_descent(obj_fn, par0 = c(0, 0),
                                  learning_rate = 2e-3, n_iter = 5000)
res_too_large$par
```

```
[1] 0 0
```

Con $\eta = 2 \times 10^{-3}$ el primer paso ya se pasa del otro lado del valle y el MSE sube.
El condicional `if (f_new < f_prev)` lo descarta, el gradiente vuelve a ser el mismo, y así las 5000 veces.
El algoritmo no se mueve del punto inicial.

ESTRATEGIA 2 - GRADIENTE A MANO

Paso anterior pero 50.000 ieraciones

```
res_long <- gradient_descent(obj_fn, par0 = c(0, 0),
                             learning_rate = 1e-3, n_iter = 50000)
res_long$par
```

```
[1] -72045.27  32093.58
```

```
coef(mod)
```

```
(Intercept)         age
  -72557.64    32110.37
```

Estrategia 3 - `optim()`

optim()

Hay algoritmos de este mismo tipo mucho más estables y eficientes: mejor estimación de las derivadas, paso variable, estimación de la curvatura.

optim()

Hay algoritmos de este mismo tipo mucho más estables y eficientes: mejor estimación de las derivadas, paso variable, estimación de la curvatura.

En R accedemos a ellos con `optim()`, y le pasamos la función tal como la escribimos al principio:

```
res_optim <- optim(
  par = c(0, 0),
  fn = compute_mse_lm,   # la función que queremos minimizar
  x_data = x_data, y_data = y_data,
  method = "BFGS"        # el algoritmo de optimización
)
res_optim$par
```

```
[1] -72557.65  32110.37
```

optim()

```
coef(mod)
```

```
(Intercept)         age
  -72557.64    32110.37
```

optim()

```
coef(mod)
```

```
(Intercept)         age
  -72557.64    32110.37
```

Los mismos números que la fórmula cerrada, con dos decimales de diferencia, y solo 21 evaluaciones.

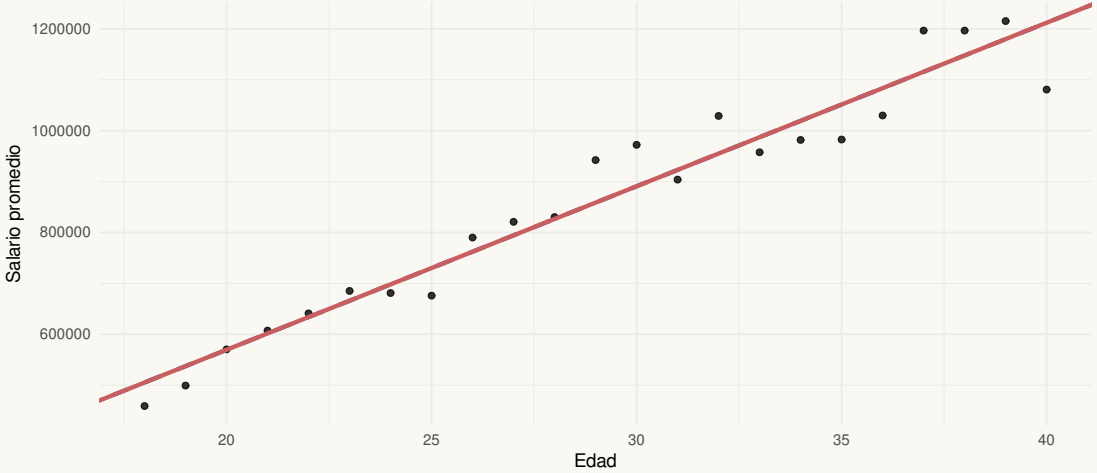
optim()

```
coef(mod)
```

```
(Intercept)         age
  -72557.64    32110.37
```

Los mismos números que la fórmula cerrada, con dos decimales de diferencia, y solo 21 evaluaciones.

optim()



Las dos rectas están dibujadas, pero una tapa a la otra.

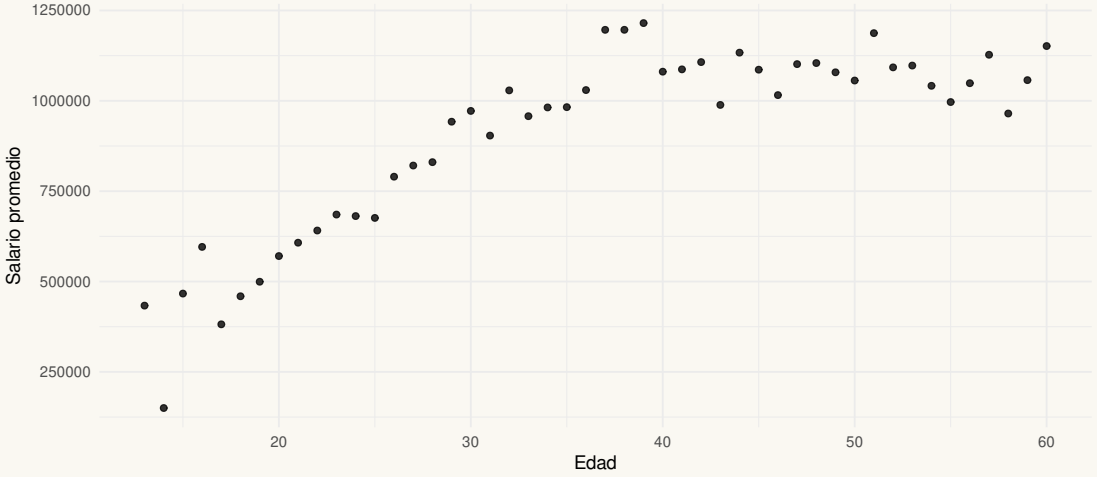
Modelos no lineales

Todas las edades

`optim()` no sabe nada de rectas; minimiza la función que le pasemos, así que podemos cambiar el modelo sin cambiar el método. Probemos sobre el rango completo de edades.

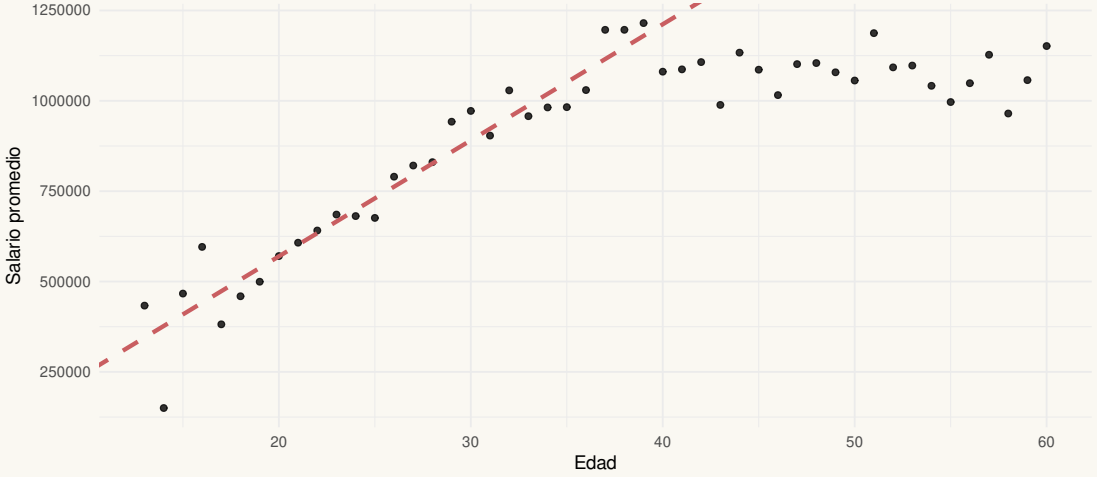
```
age_salary_0_60 <- df %>%
  filter(between(age, 0, 60)) %>%
  group_by(age) %>%
  summarise(mean_salary = mean(salary)) %>%
  ungroup() %>%
  arrange(age)
```

Todas las edades



MODELOS NO LINEALES

La recta de 18 a 40 fuera de su rango



Un modelo no lineal

La función que proponemos es

$$\mathrm{E}(Y|X) = a - \exp(-b\,x + c)$$

Un modelo no lineal

La función que proponemos es

$$\mathrm{E}(Y|X) = a - \exp(-b\,x + c)$$

donde a, b y c son parámetros libres que queremos optimizar.

Un modelo no lineal

La función que proponemos es

$$\mathrm{E}(Y|X) = a - \exp(-b\,x + c)$$

donde a, b y c son parámetros libres que queremos optimizar.

Crece rápido al principio, se va frenando y tiende a a, que hace de techo salarial.
No existe fórmula cerrada para la estimación de estos tres parámetros.

Un modelo no lineal

```
compute_mse_sat_exp <- function(par, x_data, y_data) {
  a <- par[1]
  b <- par[2]
  c <- par[3]
```

Un modelo no lineal

```
compute_mse_sat_exp <- function(par, x_data, y_data) {
  a <- par[1]
  b <- par[2]
  c <- par[3]

  y_hat <- a - exp(-b * x_data + c)
```

Un modelo no lineal

```
compute_mse_sat_exp <- function(par, x_data, y_data) {
  a <- par[1]
  b <- par[2]
  c <- par[3]

  y_hat <- a - exp(-b * x_data + c)

  diff <- y_hat - y_data
  mean(diff^2)
}
```

Un modelo no lineal

```
compute_mse_sat_exp <- function(par, x_data, y_data) {
  a <- par[1]
  b <- par[2]
  c <- par[3]

  y_hat <- a - exp(-b * x_data + c)

  diff <- y_hat - y_data
  mean(diff^2)
}
```

Lo único que cambió respecto de `compute_mse_lm` es la línea de `y_hat`.

Un modelo con techo

```
res_sat_exp <- optim(
  par = c(max(age_salary_0_60$mean_salary), 0, 0),
  fn = compute_mse_sat_exp,
  x_data = age_salary_0_60$age,
  y_data = age_salary_0_60$mean_salary,
  method = "BFGS"
)
```

Un modelo con techo

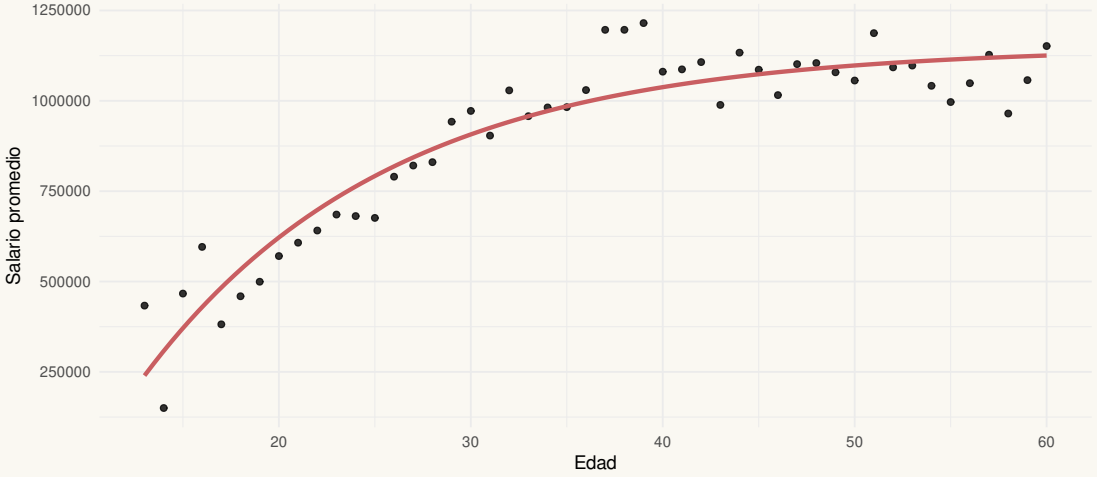
```
res_sat_exp <- optim(
  par = c(max(age_salary_0_60$mean_salary), 0, 0),
  fn = compute_mse_sat_exp,
  x_data = age_salary_0_60$age,
  y_data = age_salary_0_60$mean_salary,
  method = "BFGS"
)

a <- res_sat_exp$par[1]
b <- res_sat_exp$par[2]
c <- res_sat_exp$par[3]
cat(a, b, c)
```

```
1148666 0.0779341 14.73315
```

Un modelo con techo

```
ggplot(age_salary_0_60, aes(x = age, y = mean_salary)) +
  geom_point(alpha = 0.8) +
  geom_function(fun = \(x) a - exp(-b * x + c),
                linewidth = 1.2, col = "#C95E61") +
  theme_minimal()
```

Un modelo con techo



Esto es todo.