TP1

April 17, 2023

0.1 Primera parte

El objetivo de esta sección es deducir una fórmula para la solución óptima β^* siguiendo los pasos a continuación:

a) Mostrar que el espacio columna de la matriz X es un subespacio vectorial de $\mathbb{R}^n$:

$$Col(X) = \{b \text{ en } \mathbb{R}^n \text{ tales que b = X con variando en } \mathbb{R}^p\}$$

Para mostrar que $Col(X)$ es un subespacio vectorial de $\mathbb{R}^n$, debemos demostrar que $Col(X)$ cumple con las siguientes propiedades:

1. El vector cero pertenece al espacio $Col(X)$.

$$Col(X) = Gen\{x_1, x_2, ..., x_p\}$$

El vector 0 está en $Col(X)$ porque $0 * x_1 + 0 * x_2$ es combinación lineal de los vectores de %Col(X)%.

2. La suma de dos vectores pertenecientes a $Col(X)$ también pertenece a $Col(X)$.

Sean u, v vectores en $Col(X)$, s_1, s_2 y t_1, t_2 escalares tales que:

$$u = s_1x_1 + s_2x_2 \iff u \in Col(X)$$

Por ser combinación lineal de los vectores de $Col(X)$.

$$v = t_1x_1 + t_2x_2 \iff v \in Col(X)$$

Por ser combinación lineal de los vectores de $Col(X)$.

Entonces:

$$u + v = s_1x_1 + s_2x_2 + t_1x_1 + t_2x_2$$
$$u + v = (s_1 + t_1)x_1 + (s_2 + t_2)x_2$$

Llamamos $c_1 = s_1 + t_1$ y $c_2 = s_2 + t_2$.

Entonces:

$$u + v = c_1x_1 + c_2x_2$$

$$c_1x_1 \in Col(X) \land c_2x_2 \in Col(X) \implies c_1x_1 + c_2x_2 \in Col(X) \implies u + v \in Col(X)$$

3. El producto de un escalar por un vector perteneciente a $Col(X)$ también pertenece a $Col(X)$.

Sea u un vector en $Col(X)$, s_1, s_2 escalares tales que:

$$u = s_1x_1 + s_2x_2 \iff u \in Col(X)$$

Llamemos c al escalar.

Entonces:

$$cu = c * (s_1x_1 + s_2x_2)$$
$$cu = (c * s_1)x_1 + (c * s_2)x_2$$

Llamemos $c_1 = c * s_1$ y $c_2 = c * s_2$

Entonces:

$$cu = c_1x_1 + c_2x_2$$

$$c_1x_1 \in Col(X) \land c_2x_2 \in Col(X) \implies c_1x_1 + c_2x_2 \in Col(X) \implies cu \in Col(X)$$

Por lo tanto $Col(X)$ es un subespacio vectorial de $\mathbb{R}^n$.

b) Supongamos que cuando hablamos de vectores en $\mathbb{R}^n$ nos referimos a vectores columna de $\mathbb{R}^{n\times 1}$. Mostrar en ese caso que el producto escalar entre dos vectores u, v en $\mathbb{R}^n$ puede calcularse como:

$$u \cdot v = v^\top u$$

donde operación en el lado derecho de la igualdad es el producto de matrices usual.

Sean u, v vectores columna en $\mathbb{R}^n$.

$$u = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}, v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

Entonces:

$$v^\top = \begin{bmatrix} v_1 & v_2 & ... & v_n \end{bmatrix}$$

$$u \cdot v = \begin{bmatrix} v_1 & v_2 & ... & v_n \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} = \sum_{i=1}^{n} v_i u_i$$

Ademas, como el producto vectorial es conmutativo:

$$u \cdot v = v \cdot u \implies v \cdot u = u^\top v = \begin{bmatrix} u_1 & u_2 & ... & u_n \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \sum_{i=1}^{n} u_i v_i$$

$$\sum_{i=1}^{n} u_i v_i = \begin{bmatrix} u_1 & u_2 & ... & u_n \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

c) Aplicando el teorema tomando como subespacio S el subespacio del ítem (a), el punto y de $\mathbb{R}^n$ como el vector de la variable dependiente, y el vector b como $b = X\beta^*$, convertir esta ecuación de optimalidad

$$||y - X\beta^*|| = \min_{\beta \text{ en } \mathbb{R}^p} ||y - X\beta||$$

en la condición de ortogonalidad que corresponde a la equivalencia 2 del teorema.

$$\forall \beta \in \mathbb{R}^p, (\mathrm{y} - \mathrm{X}^*) \cdot \mathrm{X} = 0$$

Llamemos $t = X\beta^*$ y $s = X\beta$ donde s son todos los valores en el subespacio $S = X\beta$ con β variando en $\mathbb{R}^p$, y t es el vector $X\beta^*$ que es el vector que minimiza la distancia entre y y $X\beta$. t es la *proyección ortogonal* de y sobre el subespacio S.

Entonces por el Teroema. Sea y un vector cualquiera de $\mathbb{R}^n$ y S un subespacio de $\mathbb{R}^n$. El vector de S que minimiza la distancia del subespacio S al vector y es aquel b de S tal que $y - b$ es ortogonal a todo vector s de S. Es decir, las siguientes dos condiciones son equivalentes:

$$\begin{aligned} ||y - t|| &= \min_{s \in S} ||y - s|| \\ \implies (y - t) \cdot s &= 0, \forall s \in S \\ \implies (y - X\beta^*) \cdot X\beta &= 0, \forall \beta \in \mathbb{R}^p \end{aligned}$$

d) A la ecuación obtenida en el ítem (c), aplicarle la identidad del producto escalar vista en el item (b), para llegar a la ecuación:

$$X^\top (y - X\beta^*) \cdot \beta = 0$$

$$(y - X\beta^*) \cdot X\beta = 0$$

Llamemos $u = y - X\beta^*$ y $v = X\beta$ con $u \in \mathbb{R}^n$ y $v \in \mathbb{R}^n$.

Entonces por la propiedad del producto escalar del item (b):

$$\begin{aligned} u \cdot v = v^\top u &\iff (X\beta)^\top (y - X\beta^*) = 0 \\ &\iff \beta^\top X^\top (y - X\beta^*) = 0 \end{aligned}$$

Ahora llamamos $w^\top = \beta^\top$ y $z = X^\top(y - X\beta^*)$, y por la misma propiedad del item (b):

$$\begin{aligned} w^\top z = z \cdot w &\iff 0 = \beta^\top (X^\top (y - X\beta^*)) \\ &\qquad = (X^\top (y - X\beta^*)) \cdot \beta \end{aligned}$$

$$\iff X^\top (y - X\beta^*) \cdot \beta = 0$$

e) Se sabe que el único vector que es ortogonal a todo vector v de $\mathbb{R}^n$ es el vector nulo. Es decir, si u es un vector fijo tal que $u \cdot v = 0$ para todo v en $\mathbb{R}^n$, entonces $u = 0$. Usando esto y la ecuación obtenida en el ítem (d), llegar a la fórmula:

$$X^\top X\beta^* = X^\top y$$

Partimos de la igualdad obtenida en el ítem (d):

$$\begin{aligned} 0 &= X^\top (y - X\beta^*) \cdot \beta \\ &= (X^\top y - X^\top X\beta^*) \cdot \beta \end{aligned}$$

Fijamos $(X^\top y - X^\top X\beta^*) = 0$ ya que es el único vector que es ortogonal a todo vector β de $\mathbb{R}^p$.

$$\begin{aligned} &(X^\top y - X^\top X\beta^*) = 0 \\ &\iff X^\top X\beta^* = X^\top y \end{aligned}$$

f) Finalmente, suponiendo que las columnas de X son linealmente independientes, se tiene que la matriz $X^\top X$ es invertible. Despejar β^* de la ecuación del ítem (e) para llegar a la fórmula de la solución óptima al problema de regresión.

$$\begin{aligned} X^\top X\beta^* &= X^\top \\ (X^\top X)^{-1} X^\top X\beta^* &= (X^\top X)^{-1} X^\top \\ \beta^* &= (X^\top X)^{-1} X^\top y \end{aligned}$$

0.2 Segunda parte.

En esta sección la idea es realizar regresión lineal en $\mathbb{R}^2$ y analizar como se comportan las soluciones obtenidas.

0.2.1 1. Usando los datos del archivo ejercicio_1.csv:

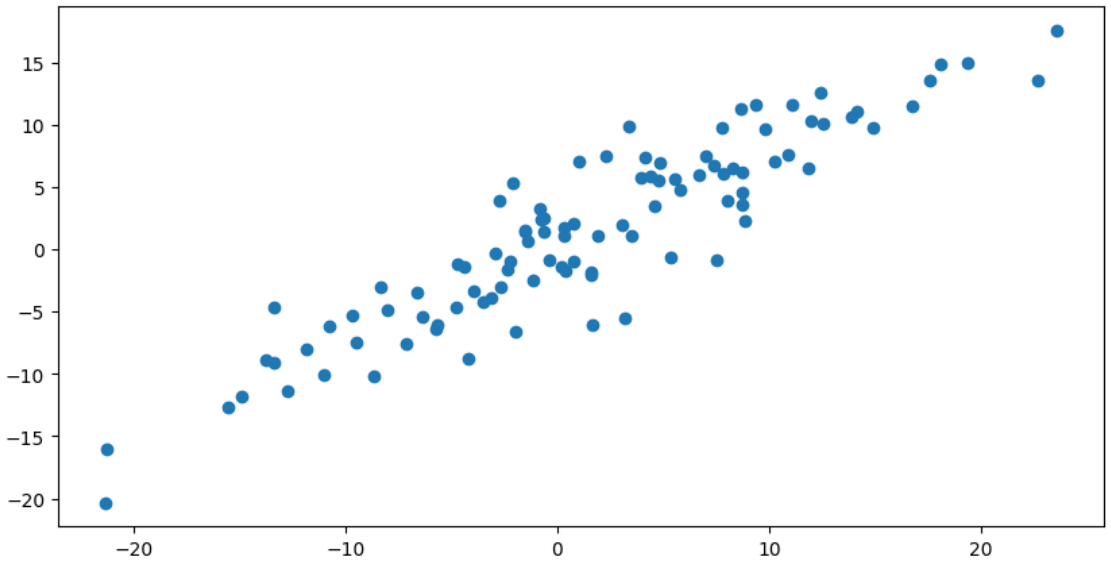
a) Graficar todos los puntos en el plano xy. Nota: La primer columna del archivo marca el valor de x y la segunda el valor de y de cada punto. Recomendamos usar la biblioteca pandas para leer los archivos con la función read_csv.

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from typing import Callable, List, Tuple
```

[ ]:
```
data_ej1: pd.DataFrame = pd.read_csv('data/ejercicio_1.csv', sep=',')
x_key: pd.Index = data_ej1.keys()[0]
y_key: pd.Index = data_ej1.keys()[1]

x_values: pd.DataFrame = data_ej1[x_key]
y_values: pd.DataFrame = data_ej1[y_key]
```

[ ]:
```
plt.figure(figsize=(10, 5))
plt.plot(x_values, y_values, 'o')
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x7fc200c99550>]
```



b) Utilizando los conceptos teóricos desarrollados en la primera parte, hallar la recta que mejor aproxima a los datos.

$$||y - X\beta^*|| = \min_{\beta \text{ en } \mathbb{R}^p} ||y - X\beta|| \beta^* = (X^\top X)^{-1} X^\top y$$

[ ]:
```
X: np.ndarray = np.array(x_values).reshape(-1, 1)
y: np.ndarray = np.array(y_values).reshape(-1, 1)
X.shape, y.shape
```

[ ]:
```
((100, 1), (100, 1))
```

[ ]:
```
def linear_regression(X: np.ndarray, y: np.ndarray) -> np.ndarray:
    """
    Args:
        X (np.ndarray): X values
        y (np.ndarray): y values

    Returns:
        np.ndarray: B values for the linear regression, B = (X^T X)^-1 X^T y,␣
 ↪without 1s in the first column
    """
    X_t = X.T
    X_t_X = np.dot(X_t, X)
    X_t_X_inv = np.linalg.inv(X_t_X)
    X_t_X_inv_X_t = np.dot(X_t_X_inv, X_t)
    B = np.dot(X_t_X_inv_X_t, y)
    return B
```

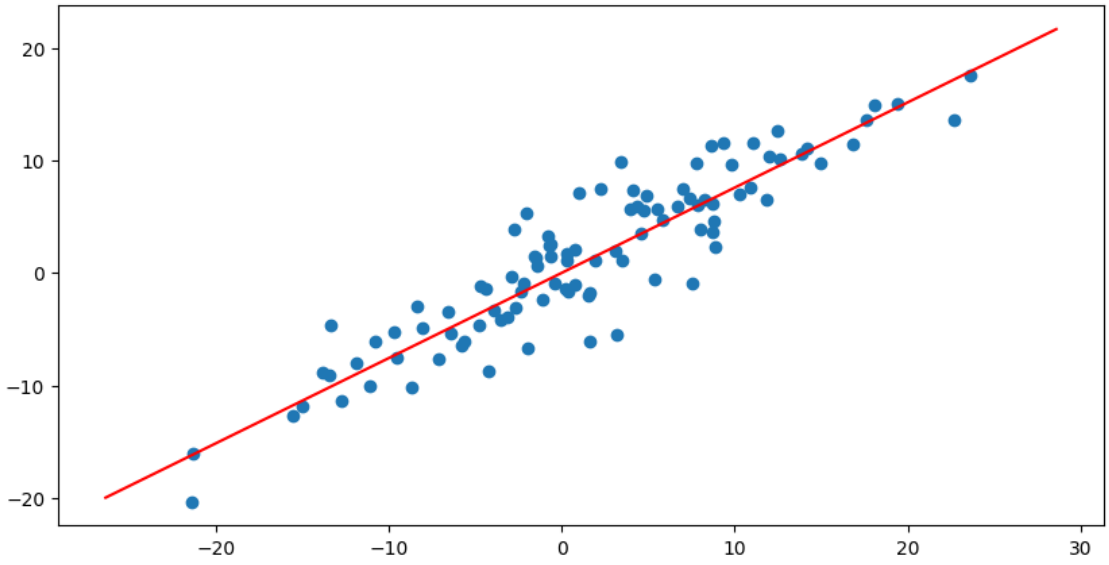
[ ]:
```
b = linear_regression(X, y)
b
```

[ ]:
```
array([[0.75785414]])
```

$$
\begin{aligned}
y &= X\beta^* \\
y &= \hat{\beta}_0 \times x \\
y &= 0.75785414 \times x
\end{aligned}
$$

[ ]:
```
def plot_regression_line(X: np.ndarray, y: np.ndarray, reg_func: Callable[[np.
 ↪ndarray, np.ndarray], np.ndarray] = linear_regression):
    """
    Args:
        X (_type_): _description_
        y (_type_): _description_
        reg_func (_type_, optional): _description_. Defaults to␣
 ↪linear_regression.
    """

    b = reg_func(X, y).flatten()

    plt.figure(figsize=(10, 5))

    plt.plot(X, y, 'o')

    min_x = np.min(X) - 5
    max_x = np.max(X) + 5

    line_x = np.linspace(min_x, max_x, 1000)
```

```
    if len(b) == 1:
        line_y = b[0] * line_x
    else:
        line_y = b[0] + b[1] * line_x

    plt.plot(line_x, line_y, 'r')

    plt.show()

plot_regression_line(X, y)
```



c) Realizar nuevamente los incisos (a) y (b) pero considerando los puntos

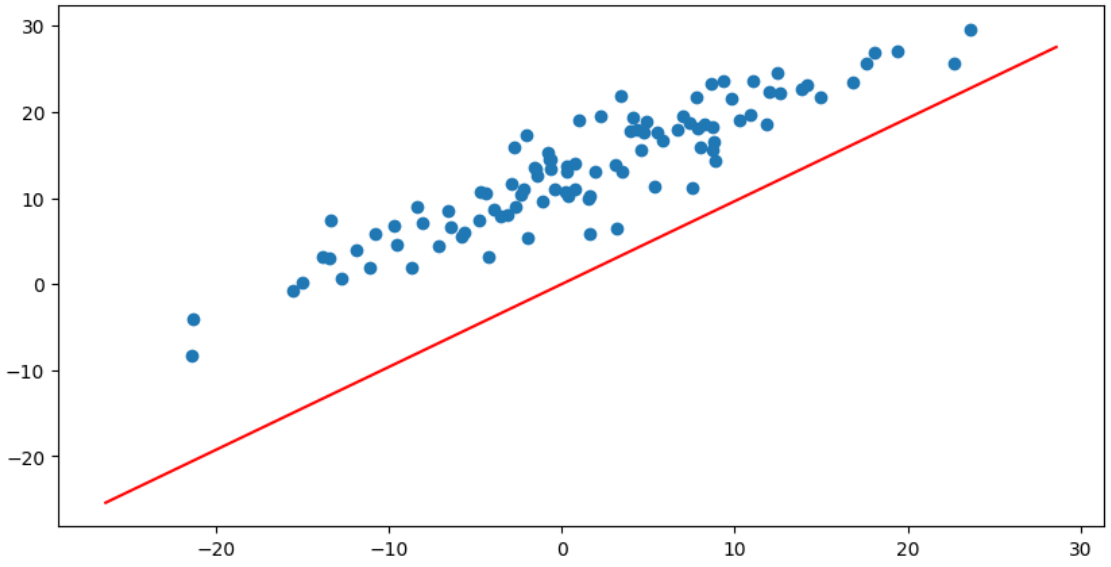
$$(x_i, y_i + 12) \text{ con } i = 1 \dots n$$

donde (x_i, y_i) eran los puntos originales. ¿Es buena la aproximación realizada?, ¿cuál es el problema?

[ ]:
```
linear_regression(X, y + 12)
```

[ ]:
```
array([[0.96332903]])
```

[ ]:
```
plot_regression_line(X, y + 12)
```



La recta aproximada no es buena ya que los puntos están alejados de la recta aproximada por 12 unidades. Esto es porque la regresión lineal no tiene en cuenta la ordenada al origen, solo la pendiente.

d) ¿Cómo se podría extender el modelo para poder aproximar cualquier recta en el plano?

Para resolverlo, se puede agregar una columna de 1's a la matriz X y agregar un coeficiente β_0 a la ecuación de la recta. De esta forma, se puede aproximar cualquier recta en el plano.

[ ]:
```python
def ones_column(X: np.ndarray) -> np.ndarray:
    """
    Args:
        X (np.ndarray): X values

    Returns:
        np.ndarray: X with 1s in the first column
    """
    ones = np.ones((X.shape[0], 1))
    new_X = np.concatenate((ones, X), axis=1)
    return new_X

def linear_regression_ones(X: np.ndarray, y: np.ndarray) -> np.ndarray:
    """
    Args:
        X (np.ndarray): X values
        y (np.ndarray): y values

    Returns:
```

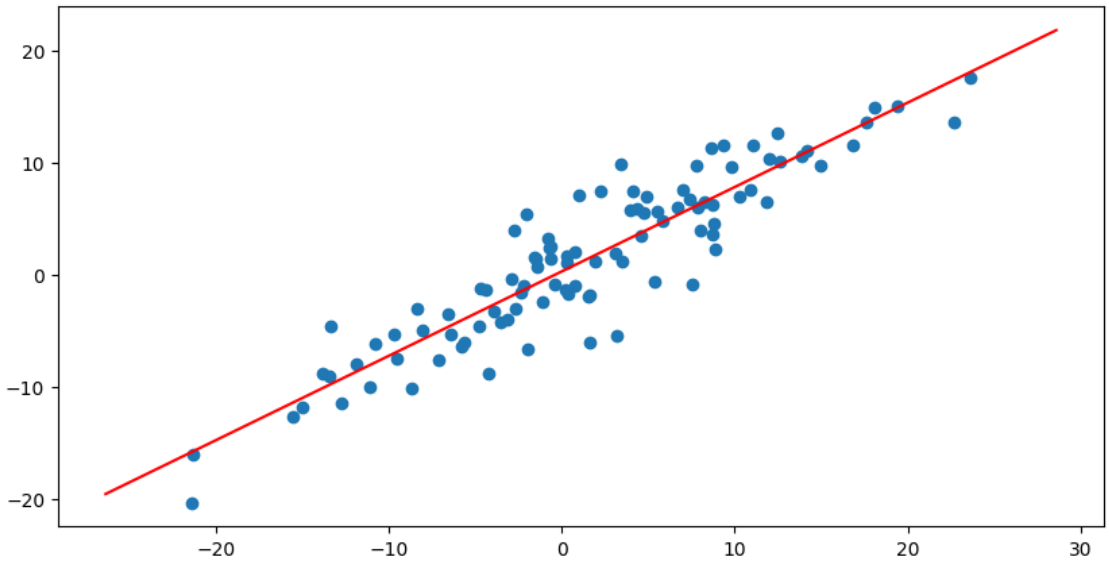
```
        np.ndarray: B values for the linear regression, B = (X^T X)^-1 X^T y,␣
↪with 1s in the first column
    """

    X = ones_column(X)
    X_t = X.T
    X_t_X = np.dot(X_t, X)
    X_t_X_inv = np.linalg.inv(X_t_X)
    X_t_X_inv_X_t = np.dot(X_t_X_inv, X_t)
    B = np.dot(X_t_X_inv_X_t, y)
    return B
```

[ ]:
```
linear_regression_ones(X, y)
```

[ ]:
```
array([[0.28565184],
       [0.75296295]])
```

$$y = \hat{\beta}_0 + \hat{\beta}_1 x$$
$$y = 0.28565184 + 0.75296295 \times x$$

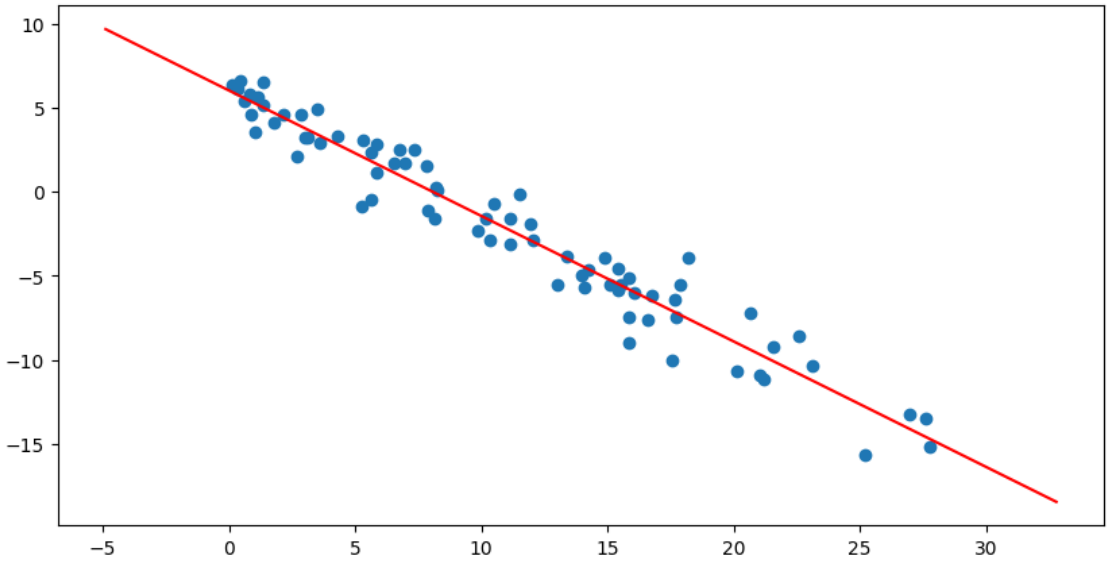
[ ]:
```
plot_regression_line(X, y, linear_regression_ones)
```



0.2.2 2. Usando los datos del archivo ejercicio_2.csv:

a) Graficar y aproximar los puntos con una recta.

```
[ ]: data_ej2: pd.DataFrame = pd.read_csv('data/ejercicio_2.csv', sep=',')

x_2_key: pd.Index = data_ej2.keys()[0]
y_2_key: pd.Index = data_ej2.keys()[1]

x_2_values: pd.DataFrame = data_ej2[x_2_key]
y_2_values: pd.DataFrame = data_ej2[y_2_key]

X2: np.ndarray = np.array(x_2_values).reshape(-1, 1)
y2: np.ndarray = np.array(y_2_values).reshape(-1, 1)

plot_regression_line(X2, y2, linear_regression_ones)
```



b) Imaginemos que los datos forman parte de mediciones de algún tipo, como por ejemplo la temperatura de un procesador a lo largo del tiempo), y queremos predecir cuál va a ser la temperatura en el futuro. ¿Es buena la aproximación que realizamos?, ¿cuál fue el problema en este caso?

0.3 Tercera parte. Regresión lineal en datos reales.

En esta sección utilizaremos el conjunto de datos provisto en Machine Learning Repository. Este consiste en datos de ventas de 414 casas en Taiwan. La información provista por casa es (en orden):

i) La fecha en que se realizó la transacción. Expresada en formato

$$\text{año} + \frac{\text{numero\_mes}}{12}$$

ii) La edad de la casa en años.

iii) La distancia a la estación de tren o subte más cercana en metros.

iv) La cantidad de almacenes alcanzables a pie.

v) La latitud en grados.

vi) La longitud en grados.

vii) El precio por Ping. La cual es una unidad utilizada en Taiwan que representa 3,3 metros cuadrados.

Vamos a dividir este conjunto de datos en dos:

i) Datos de entrenamiento: usamos los datos desde la observación 1 a la 315 inclusive.

ii) Datos de test: usamos los datos desde la observación 316 a la 414 inclusive.

[ ]:
```
real_estate: pd.DataFrame = pd.read_csv('data/real_estate.csv', sep=';')
real_estate.drop(['No'], axis=1, inplace=True)

X_matrix: np.ndarray = np.array(real_estate.drop(['Y house price of unit␣
 ↪area'], axis=1))#.reshape(-1, 6)
y_matrix: np.ndarray = np.array(real_estate['Y house price of unit area']).
 ↪reshape(-1, 1)

X_matrix.shape, y_matrix.shape
```

[ ]:
```
((414, 6), (414, 1))
```

[ ]:
```
y_matrix
```

[ ]:
```
array([[ 37.9],
       [ 42.2],
       [ 47.3],
       [ 54.8],
       [ 43.1],
       [ 32.1],
       [ 40.3],
       [ 46.7],
       [ 18.8],
       [ 22.1],
       [ 41.4],
       [ 58.1],
       [ 39.3],
       [ 23.8],
       [ 34.3],
       [ 50.5],
       [ 70.1],
       [ 37.4],
       [ 42.3],
       [ 47.7],
```

```
[ 29.3],
[ 51.6],
[ 24.6],
[ 47.9],
[ 38.8],
[ 27. ],
[ 56.2],
[ 33.6],
[ 47. ],
[ 57.1],
[ 22.1],
[ 25. ],
[ 34.2],
[ 49.3],
[ 55.1],
[ 27.3],
[ 22.9],
[ 25.3],
[ 47.7],
[ 46.2],
[ 15.9],
[ 18.2],
[ 34.7],
[ 34.1],
[ 53.9],
[ 38.3],
[ 42. ],
[ 61.5],
[ 13.4],
[ 13.2],
[ 44.2],
[ 20.7],
[ 27. ],
[ 38.9],
[ 51.7],
[ 13.7],
[ 41.9],
[ 53.5],
[ 22.6],
[ 42.4],
[ 21.3],
[ 63.2],
[ 27.7],
[ 55. ],
[ 25.3],
[ 44.3],
[ 50.7],
```

```
[ 56.8],
[ 36.2],
[ 42. ],
[ 59. ],
[ 40.8],
[ 36.3],
[ 20. ],
[ 54.4],
[ 29.5],
[ 36.8],
[ 25.6],
[ 29.8],
[ 26.5],
[ 40.3],
[ 36.8],
[ 48.1],
[ 17.7],
[ 43.7],
[ 50.8],
[ 27. ],
[ 18.3],
[ 48. ],
[ 25.3],
[ 45.4],
[ 43.2],
[ 21.8],
[ 16.1],
[ 41. ],
[ 51.8],
[ 59.5],
[ 34.6],
[ 51. ],
[ 62.2],
[ 38.2],
[ 32.9],
[ 54.4],
[ 45.7],
[ 30.5],
[ 71. ],
[ 47.1],
[ 26.6],
[ 34.1],
[ 28.4],
[ 51.6],
[ 39.4],
[ 23.1],
[  7.6],
```

```
[ 53.3],
[ 46.4],
[ 12.2],
[ 13. ],
[ 30.6],
[ 59.6],
[ 31.3],
[ 48. ],
[ 32.5],
[ 45.5],
[ 57.4],
[ 48.6],
[ 62.9],
[ 55. ],
[ 60.7],
[ 41. ],
[ 37.5],
[ 30.7],
[ 37.5],
[ 39.5],
[ 42.2],
[ 20.8],
[ 46.8],
[ 47.4],
[ 43.5],
[ 42.5],
[ 51.4],
[ 28.9],
[ 37.5],
[ 40.1],
[ 28.4],
[ 45.5],
[ 52.2],
[ 43.2],
[ 45.1],
[ 39.7],
[ 48.5],
[ 44.7],
[ 28.9],
[ 40.9],
[ 20.7],
[ 15.6],
[ 18.3],
[ 35.6],
[ 39.4],
[ 37.4],
[ 57.8],
```

```
[ 39.6],
[ 11.6],
[ 55.5],
[ 55.2],
[ 30.6],
[ 73.6],
[ 43.4],
[ 37.4],
[ 23.5],
[ 14.4],
[ 58.8],
[ 58.1],
[ 35.1],
[ 45.2],
[ 36.5],
[ 19.2],
[ 42. ],
[ 36.7],
[ 42.6],
[ 15.5],
[ 55.9],
[ 23.6],
[ 18.8],
[ 21.8],
[ 21.5],
[ 25.7],
[ 22. ],
[ 44.3],
[ 20.5],
[ 42.3],
[ 37.8],
[ 42.7],
[ 49.3],
[ 29.3],
[ 34.6],
[ 36.6],
[ 48.2],
[ 39.1],
[ 31.6],
[ 25.5],
[ 45.9],
[ 31.5],
[ 46.1],
[ 26.6],
[ 21.4],
[ 44. ],
[ 34.2],
```

```
[ 26.2],
[ 40.9],
[ 52.2],
[ 43.5],
[ 31.1],
[ 58. ],
[ 20.9],
[ 48.1],
[ 39.7],
[ 40.8],
[ 43.8],
[ 40.2],
[ 78.3],
[ 38.5],
[ 48.5],
[ 42.3],
[ 46. ],
[ 49. ],
[ 12.8],
[ 40.2],
[ 46.6],
[ 19. ],
[ 33.4],
[ 14.7],
[ 17.4],
[ 32.4],
[ 23.9],
[ 39.3],
[ 61.9],
[ 39. ],
[ 40.6],
[ 29.7],
[ 28.8],
[ 41.4],
[ 33.4],
[ 48.2],
[ 21.7],
[ 40.8],
[ 40.6],
[ 23.1],
[ 22.3],
[ 15. ],
[ 30. ],
[ 13.8],
[ 52.7],
[ 25.9],
[ 51.8],
```

```
[ 17.4],
[ 26.5],
[ 43.9],
[ 63.3],
[ 28.8],
[ 30.7],
[ 24.4],
[ 53. ],
[ 31.7],
[ 40.6],
[ 38.1],
[ 23.7],
[ 41.1],
[ 40.1],
[ 23. ],
[117.5],
[ 26.5],
[ 40.5],
[ 29.3],
[ 41. ],
[ 49.7],
[ 34. ],
[ 27.7],
[ 44. ],
[ 31.1],
[ 45.4],
[ 44.8],
[ 25.6],
[ 23.5],
[ 34.4],
[ 55.3],
[ 56.3],
[ 32.9],
[ 51. ],
[ 44.5],
[ 37. ],
[ 54.4],
[ 24.5],
[ 42.5],
[ 38.1],
[ 21.8],
[ 34.1],
[ 28.5],
[ 16.7],
[ 46.1],
[ 36.9],
[ 35.7],
```

```
[ 23.2],
[ 38.4],
[ 29.4],
[ 55. ],
[ 50.2],
[ 24.7],
[ 53. ],
[ 19.1],
[ 24.7],
[ 42.2],
[ 78. ],
[ 42.8],
[ 41.6],
[ 27.3],
[ 42. ],
[ 37.5],
[ 49.8],
[ 26.9],
[ 18.6],
[ 37.7],
[ 33.1],
[ 42.5],
[ 31.3],
[ 38.1],
[ 62.1],
[ 36.7],
[ 23.6],
[ 19.2],
[ 12.8],
[ 15.6],
[ 39.6],
[ 38.4],
[ 22.8],
[ 36.5],
[ 35.6],
[ 30.9],
[ 36.3],
[ 50.4],
[ 42.9],
[ 37. ],
[ 53.5],
[ 46.6],
[ 41.2],
[ 37.9],
[ 30.8],
[ 11.2],
[ 53.7],
```

```
[ 47. ],
[ 42.3],
[ 28.6],
[ 25.7],
[ 31.3],
[ 30.1],
[ 60.7],
[ 45.3],
[ 44.9],
[ 45.1],
[ 24.7],
[ 47.1],
[ 63.3],
[ 40. ],
[ 48. ],
[ 33.1],
[ 29.5],
[ 24.8],
[ 20.9],
[ 43.1],
[ 22.8],
[ 42.1],
[ 51.7],
[ 41.5],
[ 52.2],
[ 49.5],
[ 23.8],
[ 30.5],
[ 56.8],
[ 37.4],
[ 69.7],
[ 53.3],
[ 47.3],
[ 29.3],
[ 40.3],
[ 12.9],
[ 46.6],
[ 55.3],
[ 25.6],
[ 27.3],
[ 67.7],
[ 38.6],
[ 31.3],
[ 35.3],
[ 40.3],
[ 24.7],
[ 42.5],
```

```
        [ 31.9],
        [ 32.2],
        [ 23. ],
        [ 37.3],
        [ 35.5],
        [ 27.7],
        [ 28.5],
        [ 39.7],
        [ 41.2],
        [ 37.2],
        [ 40.5],
        [ 22.3],
        [ 28.1],
        [ 15.4],
        [ 50. ],
        [ 40.6],
        [ 52.5],
        [ 63.9]])
```

[ ]:
```
X_train: np.ndarray = X_matrix[0:315]
y_train: np.ndarray = y_matrix[0:315]

X_test: np.ndarray = X_matrix[315:]
y_test: np.ndarray = y_matrix[315:]

X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

[ ]:
```
((315, 6), (315, 1), (99, 6), (99, 1))
```

1. Teniendo en cuenta la teoría desarrollada en la primer parte del trabajo práctico y usando los datos de entrenamiento:

a) Estimar los parámetros $\hat{\beta}$ que minimizan el error cuadrático medio para este problema.

[ ]:
```
b_top: np.ndarray = linear_regression_ones(X_train, y_train)
b_top, b_top.shape
```

[ ]:
```
(array([[-1.48729480e+04],
        [ 4.96257468e+00],
        [-2.79853534e-01],
        [-4.29360382e-03],
        [ 1.11451659e+00],
        [ 2.63780784e+02],
        [-1.36639291e+01]]),
 (7, 1))
```

b) Encontrar $\hat{y}$ la estimación de la variable de respuesta.

[ ]:
```
X_train_ones: np.ndarray = ones_column(X_train)
y_pred: np.ndarray = np.dot(X_train_ones, b_top)
y_pred.shape
```

[ ]:
```
(315, 1)
```

c) ¿Cuánto vale el error cuadrático medio?

Definimos error cuadrático medio como

$$\text{ECM} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

donde y_i son observaciones de una variable y $\hat{y}_i$ estimaciones de las mismas.

[ ]:
```
def ECM(y: np.ndarray, y_pred: np.ndarray) -> float:
    """
    Args:
        y (np.ndarray): Valores reales
        y_pred (np.ndarray): Valores predichos

    Returns:
        float: Error Cuadrático Medio (ECM) entre "y" e "y_pred"
    """
    return np.sum((y - y_pred)**2) / len(y)

ECM(y_train, y_pred)
```

[ ]:
```
83.16551901819933
```

2. Utilizando los datos de test, analizar cuál es el error cuadrático medio al utilizar los parámetros $\hat{\beta}$ estimados en el punto anterior.

[ ]:
```
X_test_ones: np.ndarray = ones_column(X_test)
y_eval: np.ndarray = np.dot(X_test_ones, b_top)
ECM(y_test, y_eval), y_eval.shape
```

[ ]:
```
(58.664540889884876, (99, 1))
```

a) ¿Es la estimación mejor que sobre los datos originales?, ¿a qué se debe la discrepancia?

La estimación es mejor que sobre los datos originales ya que el error cuadrático medio es menor. Esto se debe a que los datos de entrenamiento son más cercanos a los datos de test que a los datos "originales".

b) ¿Qué sucede con el ECM del segundo conjunto de casas si se realiza la regresión sobre todos los datos al mismo tiempo (es decir, las 414 casas)?

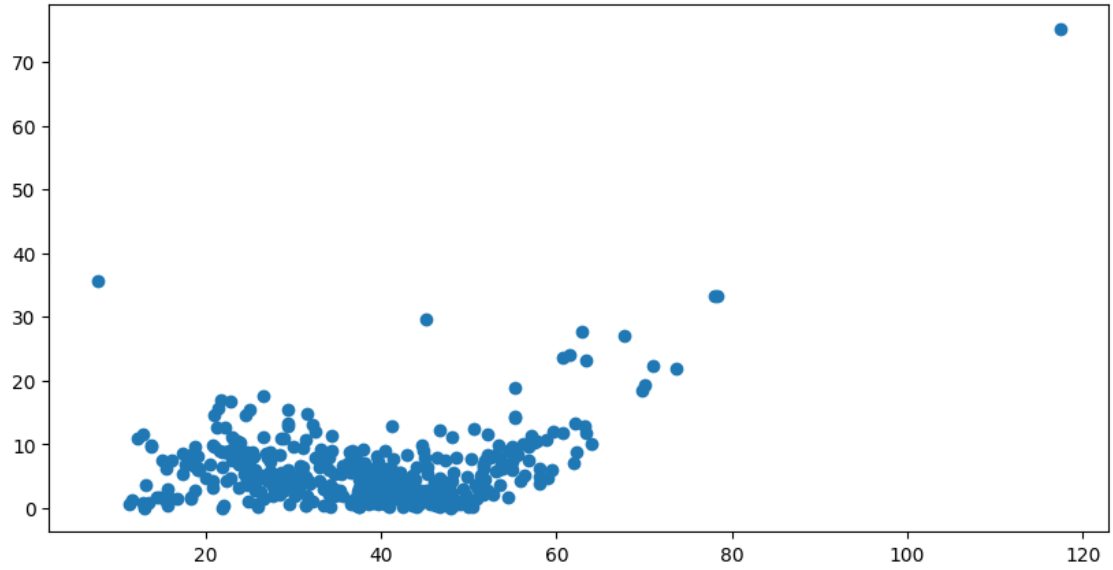
[ ]:
```
b_: np.ndarray = linear_regression_ones(X_matrix, y_matrix)
```

```
X_matrix_ones: np.ndarray = ones_column(X_matrix)

y_pred_: np.ndarray = np.dot(X_matrix_ones, b_)
ECM(y_matrix, y_pred_)
```

[ ]: 77.13171720187393

3. Graficar el error cometido por cada casa. Es decir el valor absoluto de la diferencia entre el precio por Ping real y el estimado.

[ ]:
```
y_matrix_plt: np.ndarray = y_matrix.T[0]
y_pred_plt: np.ndarray = y_pred_.T[0]

plt.figure(figsize=(10, 5))
plt.plot(y_matrix_plt, abs(y_matrix_plt - y_pred_plt), 'o')
plt.show()
```



4. Imaginemos que se agrega una nueva columna a los datos que informa el año en que la misma fue construida. ¿Disminuiría esto el ECM?

Como el año en el que la casa fue construida se puede obtener como combinación linea de la fecha de la transacción y la edad de la casa, se puede decir que son variables colineales. Por lo tanto, no se espera que disminuya el ECM.

[ ]:
```
trans_date: np.ndarray = np.array(real_estate['X1 transaction date']).
  ↪reshape(-1, 1)
age: np.ndarray = np.array(real_estate['X2 house age']).reshape(-1, 1)
anio: np.ndarray = trans_date - age
```

```
X_ext = np.concatenate((X_matrix, anio), axis=1)
X_ext.shape, y_matrix.shape
```

[ ]: ((414, 7), (414, 1))

```
b_ext = linear_regression_ones(X_ext, y_matrix)
b_ext, b_ext.shape
```

[ ]: (array([[-1.44166564e+04],
[-1.01805167e+01],
[1.50321368e+01],
[-4.48746137e-03],
[1.13327690e+00],
[2.25472973e+02],
[-1.24236291e+01],
[1.53123616e+01]]),
(8, 1))

```
y_pred_ext = np.dot(ones_column(X_ext), b_ext)
ECM(y_matrix, y_pred_ext)
```

[ ]: 152.8330448945791

Podemos ver que el ECM para el conjunto total de datos original X es menor que el ECM para el conjunto de datos X' extendido que incluye la columna de año de construcción. Por lo tanto, la colinealidad de la variable año de construcción respecto al resto de las variables del modelo afecta negativamente al ECM.