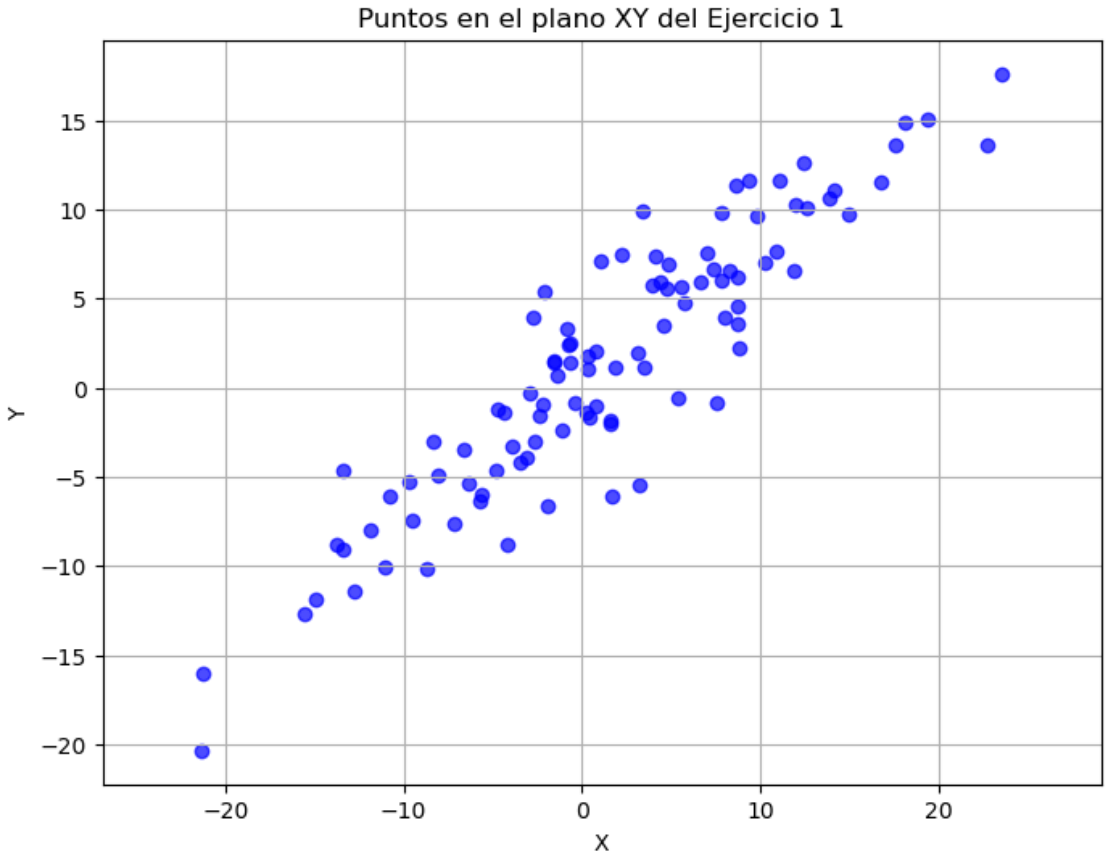
metodos

May 16, 2025

[23]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Ejercicio 1, inciso A

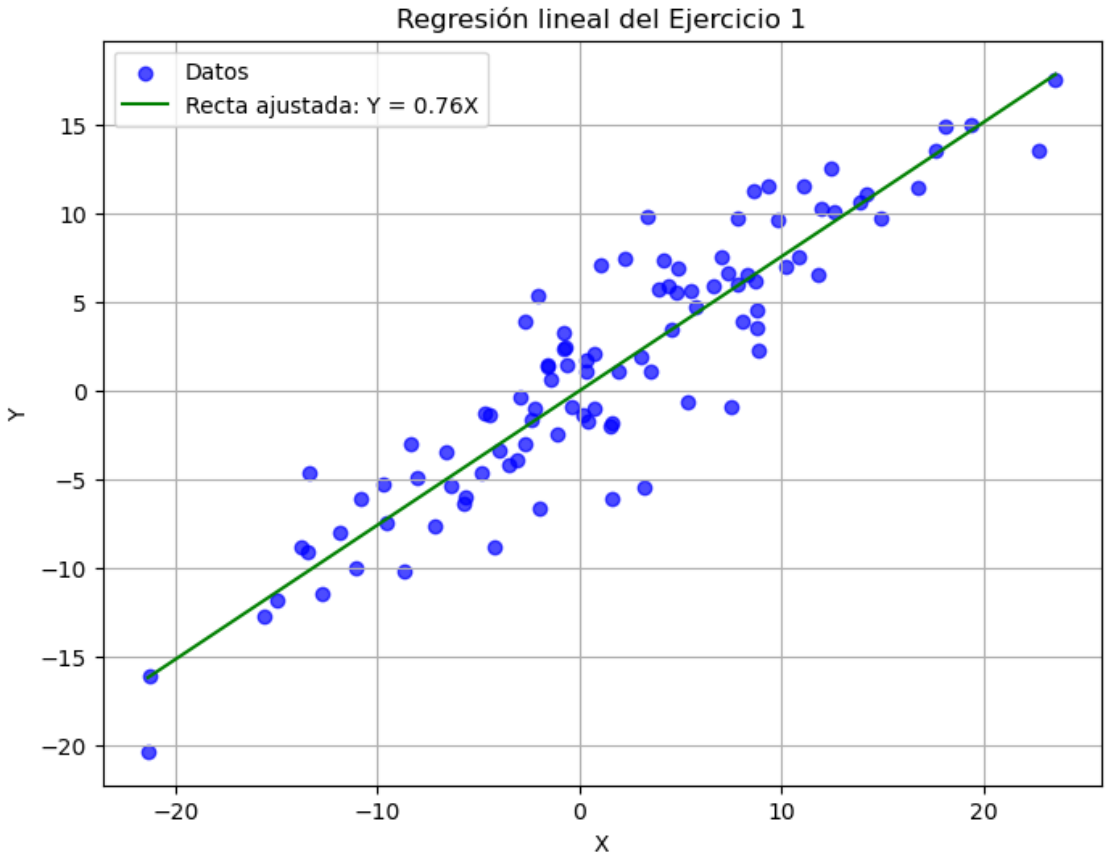
[24]:
```
ej1 = pd.read_csv('ejercicio_1.csv')
ej1.columns = ej1.columns.str.strip()
plt.figure(figsize=(8, 6))
plt.scatter(ej1['X'], ej1['Y'], color='blue', alpha=0.7)

plt.xlabel('X')
plt.ylabel('Y')
plt.title('Puntos en el plano XY del Ejercicio 1')
plt.grid(True)
plt.axis('equal')  # Para que X e Y usen la misma escala

plt.show()
```



Ejercicio 1, inciso B

[25]:
```
X = ej1["X"].values.reshape(-1, 1)
Y = ej1["Y"].values

# Calculamos beta con la formula que encontramos
XtX = X.T @ X
XtY = X.T @ Y
beta = np.linalg.inv(XtX) @ XtY

# Hacemos la recta
X_valores = np.linspace(ej1["X"].min(), ej1["X"].max(), 100)
Y_recta = beta[0] * X_valores

# Graficamos
plt.figure(figsize=(8, 6))
plt.scatter(ej1["X"], ej1["Y"], label="Datos", alpha=0.7, color="blue")
plt.plot(X_valores, Y_recta, color="green", label=f"Recta ajustada: Y =␣
↪{beta[0]:.2f}X")
plt.xlabel("X")
```

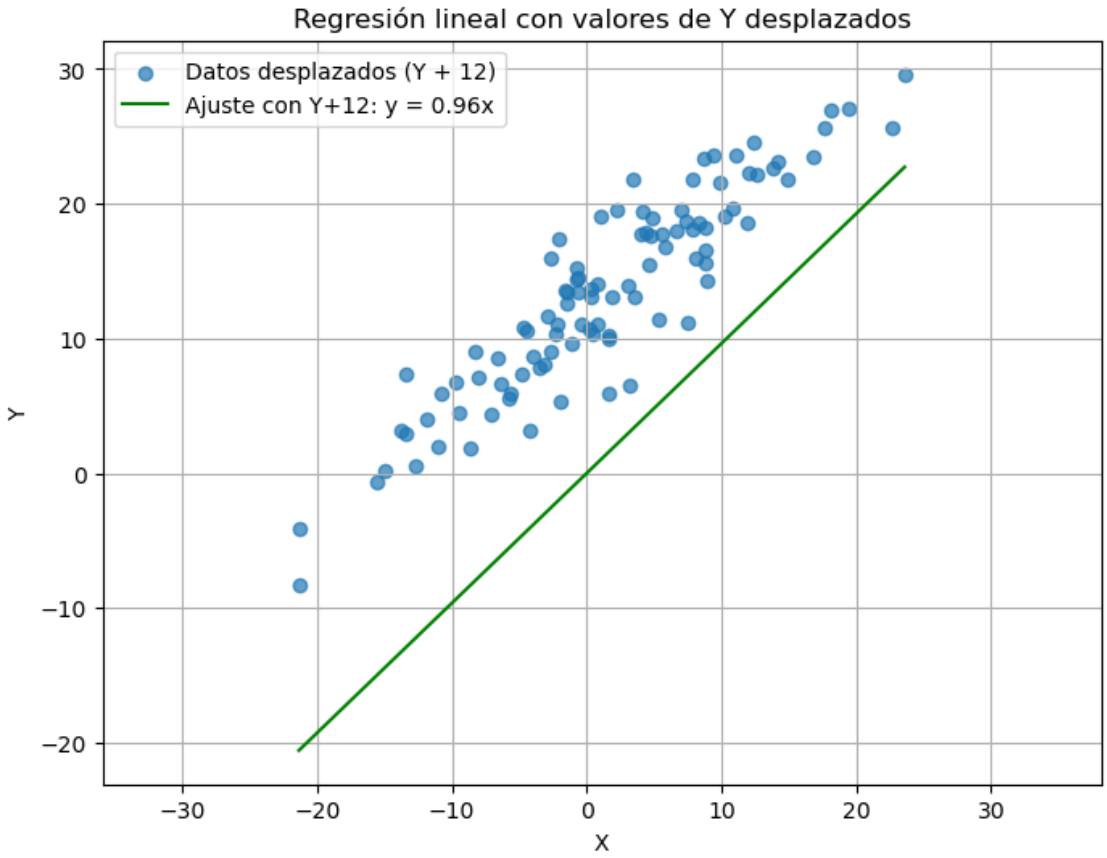
```
plt.ylabel("Y")
plt.title("Regresión lineal del Ejercicio 1")
plt.legend()
plt.grid(True)
plt.show()
```



Ejercicio 1, inciso C

[26]:
```
# Creamos una columna "Y" pero desplazada con el "+ 12"
ej1["Y_desplazado"] = ej1["Y"] + 12

X = ej1["X"].values.reshape(-1, 1)
Y2 = ej1["Y_desplazado"].values

XtY2 = X.T @ Y2
beta2 = np.linalg.inv(X.T @ X) @ XtY2

# Hacemos la nueva recta
X_valores = np.linspace(ej1["X"].min(), ej1["X"].max(), 100)
Y_recta2 = beta2[0] * X_valores
```

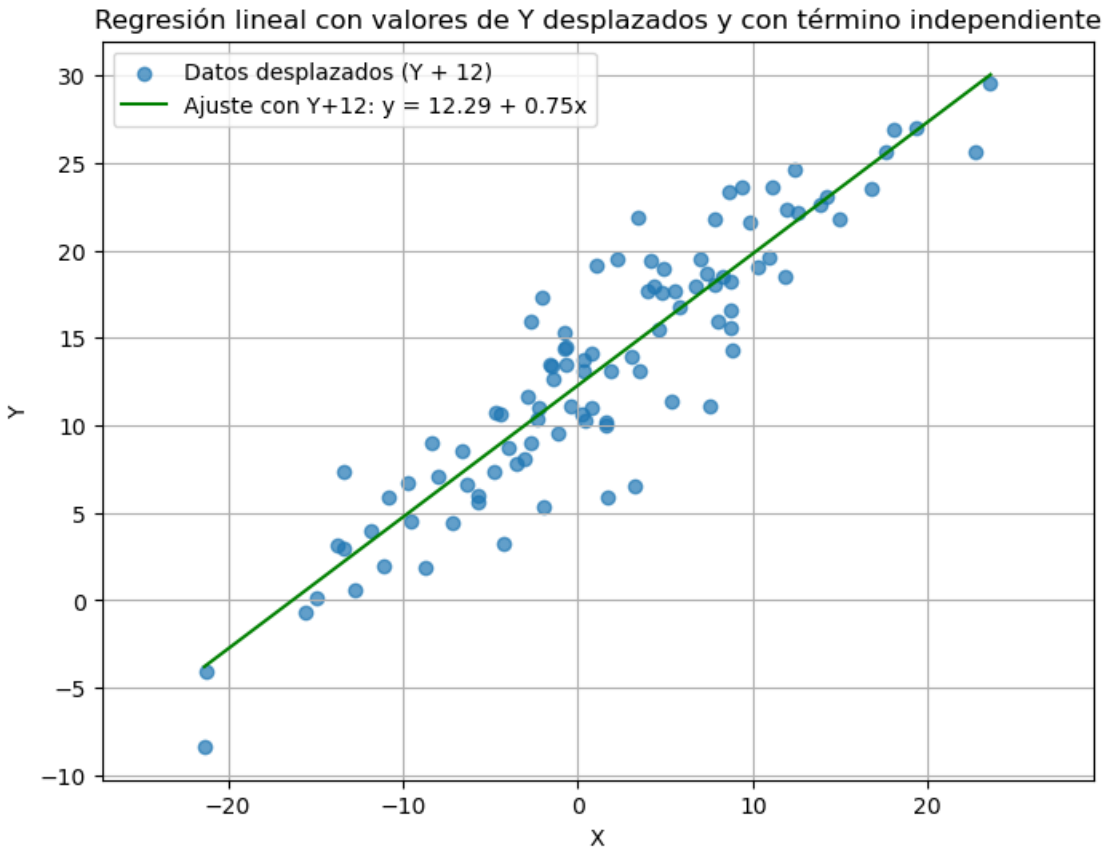
```
# Graficar
plt.figure(figsize=(8, 6))
plt.scatter(ej1["X"], ej1["Y_desplazado"], label="Datos desplazados (Y + 12)",␣
 ↪alpha=0.7)
plt.plot(X_valores, Y_recta2, color="green", label=f"Ajuste con Y+12: y =␣
 ↪{beta2[0]:.2f}x")
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Regresión lineal con valores de Y desplazados")
plt.legend()
plt.grid(True)
plt.axis('equal')
plt.show()
```



Agregamos columna de unos para arreglarlo

[27]:
```
# Creamos una columna "Y" pero desplazada con el "+ 12"
ej1["Y_desplazado"] = ej1["Y"] + 12
```

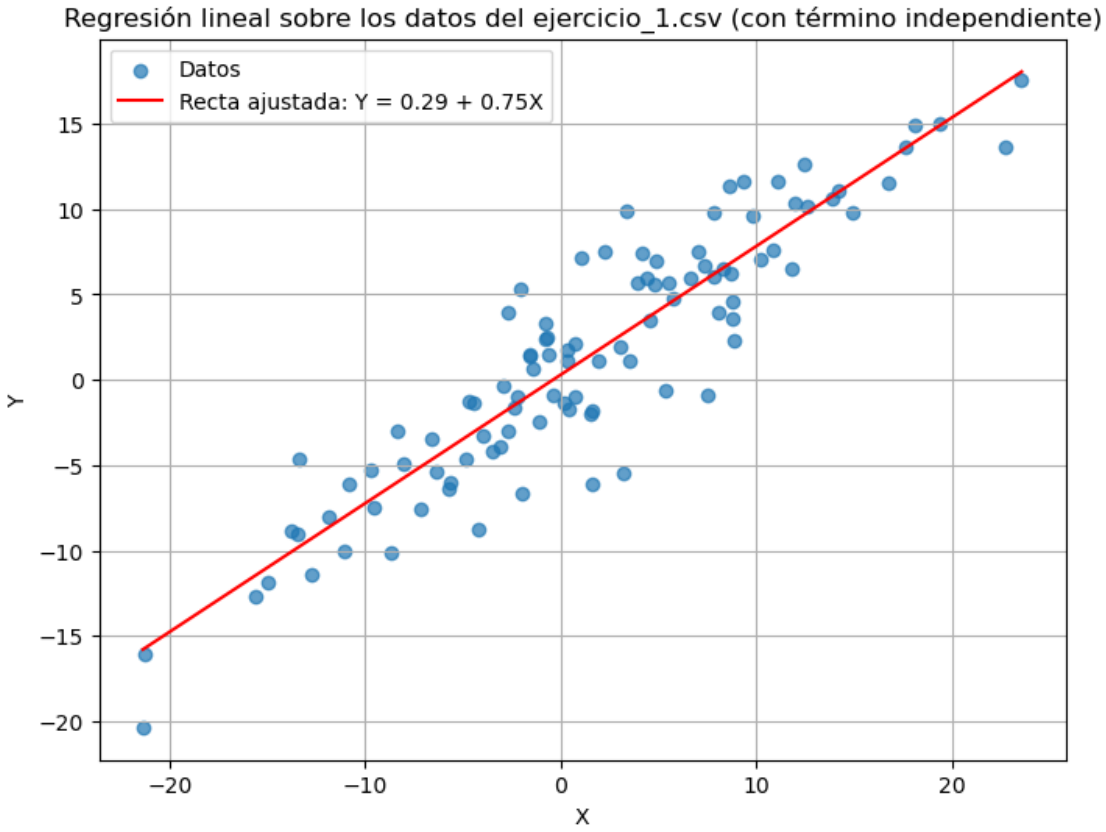
```
# Agregamos columna de unos a la matriz X
X = np.column_stack((np.ones(len(ej1)), ej1["X"].values))
Y2 = ej1["Y_desplazado"].values

XtY2 = X.T @ Y2
beta2 = np.linalg.inv(X.T @ X) @ XtY2

# Hacemos la nueva recta
X_valores = np.linspace(ej1["X"].min(), ej1["X"].max(), 100)
Y_recta2 = beta2[0] + beta2[1] * X_valores

# Graficar
plt.figure(figsize=(8, 6))
plt.scatter(ej1["X"], ej1["Y_desplazado"], label="Datos desplazados (Y + 12)",␣
 ↪alpha=0.7)
plt.plot(X_valores, Y_recta2, color="green", label=f"Ajuste con Y+12: y =␣
 ↪{beta2[0]:.2f} + {beta2[1]:.2f}x")
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Regresión lineal con valores de Y desplazados y con término␣
 ↪independiente")
plt.legend()
plt.grid(True)
plt.axis('equal')
plt.show()
```



Ejercicio 1, inciso b - pero con la columna de unos (no cambia nada)

[28]:
```
# Matriz X con columna de unos (modelo con término independiente, las columnas␣
↪de X son linealmente independientes)
X = np.column_stack((np.ones(len(ej1)), ej1["X"].values))
Y = ej1["Y"].values

# Calculamos beta con la formula que encontramos
XtX = X.T @ X
XtY = X.T @ Y
beta = np.linalg.inv(XtX) @ XtY

# Hacemos la recta
X_valores = np.linspace(ej1["X"].min(), ej1["X"].max(), 100)
Y_recta = beta[0] + beta[1] * X_valores

# Graficamos
plt.figure(figsize=(8, 6))
plt.scatter(ej1["X"], ej1["Y"], label="Datos", alpha=0.7)
```

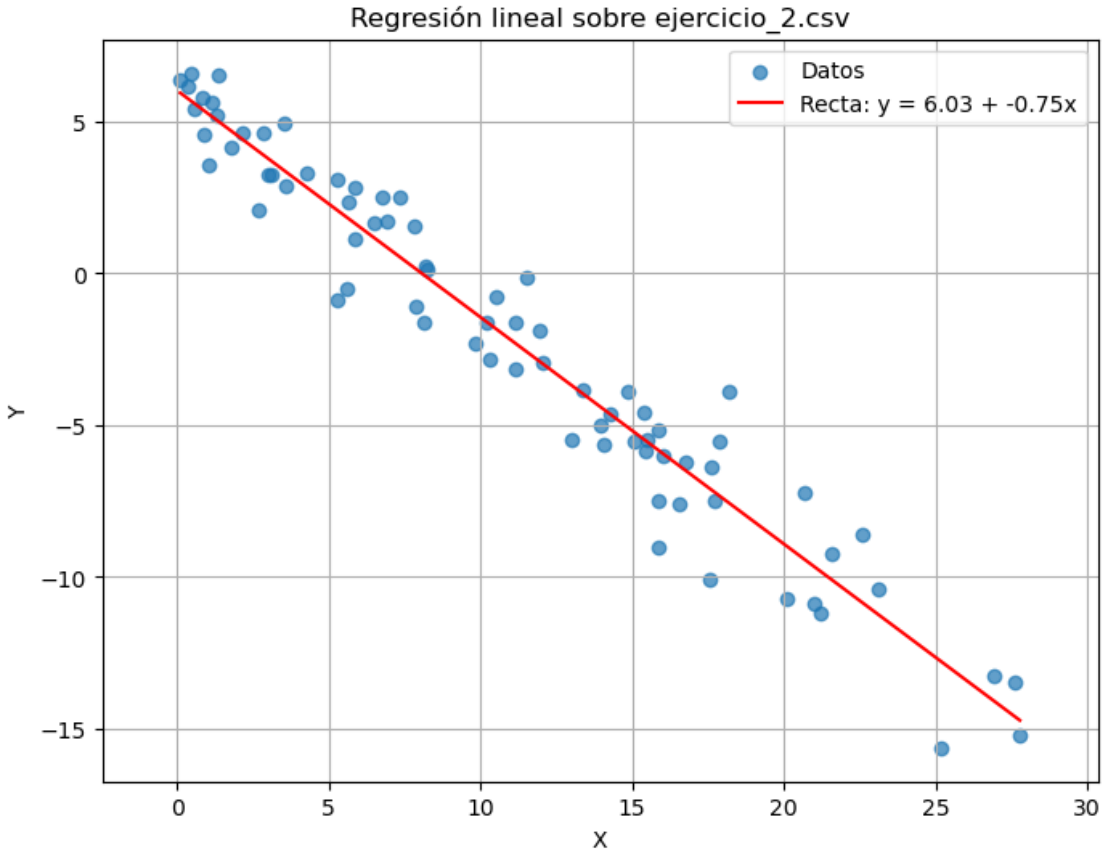
```
plt.plot(X_valores, Y_recta, color="red", label=f"Recta ajustada: Y = {beta[0]:.
↪2f} + {beta[1]:.2f}X")
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Regresión lineal sobre los datos del ejercicio_1.csv (con término␣
↪independiente)")
plt.legend()
plt.grid(True)
plt.show()
```



Ejercicio 2, inciso a

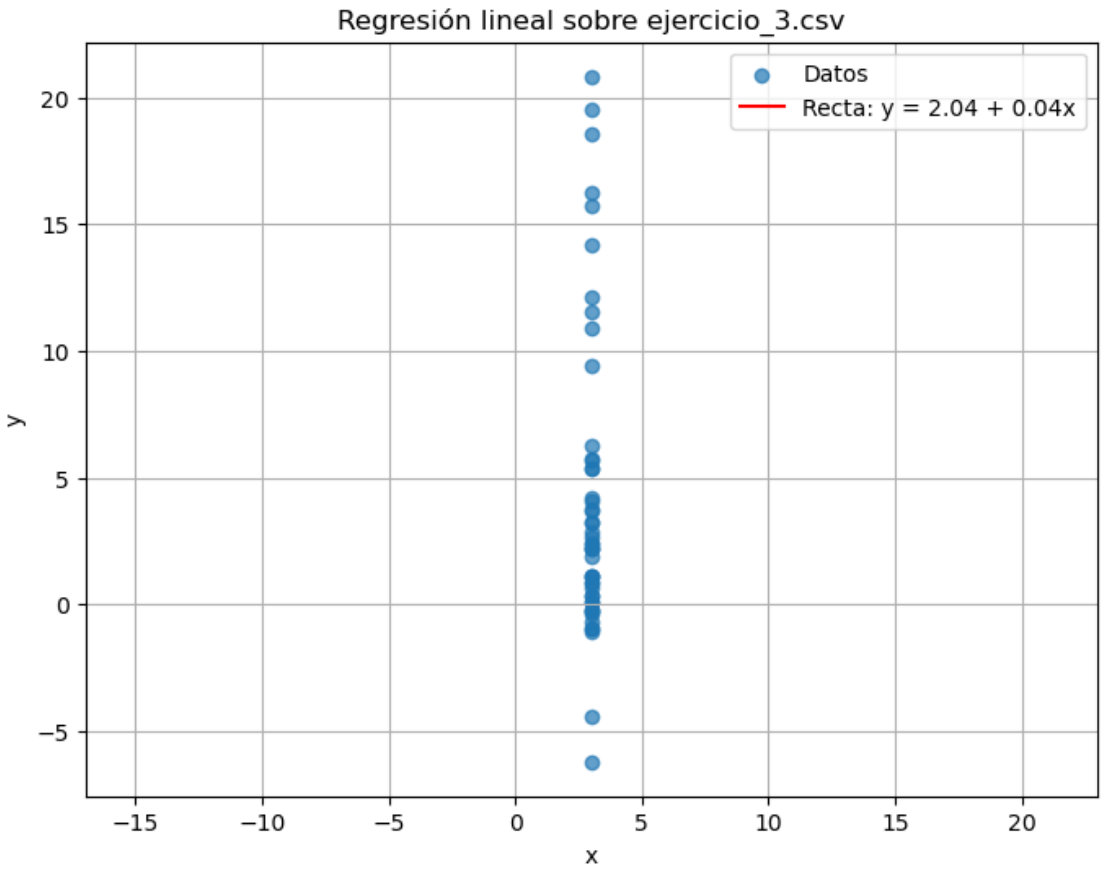
[29]:
```
ej2 = pd.read_csv("ejercicio_2.csv")
ej2.columns = ej2.columns.str.strip()

X = np.column_stack((np.ones(len(ej2)), ej2["X"].values))
Y = ej2["Y"].values

# Calculamos beta
beta = np.linalg.inv(X.T @ X) @ (X.T @ Y)
```

```python
# Recta
X_valores = np.linspace(ej2["X"].min(), ej2["X"].max(), 100)
Y_recta = beta[0] + beta[1] * X_valores

# Graficamos
plt.figure(figsize=(8, 6))
plt.scatter(ej2["X"], ej2["Y"], label="Datos", alpha=0.7)
plt.plot(X_valores, Y_recta, color="red", label=f"Recta: y = {beta[0]:.2f} +␣
↪{beta[1]:.2f}x")
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Regresión lineal sobre ejercicio_2.csv")
plt.legend()
plt.grid(True)
plt.axis("equal")
plt.show()
```



Ejercicio 3

[30]:
```
# Cargar los datos
ej3 = pd.read_csv("ejercicio_3.csv")

X = np.column_stack((np.ones(len(ej3)), ej3["x"].values))
Y = ej3["y"].values

# Calculamos beta
beta = np.linalg.inv(XtX) @ (X.T @ Y)

# Graficamos
X_valores = np.linspace(ej3["x"].min(), ej3["x"].max(), 100)
Y_recta = beta[0] + beta[1] * X_valores

plt.figure(figsize=(8, 6))
plt.scatter(ej3["x"], ej3["y"], label="Datos", alpha=0.7)
plt.plot(X_valores, Y_recta, color="red", label=f"Recta: y = {beta[0]:.2f} +␣
 ↪{beta[1]:.2f}x")
plt.xlabel("x")
plt.ylabel("y")
plt.title("Regresión lineal sobre ejercicio_3.csv")
plt.legend()
plt.grid(True)
plt.axis("equal")
plt.show()
```



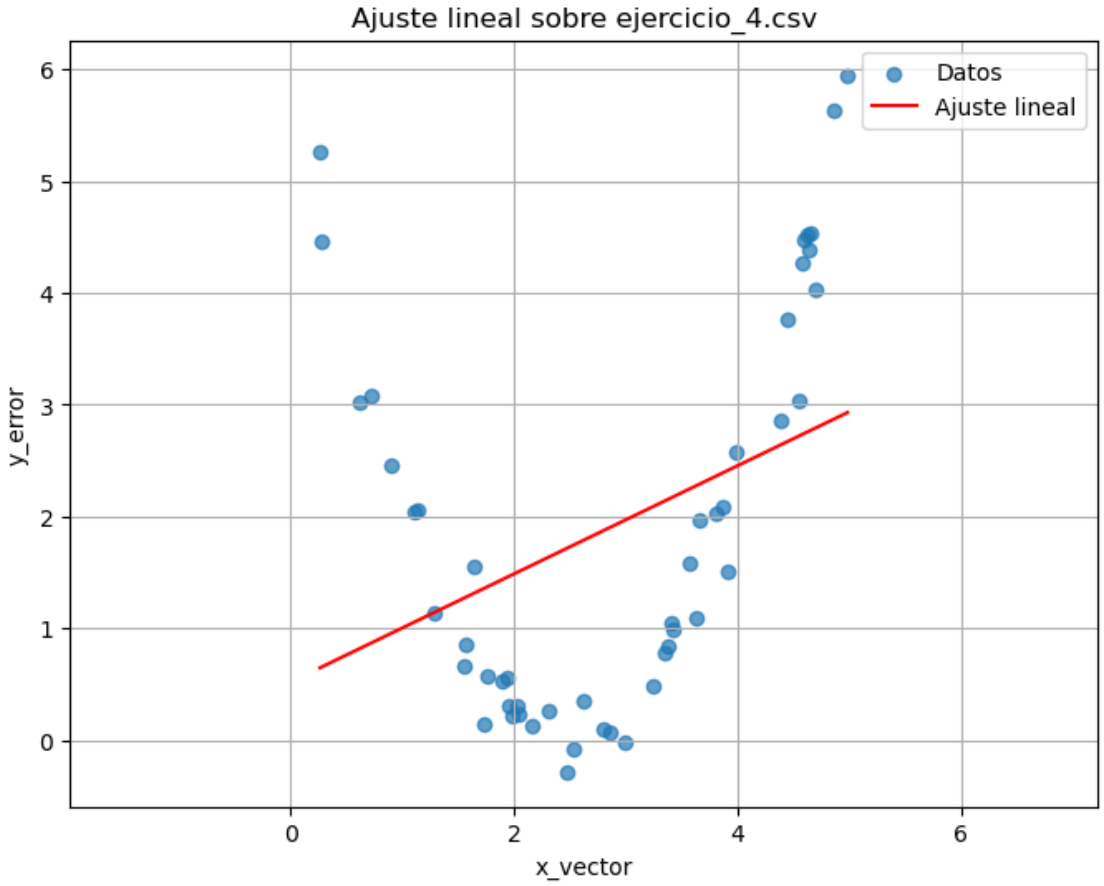
[31]:
```
# Calculamos el determinante X^t*X
XtX = X.T @ X
det_XtX = np.linalg.det(XtX)
print(f"Determinante de X^t X: {det_XtX:.4e}")
```

```
Determinante de X^t X: 4.8280e-07
```

Ejercicio 4, inciso a - Ajuste Lineal

[32]:
```
ej4 = pd.read_csv("ejercicio_4.csv")

X = np.column_stack((np.ones(len(ej4)), ej4["x_vector"].values))
Y = ej4["y_error"].values

beta_lin = np.linalg.inv(X.T @ X) @ (X.T @ Y)

X_valores = np.linspace(ej4["x_vector"].min(), ej4["x_vector"].max(), 200)
Y_recta = beta_lin[0] + beta_lin[1] * X_valores

# Graficamos
```

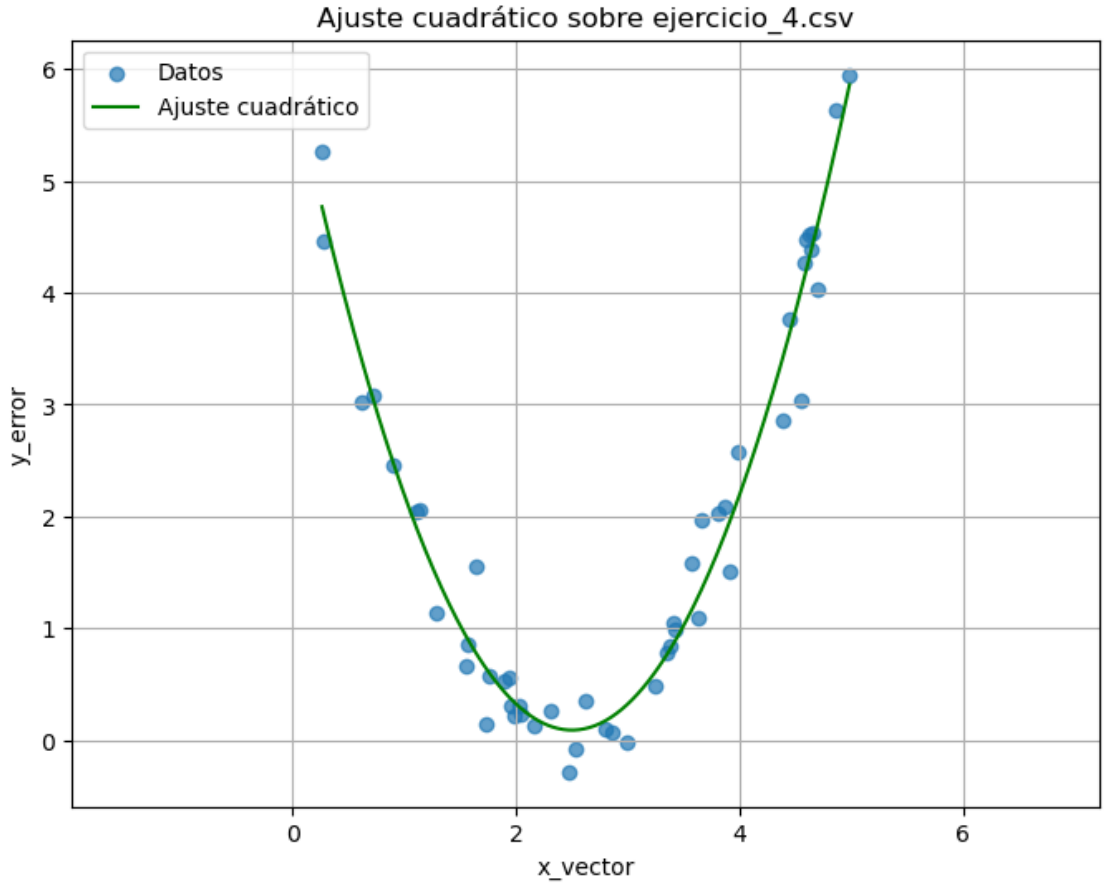
```
plt.figure(figsize=(8, 6))
plt.scatter(ej4["x_vector"], ej4["y_error"], label="Datos", alpha=0.7)
plt.plot(X_valores, Y_recta, color="red", label="Ajuste lineal")
plt.xlabel("x_vector")
plt.ylabel("y_error")
plt.title("Ajuste lineal sobre ejercicio_4.csv")
plt.legend()
plt.grid(True)
plt.axis("equal")
plt.show()
```



Paso 2: Ajuste Cuadratico

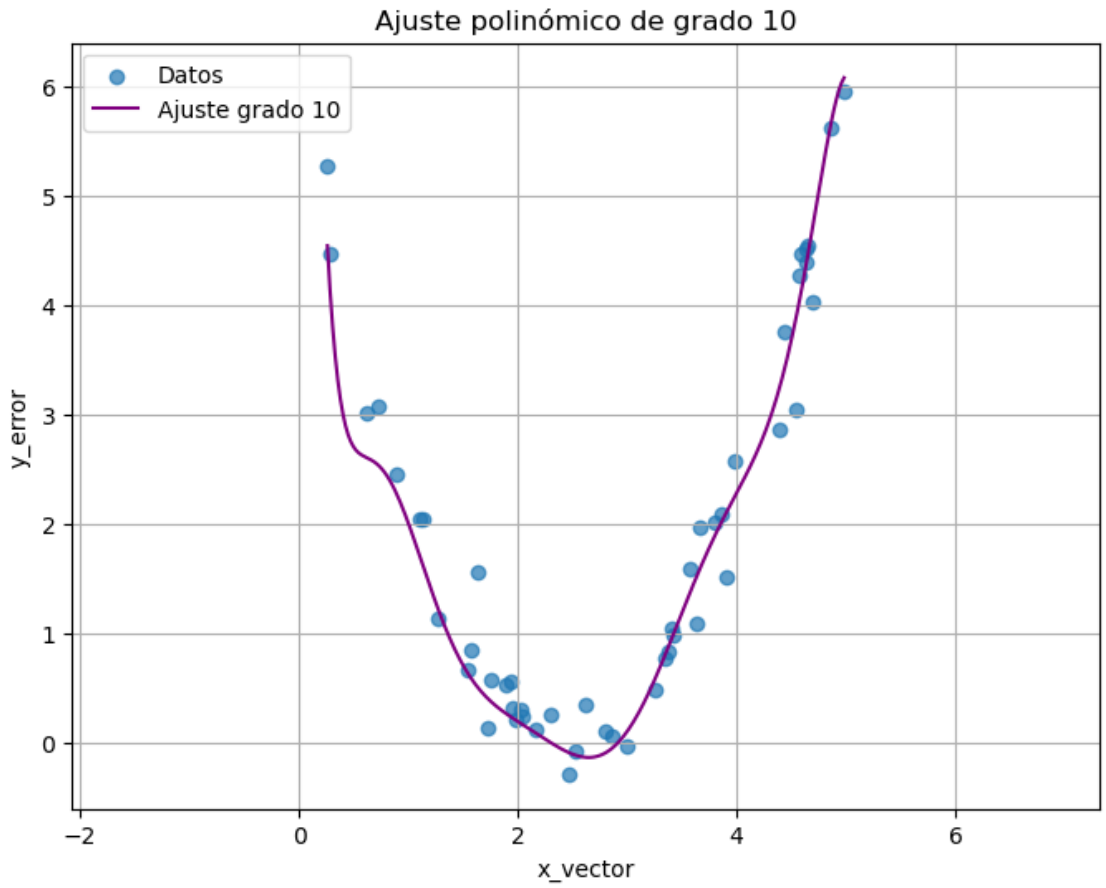
[33]:
```
# Matriz X con columna de unos, x_vector y x_vector^2
X_cuadratica = np.column_stack((
    np.ones(len(ej4)),
    ej4["x_vector"].values,
    ej4["x_vector"].values**2
))
```

```
Y = ej4["y_error"].values

# Calcular beta
beta_cuadratica = np.linalg.inv(X_cuadratica.T @ X_cuadratica) @ (X_cuadratica.
 ↪T @ Y)

# Evaluar predicción
Y_pred_quad = beta_cuadratica[0] + beta_cuadratica[1] * X_valores +␣
 ↪beta_cuadratica[2] * X_valores**2

# Graficar
plt.figure(figsize=(8, 6))
plt.scatter(ej4["x_vector"], ej4["y_error"], label="Datos", alpha=0.7)
plt.plot(X_valores, Y_pred_quad, color="green", label="Ajuste cuadrático")
plt.xlabel("x_vector")
plt.ylabel("y_error")
plt.title("Ajuste cuadrático sobre ejercicio_4.csv")
plt.legend()
plt.grid(True)
plt.axis("equal")
plt.show()
```



Paso 3: Polinomio de grado 10

[34]:
```
# Matriz con términos hasta x_vector^10
X_poli = np.column_stack([ej4["x_vector"].values**i for i in range(11)])
#X_poly = np.column_stack((np.ones(len(ej4)), X_poly))  # agregar columna de unos

Y = ej4["y_error"].values

# Calcular beta
beta_poli = np.linalg.inv(X_poli.T @ X_poli) @ (X_poli.T @ Y)

# Evaluar en X_vals
X_valores_poli = np.column_stack([X_valores**i for i in range(11)])
#X_vals_poly = np.column_stack((np.ones(len(X_vals)), X_vals_poly)) agregar columnas de unos?
Y_recta_poli = X_valores_poli @ beta_poli

# Graficar
plt.figure(figsize=(8, 6))
plt.scatter(ej4["x_vector"], ej4["y_error"], label="Datos", alpha=0.7)
plt.plot(X_valores, Y_recta_poli, color="purple", label="Ajuste grado 10")
plt.xlabel("x_vector")
plt.ylabel("y_error")
plt.title("Ajuste polinómico de grado 10")
plt.legend()
plt.grid(True)
plt.axis("equal")
plt.show()
```



Parte 3

[35]:
```
# Leer el archivo
df = pd.read_csv("student_performance.csv")

# Separar variables predictoras (X) y variable objetivo (Y)
X_columns = [
    "X1 Hours Studied",
    "X2 Previous Scores",
    "X3 Extracurricular Activities",
    "X4 Sleep Hours",
    "X5 Sample Question Papers Practiced"
]
Y_column = "Y Performance Index"

X = df[X_columns].values
Y = df[Y_column].values

# Dividir en entrenamiento (primeros 450) y test (últimos 150)
```

```
X_train = X[:450]
Y_train = Y[:450]

X_test = X[450:]
Y_test = Y[450:]
```

[36]:
```
# Agregar columna de unos para el término independiente
X_train_ones = np.column_stack((np.ones(len(X_train)), X_train))
X_test_ones = np.column_stack((np.ones(len(X_test)), X_test))

# Calcular beta
XtX = X_train_ones.T @ X_train_ones
XtY = X_train_ones.T @ Y_train
beta_hat = np.linalg.inv(XtX) @ XtY
```

[37]:
```
# Predicciones en conjunto de entrenamiento
Y_train_pred = X_train_ones @ beta_hat

# Error cuadrático medio
ECM_train = np.mean((Y_train - Y_train_pred)**2)
print(f"ECM (Datos de entrenamiento): {ECM_train:.4f}")
```

```
ECM (Datos de entrenamiento): 3.6009
```

[38]:
```
# Predicciones en test
Y_test_pred = X_test_ones @ beta_hat

# Error cuadrático medio en test
ECM_test = np.mean((Y_test - Y_test_pred)**2)
print(f"ECM (test con beta entrenado en 450 datos): {ECM_test:.4f}")
```

```
ECM (test con beta entrenado en 450 datos): 4.4364
```

[39]:
```
# Entrenar con todos los datos
X_all = np.column_stack((np.ones(len(X)), X))
beta_all = np.linalg.inv(X_all.T @ X_all) @ (X_all.T @ Y)

# Probar en el mismo conjunto de test
Y_test_pred_all = X_test_ones @ beta_all
ECM_test_all = np.mean((Y_test - Y_test_pred_all)**2)
print(f"ECM (test con beta entrenado en 600 datos): {ECM_test_all:.4f}")
```

```
ECM (test con beta entrenado en 600 datos): 4.3331
```

[40]:
```
errores_abs = np.abs(Y_test - Y_test_pred)
plt.figure(figsize=(10, 5))
plt.plot(errores_abs, marker="o", linestyle="-", alpha=0.7)
plt.xlabel("Estudiante (test)")
plt.ylabel("Error absoluto |y - ŷ|")
```

```
plt.title("Errores individuales en conjunto de test")
plt.grid(True)
plt.show()
```