Tarea8_LeninAmangandi

January 7, 2026

1 Escuela Politécnica Nacional

1.1 Métodos Numéricos

1.2 Tarea N8

Nombre: Lenin Amangandi

Tema: Ejercicios Unidad 03-C mínimos cuadrados

Enlace GitHub Tarea 8

1.3 Ejercicio 1

Dados los datos:

x_i	4.0	4.2	4.5	4.7	5.1	5.5	5.9	6.3	6.8	7.1
y_i	102.56	130.11	113.18	142.05	167.53	195.14	224.87	256.73	299.50	326.72

a. Construya el polinomio por mínimos cuadrados de grado 1 y calcule el error.

b. Construya el polinomio por mínimos cuadrados de grado 2 y calcule el error.

c. Construya el polinomio por mínimos cuadrados de grado 3 y calcule el error.

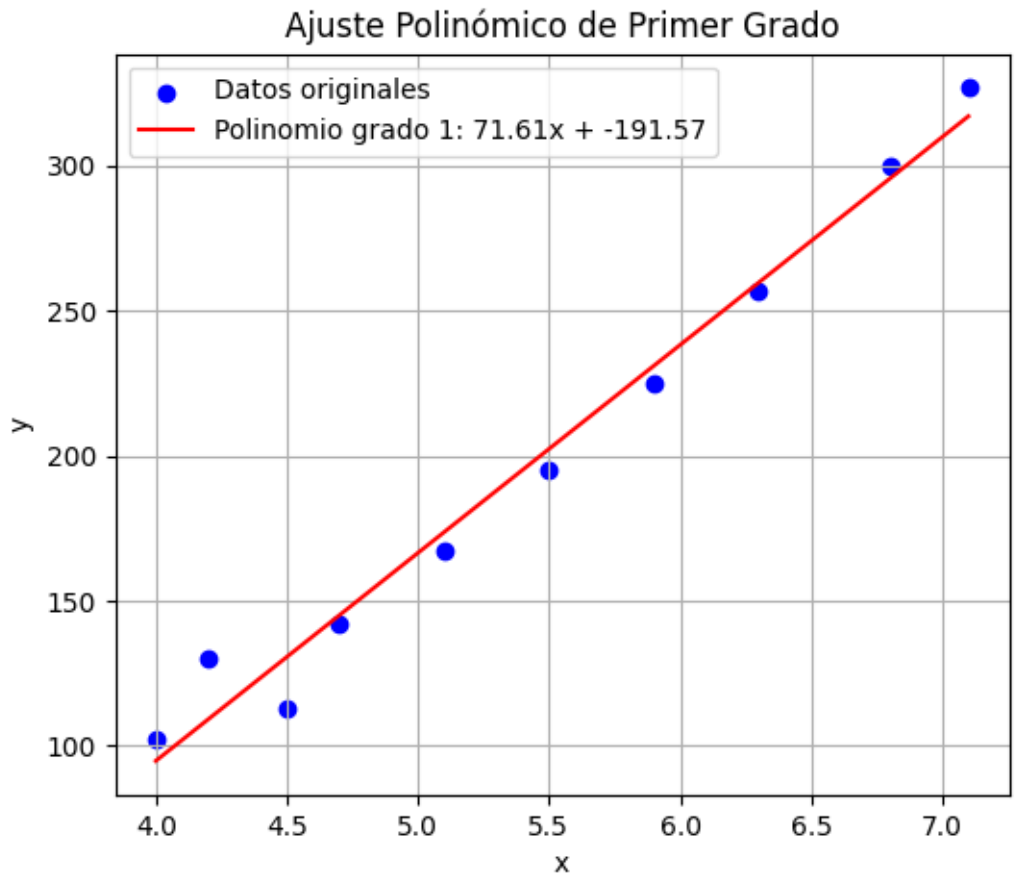
d. Construya el polinomio por mínimos cuadrados de la forma be^{ax} y calcule el error.

e. Construya el polinomio por mínimos cuadrados de la forma bx^a y calcule el error.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

x = np.array([4.0, 4.2, 4.5, 4.7, 5.1, 5.5, 5.9, 6.3, 6.8, 7.1])
y = np.array([102.56, 130.11, 113.18, 142.05, 167.53, 195.14, 224.87, 256.73,␣
 ↪299.50, 326.72])

def calcular_error(y_real, y_predicha):
    return np.mean((y_real - y_predicha)**2)
```

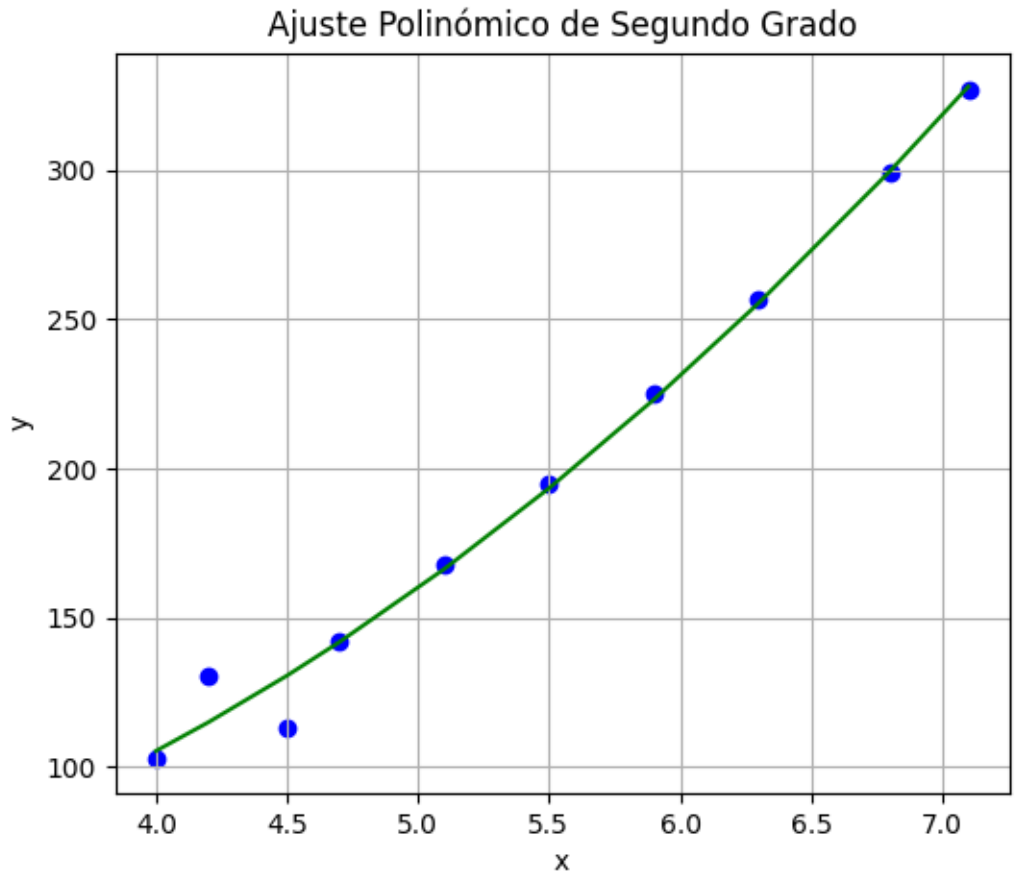
[2]:
```
coeffs_1 = np.polyfit(x, y, 1)
y_pred_1 = np.polyval(coeffs_1, x)
error_1 = calcular_error(y, y_pred_1)

print(f"\nPolinomio de grado 1:")
print(f"Coeficientes: {coeffs_1}")
print(f"Error (ECM): {error_1:.4f}")

plt.figure(figsize=(6, 5))
plt.scatter(x, y, color='blue', label='Datos originales')
plt.plot(x, y_pred_1, color='red', label=f'Polinomio grado 1: {coeffs_1[0]:.
  ↪2f}x + {coeffs_1[1]:.2f}')
plt.legend()
plt.xlabel('x')
plt.ylabel('y')
plt.title('Ajuste Polinómico de Primer Grado')
plt.grid(True)
plt.show()
```

```
Polinomio de grado 1:
Coeficientes: [  71.61024372 -191.57241853]
Error (ECM): 105.8839
```



[3]:
```
coeffs_2 = np.polyfit(x, y, 2)
y_pred_2 = np.polyval(coeffs_2, x)
error_2 = calcular_error(y, y_pred_2)

print(f"\nPolinomio de grado 2:")
print(f"Coeficientes: {coeffs_2}")
print(f"Error (ECM): {error_2:.4f}")

plt.figure(figsize=(6, 5))
plt.scatter(x, y, color='blue', label='Datos originales')
plt.plot(x, y_pred_2, color='green', label=f'Polinomio grado 2: {coeffs_2[0]:.
  ↪2f}x² + {coeffs_2[1]:.2f}x + {coeffs_2[2]:.2f}')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Ajuste Polinómico de Segundo Grado')
plt.grid(True)
plt.show()
```

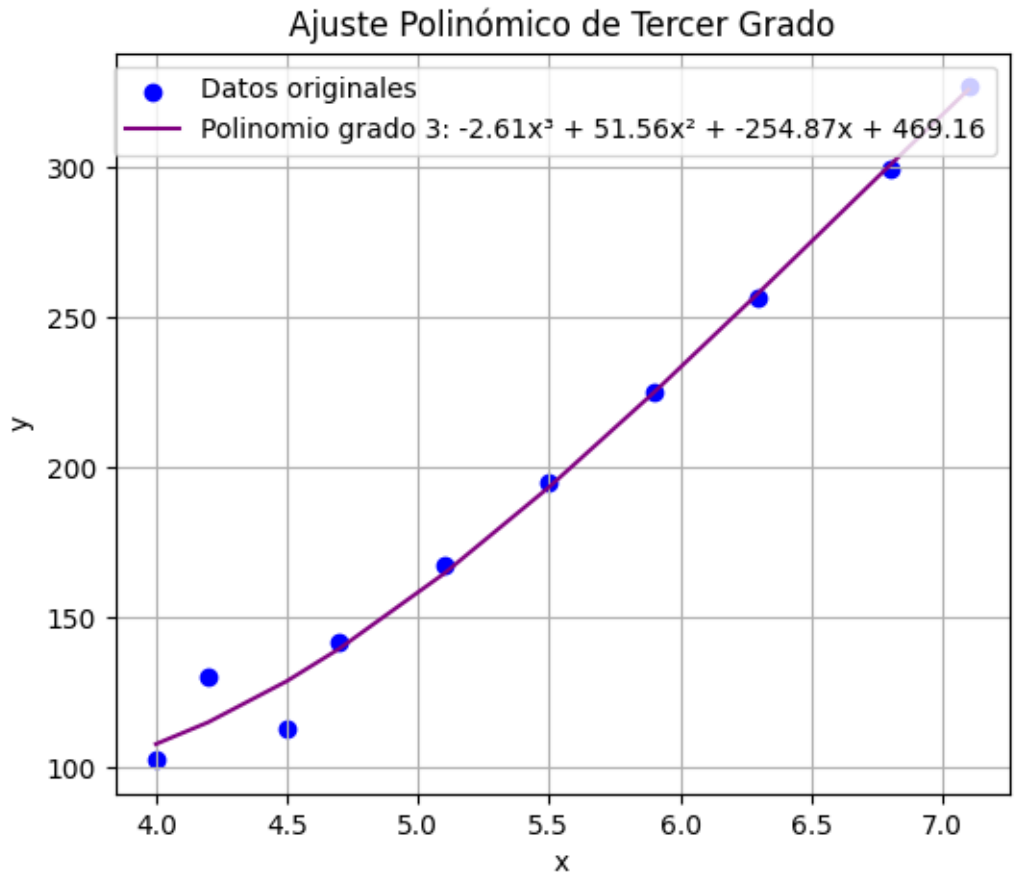
```
Polinomio de grado 2:
Coeficientes: [  8.21707232 -19.30860379  51.00078939]
Error (ECM): 55.1656
```



[4]:
```
coeffs_3 = np.polyfit(x, y, 3)
y_pred_3 = np.polyval(coeffs_3, x)
error_3 = calcular_error(y, y_pred_3)

print(f"\nPolinomio de grado 3:")
print(f"Coeficientes: {coeffs_3}")
print(f"Error (ECM): {error_3:.4f}")

plt.figure(figsize=(6, 5))
plt.scatter(x, y, color='blue', label='Datos originales')
plt.plot(x, y_pred_3, color='purple', label=f'Polinomio grado 3: {coeffs_3[0]:.
  ↪2f}x³ + {coeffs_3[1]:.2f}x² + {coeffs_3[2]:.2f}x + {coeffs_3[3]:.2f}')
plt.legend()
plt.xlabel('x')
plt.ylabel('y')
```

```
plt.title('Ajuste Polinómico de Tercer Grado')
plt.grid(True)
plt.show()
```

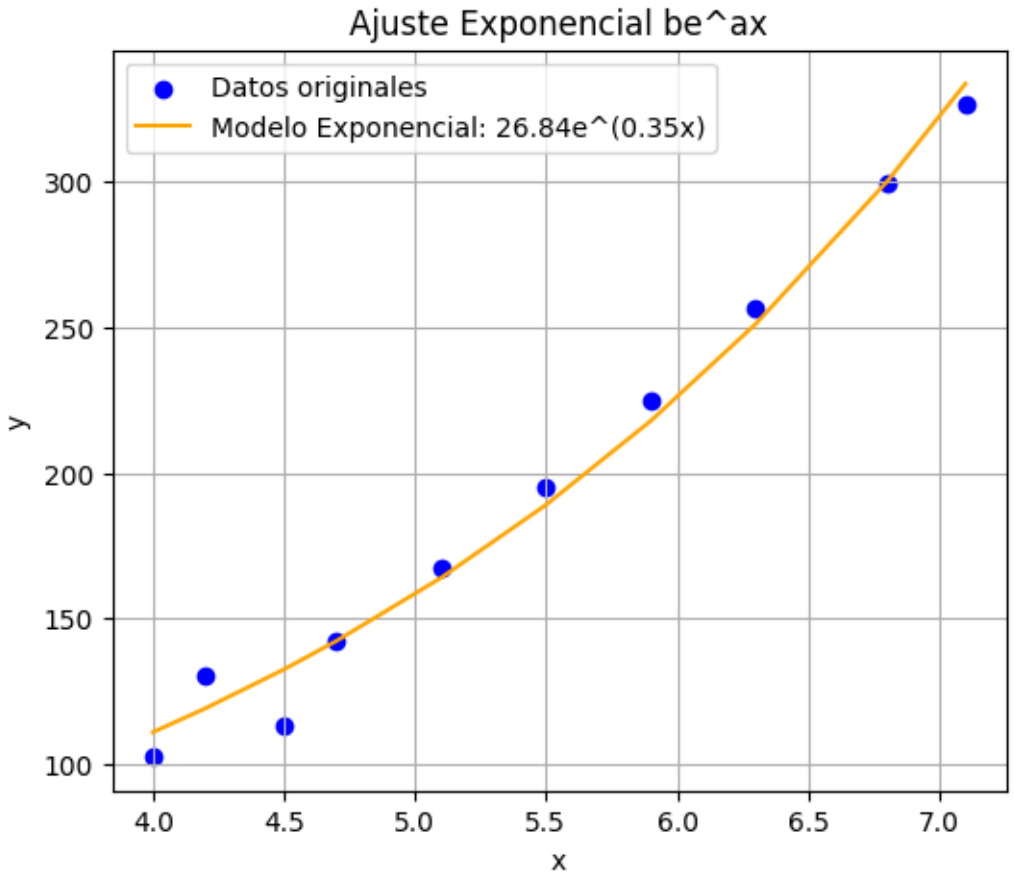
```
Polinomio de grado 3:
Coeficientes: [  -2.60683872   51.56095694 -254.87478338  469.16326528]
Error (ECM): 51.8383
```



[5]:
```
def exp_func(x, b, a):
    return b * np.exp(a * x)

params_exp, _ = curve_fit(exp_func, x, y, p0=[1, 0.1])

y_pred_exp = exp_func(x, *params_exp)
error_exp = calcular_error(y, y_pred_exp)

print(f"\nModelo Exponencial:")
print(f"a = {params_exp[1]:.4f}, b = {params_exp[0]:.4f}")
```

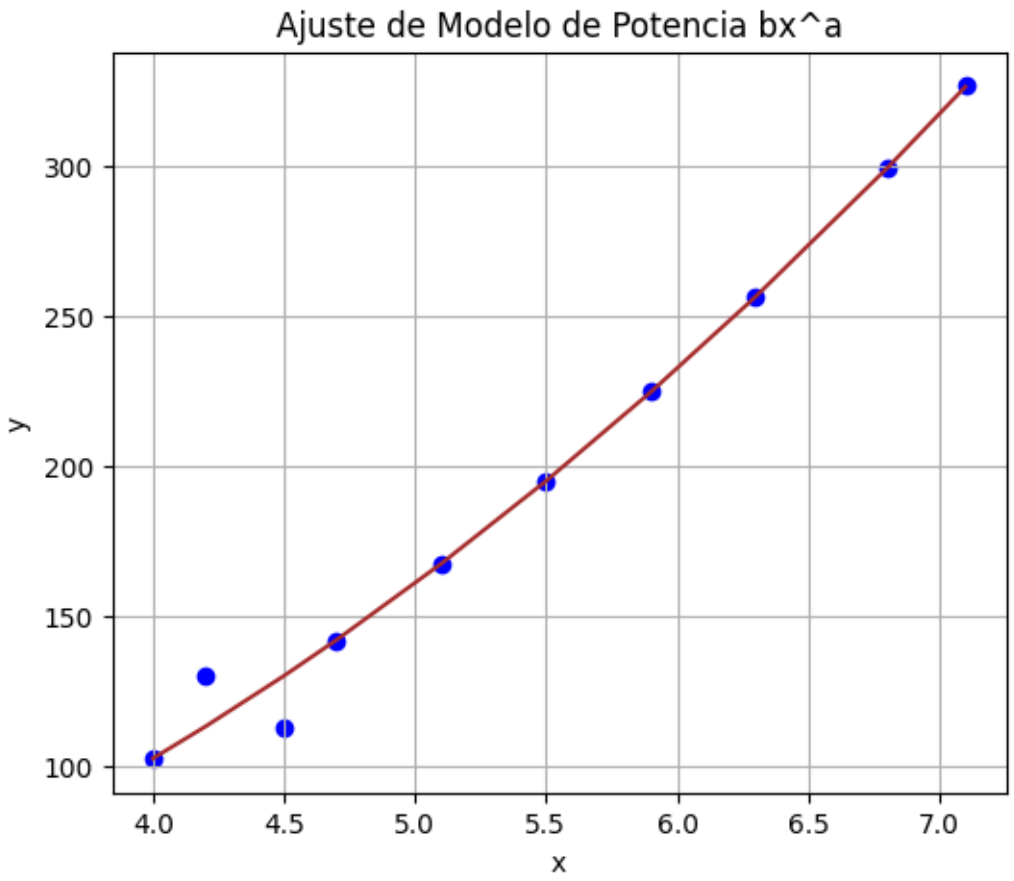
```
print(f"Error (ECM): {error_exp:.4f}")

plt.figure(figsize=(6, 5))
plt.scatter(x, y, color='blue', label='Datos originales')
plt.plot(x, y_pred_exp, color='orange', label=f'Modelo Exponencial:␣
 ↪{params_exp[0]:.2f}e^({params_exp[1]:.2f}x)')
plt.legend()
plt.xlabel('x')
plt.ylabel('y')
plt.title('Ajuste Exponencial be^ax')
plt.grid(True)
plt.show()
```

```
Modelo Exponencial:
a = 0.3549, b = 26.8408
Error (ECM): 74.3622
```



[6]:
```
def power_func(x, b, a):
    return b * x**a

params_power, _ = curve_fit(power_func, x, y, p0=[1, 1])

y_pred_power = power_func(x, *params_power)
error_power = calcular_error(y, y_pred_power)

print(f"\nModelo Potencia:")
print(f"a = {params_power[1]:.4f}, b = {params_power[0]:.4f}")
print(f"Error (ECM): {error_power:.4f}")

plt.figure(figsize=(6, 5))
plt.scatter(x, y, color='blue', label='Datos originales')
plt.plot(x, y_pred_power, color='brown', label=f'Modelo Potencia:␣
 ↪{params_power[0]:.2f}x^{params_power[1]:.2f}')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Ajuste de Modelo de Potencia bx^a')
plt.grid(True)
plt.show()
```

```
Modelo Potencia:
a = 2.0152, b = 6.2840
Error (ECM): 57.2582
```



1.4 Ejercicio 2

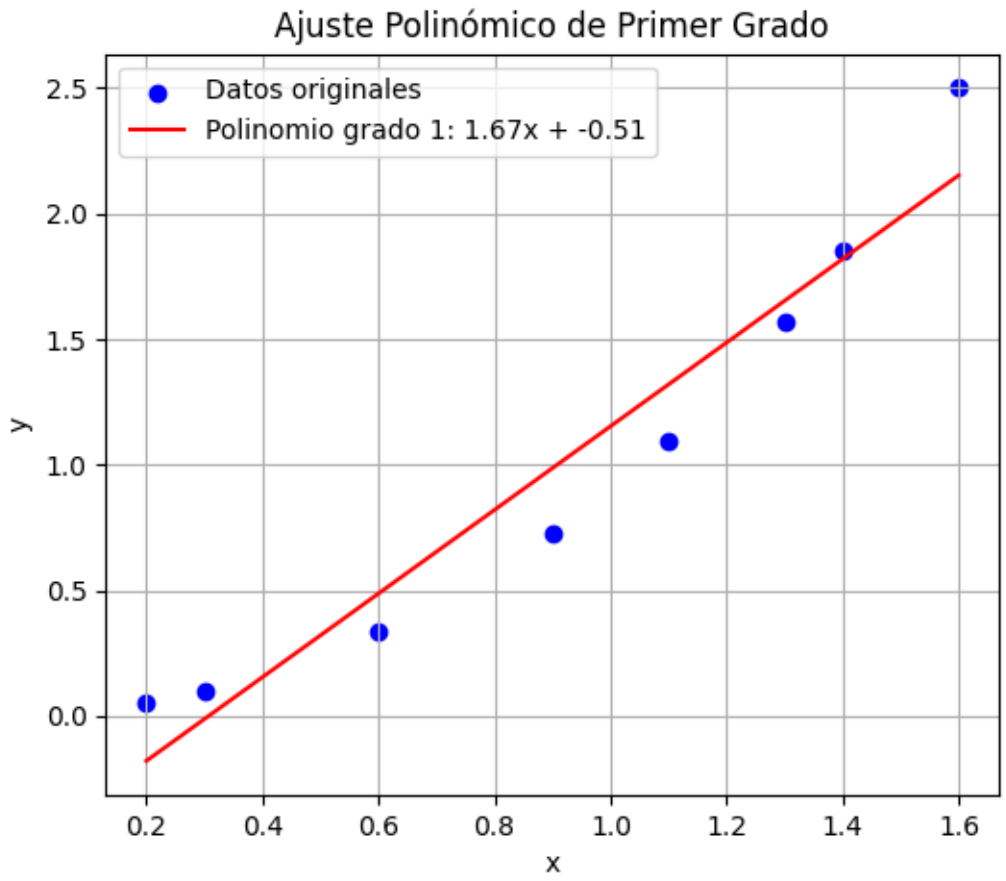
Repita el ejercicio 5 para los siguientes datos.

x_i	0.2	0.3	0.6	0.9	1.1	1.3	1.4	1.6
y_i	0.050446	0.098426	0.33277	0.72660	1.0972	1.5697	1.8487	2.5015

[7]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

x = np.array([0.2, 0.3, 0.6, 0.9, 1.1, 1.3, 1.4, 1.6])
y = np.array([0.050446, 0.098426, 0.33277, 0.72660, 1.0972, 1.5697, 1.8487, 2.
 ↪5015])

def calcular_error(y_real, y_predicha):
    return np.mean((y_real - y_predicha)**2)
```

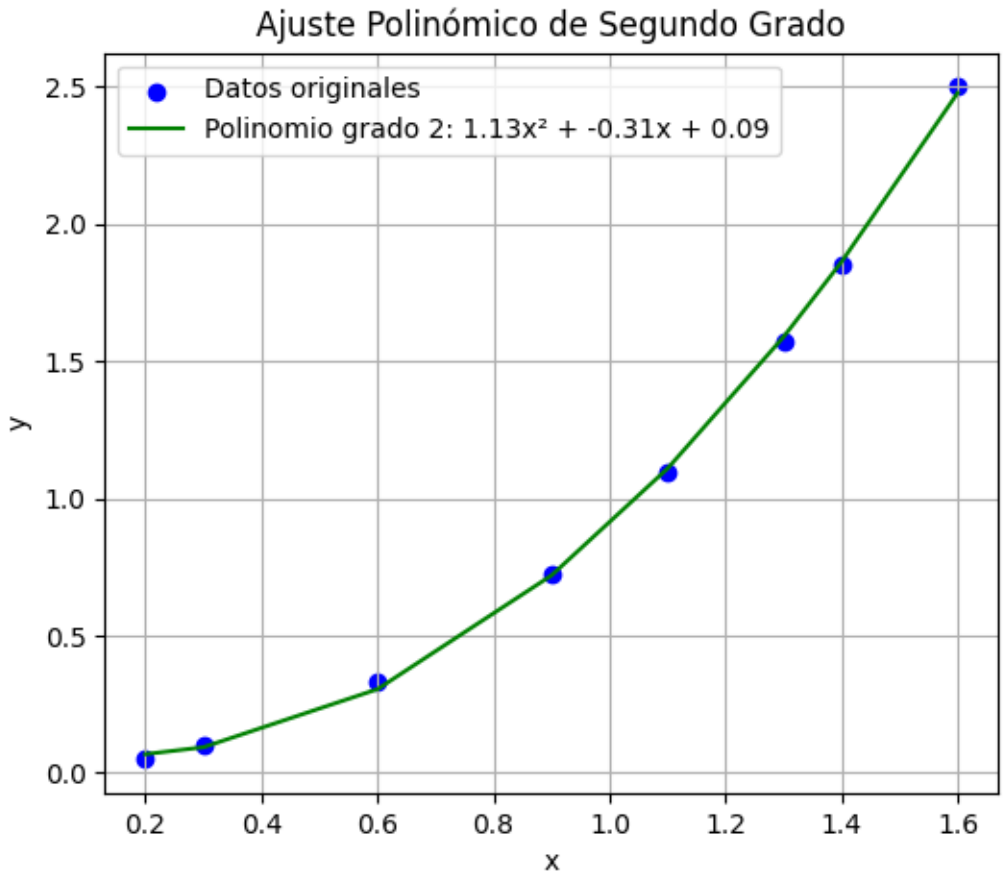
[8]:
```
coeffs_1 = np.polyfit(x, y, 1)
y_pred_1 = np.polyval(coeffs_1, x)
error_1 = calcular_error(y, y_pred_1)

print(f"\nPolinomio de grado 1:")
print(f"Coeficientes: {coeffs_1}")
print(f"Error (ECM): {error_1:.4f}")

plt.figure(figsize=(6, 5))
plt.scatter(x, y, color='blue', label='Datos originales')
plt.plot(x, y_pred_1, color='red', label=f'Polinomio grado 1: {coeffs_1[0]:.
  ↪2f}x + {coeffs_1[1]:.2f}')
plt.legend()
plt.xlabel('x')
plt.ylabel('y')
plt.title('Ajuste Polinómico de Primer Grado')
plt.grid(True)
plt.show()
```

```
Polinomio de grado 1:
Coeficientes: [ 1.66554008 -0.51245682]
Error (ECM): 0.0419
```



[9]:
```
coeffs_2 = np.polyfit(x, y, 2)
y_pred_2 = np.polyval(coeffs_2, x)
error_2 = calcular_error(y, y_pred_2)

print(f"\nPolinomio de grado 2:")
print(f"Coeficientes: {coeffs_2}")
print(f"Error (ECM): {error_2:.4f}")

plt.figure(figsize=(6, 5))
plt.scatter(x, y, color='blue', label='Datos originales')
plt.plot(x, y_pred_2, color='green', label=f'Polinomio grado 2: {coeffs_2[0]:.
  ↪2f}x² + {coeffs_2[1]:.2f}x + {coeffs_2[2]:.2f}')
plt.legend()
plt.xlabel('x')
plt.ylabel('y')
plt.title('Ajuste Polinómico de Segundo Grado')
plt.grid(True)
plt.show()
```

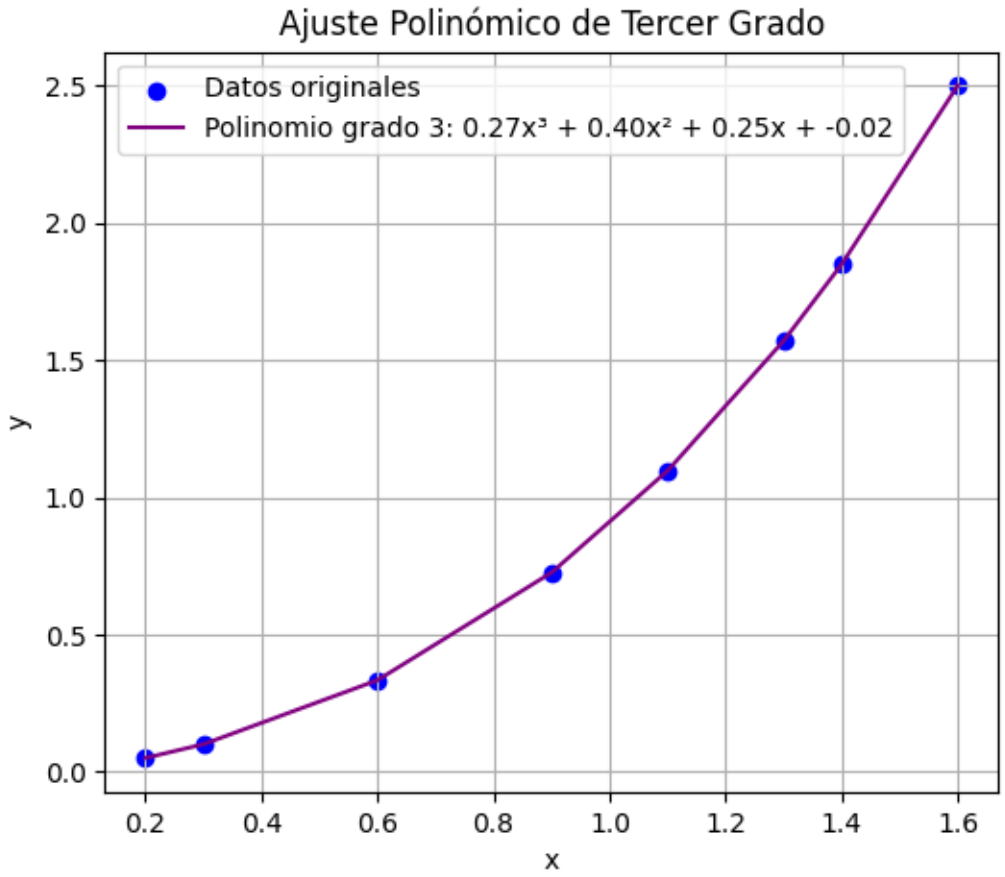
```
Polinomio de grado 2:
Coeficientes: [ 1.12942387 -0.31140346  0.08514393]
Error (ECM): 0.0003
```



[10]:
```
coeffs_3 = np.polyfit(x, y, 3)
y_pred_3 = np.polyval(coeffs_3, x)
error_3 = calcular_error(y, y_pred_3)

print(f"\nPolinomio de grado 3:")
print(f"Coeficientes: {coeffs_3}")
print(f"Error (ECM): {error_3:.4f}")

plt.figure(figsize=(6, 5))
plt.scatter(x, y, color='blue', label='Datos originales')
plt.plot(x, y_pred_3, color='purple', label=f'Polinomio grado 3: {coeffs_3[0]:.
  ↪2f}x³ + {coeffs_3[1]:.2f}x² + {coeffs_3[2]:.2f}x + {coeffs_3[3]:.2f}')
plt.legend()
plt.xlabel('x')
```

```
plt.ylabel('y')
plt.title('Ajuste Polinómico de Tercer Grado')
plt.grid(True)
plt.show()
```

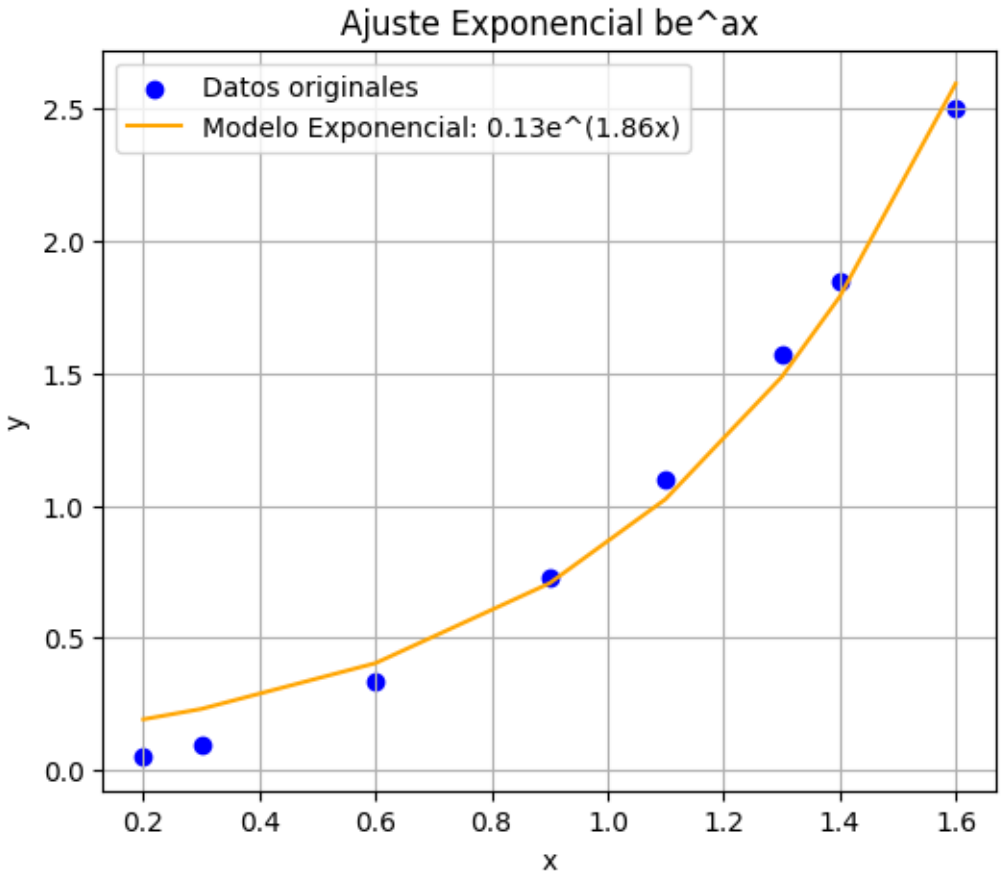
```
Polinomio de grado 3:
Coeficientes: [ 0.2662081   0.40293221  0.24838578 -0.0184014 ]
Error (ECM): 0.0000
```



[11]:
```
def exp_func(x, b, a):
    return b * np.exp(a * x)

params_exp, _ = curve_fit(exp_func, x, y, p0=[1, 0.1])
y_pred_exp = exp_func(x, *params_exp)
error_exp = calcular_error(y, y_pred_exp)

print(f"\nModelo Exponencial:")
print(f"a = {params_exp[1]:.4f}, b = {params_exp[0]:.4f}")
```

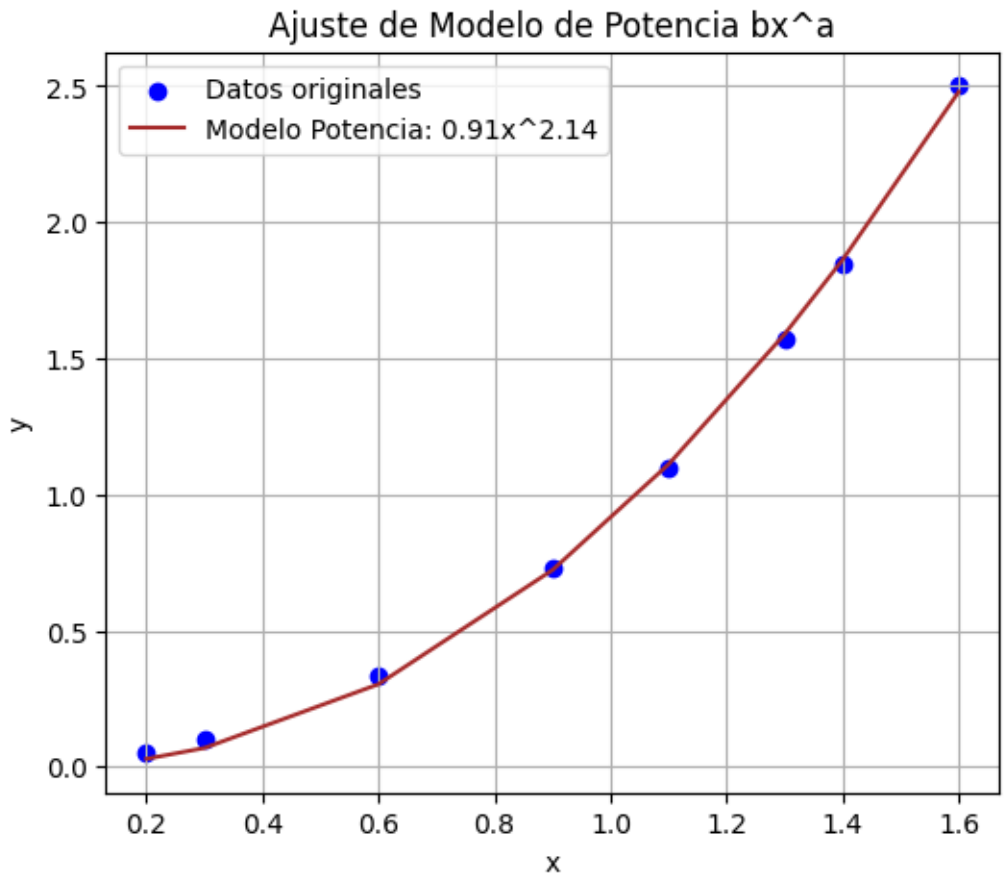
```
print(f"Error (ECM): {error_exp:.4f}")

plt.figure(figsize=(6, 5))
plt.scatter(x, y, color='blue', label='Datos originales')
plt.plot(x, y_pred_exp, color='orange', label=f'Modelo Exponencial:␣
 ↪{params_exp[0]:.2f}e^({params_exp[1]:.2f}x)')
plt.legend()
plt.xlabel('x')
plt.ylabel('y')
plt.title('Ajuste Exponencial be^ax')
plt.grid(True)
plt.show()
```

```
Modelo Exponencial:
a = 1.8583, b = 0.1326
Error (ECM): 0.0085
```



[12]:
```
def power_func(x, b, a):
    return b * x**a

params_power, _ = curve_fit(power_func, x, y, p0=[1, 1])
y_pred_power = power_func(x, *params_power)
error_power = calcular_error(y, y_pred_power)

print(f"\nModelo Potencia:")
print(f"a = {params_power[1]:.4f}, b = {params_power[0]:.4f}")
print(f"Error (ECM): {error_power:.4f}")

plt.figure(figsize=(6, 5))
plt.scatter(x, y, color='blue', label='Datos originales')
plt.plot(x, y_pred_power, color='brown', label=f'Modelo Potencia:␣
 ↪{params_power[0]:.2f}x^{params_power[1]:.2f}')
plt.legend()
plt.xlabel('x')
plt.ylabel('y')
plt.title('Ajuste de Modelo de Potencia bx^a')
plt.grid(True)
plt.show()
```

```
Modelo Potencia:
a = 2.1428, b = 0.9055
Error (ECM): 0.0004
```



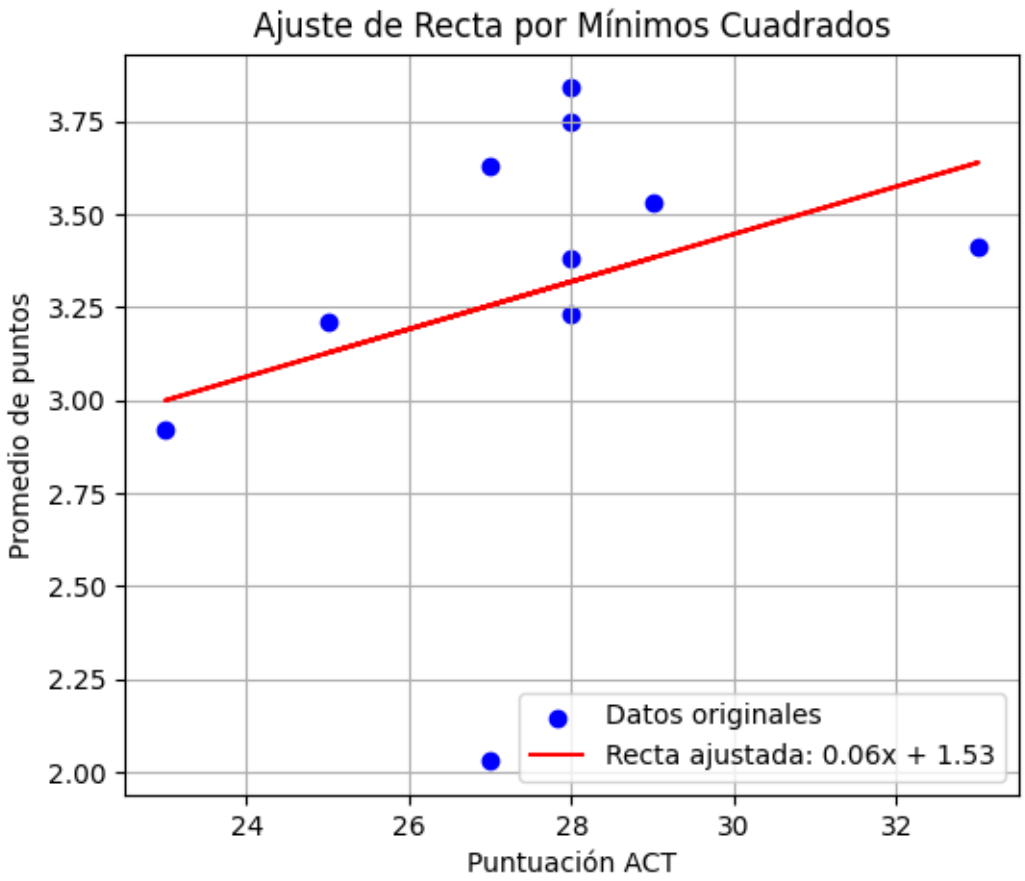
1.5 Ejercicio 3

La siguiente tabla muestra los promedios de puntos del colegio de 20 especialistas en matemáticas y ciencias computacionales, junto con las calificaciones que recibieron estos estudiantes en la parte de matemáticas de la prueba ACT (Programa de Pruebas de Colegios Americanos) mientras estaban en secundaria. Grafique estos datos y encuentre la ecuación de la recta por mínimos cuadrados para estos datos.

Puntuación ACT	Promedio de puntos	Puntuación ACT	Promedio de puntos
28	3.84	29	3.75
25	3.21	28	3.65
28	3.23	27	3.87
27	3.63	29	3.75
28	3.75	21	1.66
33	3.20	28	3.12
28	3.41	28	2.96
29	3.38	26	2.92
23	3.53	30	3.10
27	2.03	24	2.81

[13]:
```python
import numpy as np
import matplotlib.pyplot as plt

x = np.array([28, 25, 28, 27, 28, 33, 28, 29, 23, 27])
y = np.array([3.84, 3.21, 3.23, 3.63, 3.75, 3.41, 3.38, 3.53, 2.92, 2.03])

coeffs = np.polyfit(x, y, 1)
y_pred = np.polyval(coeffs, x)
error = np.mean((y - y_pred)**2)

print("--- RESULTADOS ---")
print(f"Coeficientes: {coeffs}")
print(f"Error (ECM): {error:.4f}")

plt.figure(figsize=(6, 5))
plt.scatter(x, y, color='blue', label='Datos originales')
plt.plot(x, y_pred, color='red', label=f'Recta ajustada: {coeffs[0]:.2f}x +␣
 ↪{coeffs[1]:.2f}')
plt.legend()
plt.xlabel('Puntuación ACT')
plt.ylabel('Promedio de puntos')
plt.title('Ajuste de Recta por Mínimos Cuadrados')
plt.grid(True)
plt.show()
```

```
--- RESULTADOS ---
Coeficientes: [0.0639404  1.52824503]
Error (ECM): 0.2197
```



1.6 Ejercicio 4

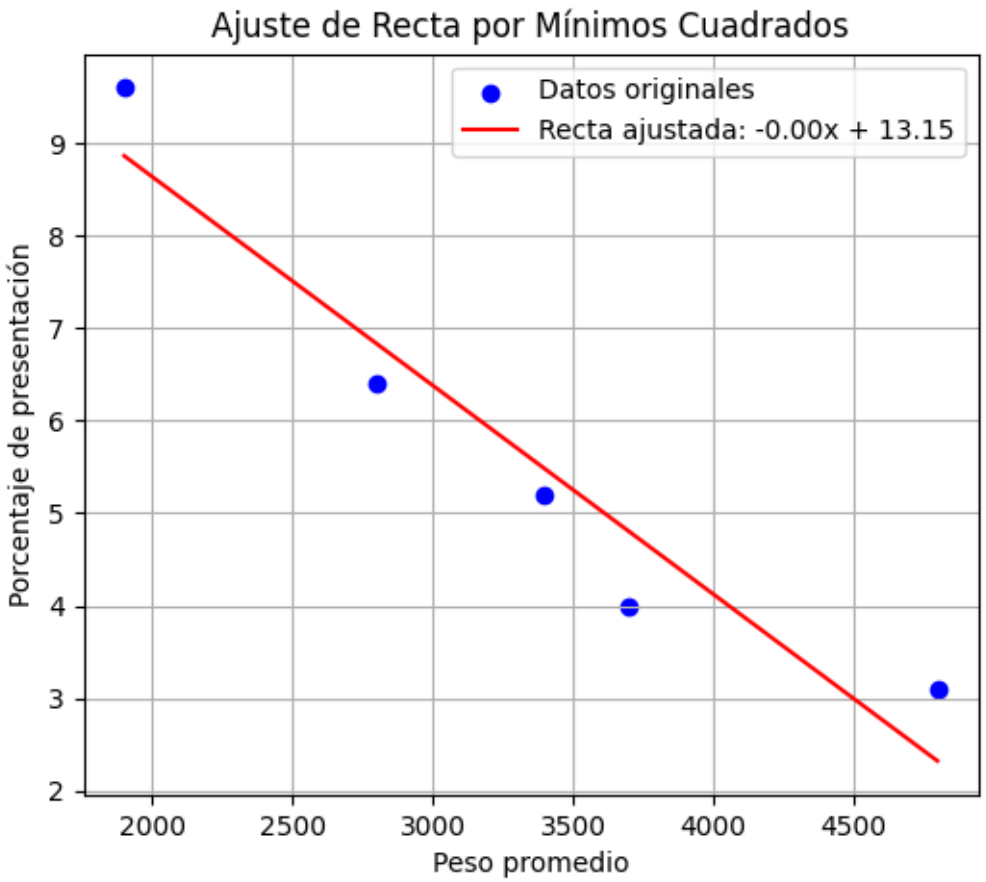
El siguiente conjunto de datos, presentado al Subcomité Antimonopolio del Senado, muestra las características comparativas de supervivencia durante un choque de automóviles de diferentes clases. Encuentre la recta por mínimos cuadrados que aproxima estos datos (la tabla muestra el porcentaje de vehículos que participaron en un accidente en los que la lesión más grave fue fatal o seria).

Tipo	Peso promedio	Porcentaje de presentación
1. Regular lujoso doméstico	4800 lb	3.1
2. Regular intermediario doméstico	3700 lb	4.0
3. Regular económico doméstico	3400 lb	5.2
4. Compacto doméstico	2800 lb	6.4
5. Compacto extranjero	1900 lb	9.6

[14]:
```
import numpy as np
import matplotlib.pyplot as plt
```

```python
peso_promedio = np.array([4800, 3700, 3400, 2800, 1900])
porcentaje_presentacion = np.array([3.1, 4.0, 5.2, 6.4, 9.6])

coeffs = np.polyfit(peso_promedio, porcentaje_presentacion, 1)
y_pred = np.polyval(coeffs, peso_promedio)
error = np.mean((porcentaje_presentacion - y_pred)**2)

print("--- RESULTADOS ---")
print(f"Coeficientes: {coeffs}")
print(f"Error (ECM): {error:.4f}")

plt.figure(figsize=(6, 5))
plt.scatter(peso_promedio, porcentaje_presentacion, color='blue', label='Datos␣
 ↪originales')
plt.plot(peso_promedio, y_pred, color='red', label=f'Recta ajustada: {coeffs[0]:
 ↪.2f}x + {coeffs[1]:.2f}')
plt.legend()
plt.xlabel('Peso promedio')
plt.ylabel('Porcentaje de presentación')
plt.title('Ajuste de Recta por Mínimos Cuadrados')
plt.grid(True)
plt.show()
```

```
--- RESULTADOS ---
Coeficientes: [-2.25496975e-03  1.31464996e+01]
Error (ECM): 0.4118
```


Ajuste de Recta por Mínimos Cuadrados
Datos originales
Recta ajustada: -0.00x + 13.15
Porcentaje de presentación
Peso promedio
2
3
4
5
6
7
8
9
2000
2500
3000
3500
4000
4500