Participación en clase 11 - Ejercicio mínimos cuadrados

Nombre: Alexis Bautista
Fecha de Entrega: 22 de enero de 2025
Paralelo: GR1CC
Enlace de GitHub: https://github.com/alexis-bautista/Participacion-11-12-MN/blob/main/min_cuadrados.ipynb

Prueba 02

Interpole los siguientes conjuntos de datos con la función correspondiente.

La ecuación de la línea es:

$y(x) = a_1 x + a_0$

Al realizar el proceso de mínimos cuadrados queda el siguiente sistema de ecuaciones:

$(\sum_i (y_i - a_1 x_i - a_0), \sum_i (y_i - a_1 x_i - a_0) x_i) = 0$

In [3]:
```python
# Derivadas parciales para regresión lineal
# ######################################################################
def der_parcial_1(xs: list, ys: list) -> tuple[float, float, float]:
    """Retorna los coeficientes de la ecuación de la derivada parcial con respec
    c_1 * a_1 + c_0 * a_0 = c_ind


    ## Parameters

    ``xs``: lista de valores de x.

    ``ys``: lista de valores de y.

    ## Return

    ``c_1``: coeficiente del parámetro 1.

    ``c_0``: coeficiente del parámetro 0.

    ``c_ind``: coeficiente del término independiente.

    """

    # coeficiente del término independiente
    c_ind = sum(ys)

    # coeficiente del parámetro 1
    c_1 = sum(xs)

    # coeficiente del parámetro 0
```

```python
    c_0 = len(xs)

    return (c_1, c_0, c_ind)


def der_parcial_0(xs: list, ys: list) -> tuple[float, float, float]:
    """Retorna los coeficientes de la ecuación de la derivada parcial con respec
    c_1 * a_1 + c_0 * a_0 = c_ind


    ## Parameters

    ``xs``: lista de valores de x.

    ``ys``: lista de valores de y.

    ## Return

    ``c_1``: coeficiente del parámetro 1.

    ``c_0``: coeficiente del parámetro 0.

    ``c_ind``: coeficiente del término independiente.

    """
    c_1 = 0
    c_0 = 0
    c_ind = 0
    for xi, yi in zip(xs, ys):
        # coeficiente del término independiente
        c_ind += xi * yi

        # coeficiente del parámetro 1
        c_1 += xi * xi

        # coeficiente del parámetro 0
        c_0 += xi

    return (c_1, c_0, c_ind)
```

Conjunto de datos de ejemplo

In [4]:
```python
xs = [
    -5.0000,
    -3.8889,
    -2.7778,
    -1.6667,
    -0.5556,
    0.5556,
    1.6667,
    2.7778,
    3.8889,
    5.0000,
]
ys = [
    -12.7292,
    -7.5775,
    -7.7390,
```

```
    -4.1646,
    -4.5382,
    2.2048,
    4.3369,
    2.2227,
    9.0625,
    7.1860,
]
```

In [5]:

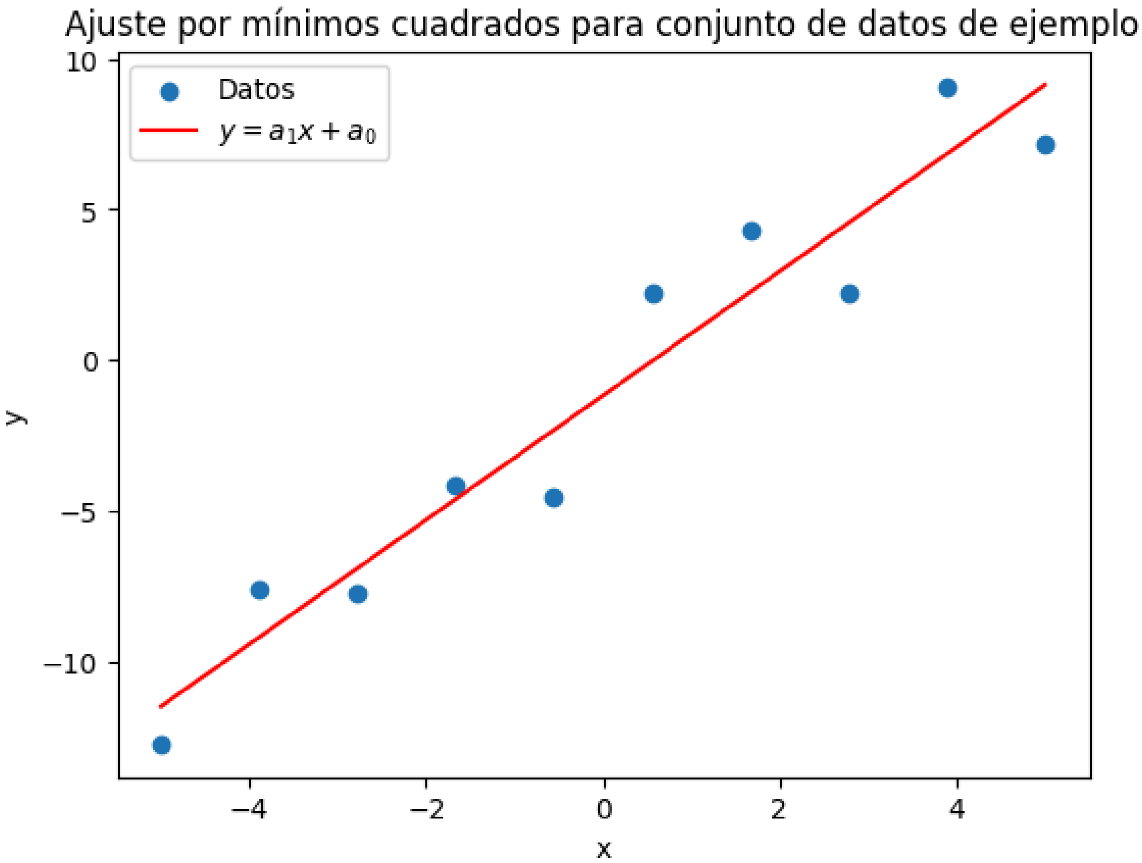
```
from src import ajustar_min_cuadrados  # no modificar esta función

pars = ajustar_min_cuadrados(xs, ys, gradiente=[der_parcial_0, der_parcial_1])
```

```
[01-22 22:21:26][INFO] Se ajustarán 2 parámetros.
[01-22 22:21:26][INFO]
[[101.8525926    0.         209.87476711]
 [  0.          10.         -11.7356    ]]
[01-22 22:21:26][INFO]
[[101.8525926    0.         209.87476711]
 [  0.          10.         -11.7356    ]]
```

In [6]:

```
import numpy as np
import matplotlib.pyplot as plt

m, b = ajustar_min_cuadrados(xs, ys, gradiente=[der_parcial_0, der_parcial_1])

x = np.linspace(-5, 5, 100)

y = [m * xi + b for xi in x]

plt.scatter(xs, ys, label="Datos")
plt.plot(x, y, color="red", label=r"$ y = a_1 x + a_0 $")
plt.xlabel("x")
plt.ylabel("y")
plt.title("Ajuste por mínimos cuadrados para conjunto de datos de ejemplo")
plt.legend()
plt.show()
```

```
[01-22 22:21:32][INFO] Se ajustarán 2 parámetros.
[01-22 22:21:32][INFO]
[[101.8525926    0.         209.87476711]
 [  0.          10.         -11.7356    ]]
```

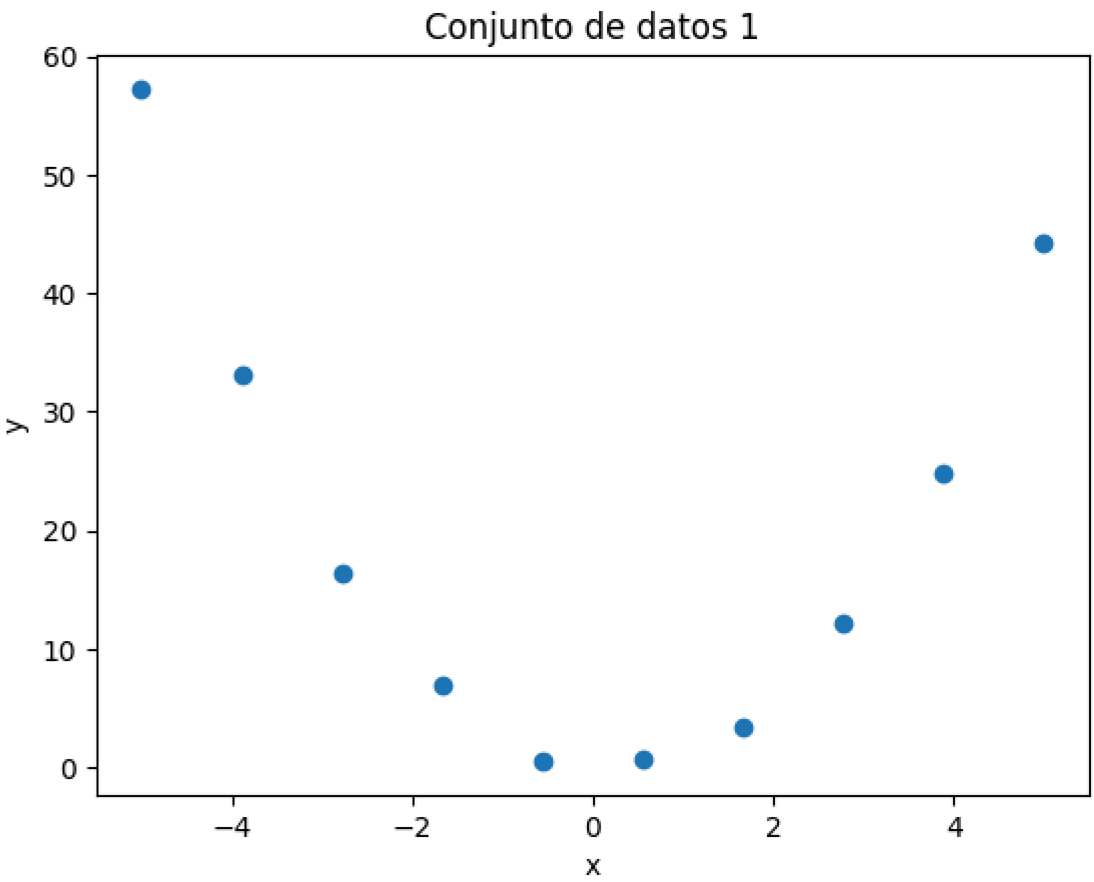


Conjunto de datos 1

In [7]:
```
xs1 = [
    -5.0000,
    -3.8889,
    -2.7778,
    -1.6667,
    -0.5556,
    0.5556,
    1.6667,
    2.7778,
    3.8889,
    5.0000,
]
ys1 = [
    57.2441,
    33.0303,
    16.4817,
    7.0299,
    0.5498,
    0.7117,
    3.4185,
    12.1767,
    24.9167,
    44.2495,
]
```

In [8]:
```
plt.scatter(xs1, ys1)
plt.xlabel("x")
plt.ylabel("y")
```

```
plt.title("Conjunto de datos 1")
plt.show()
```



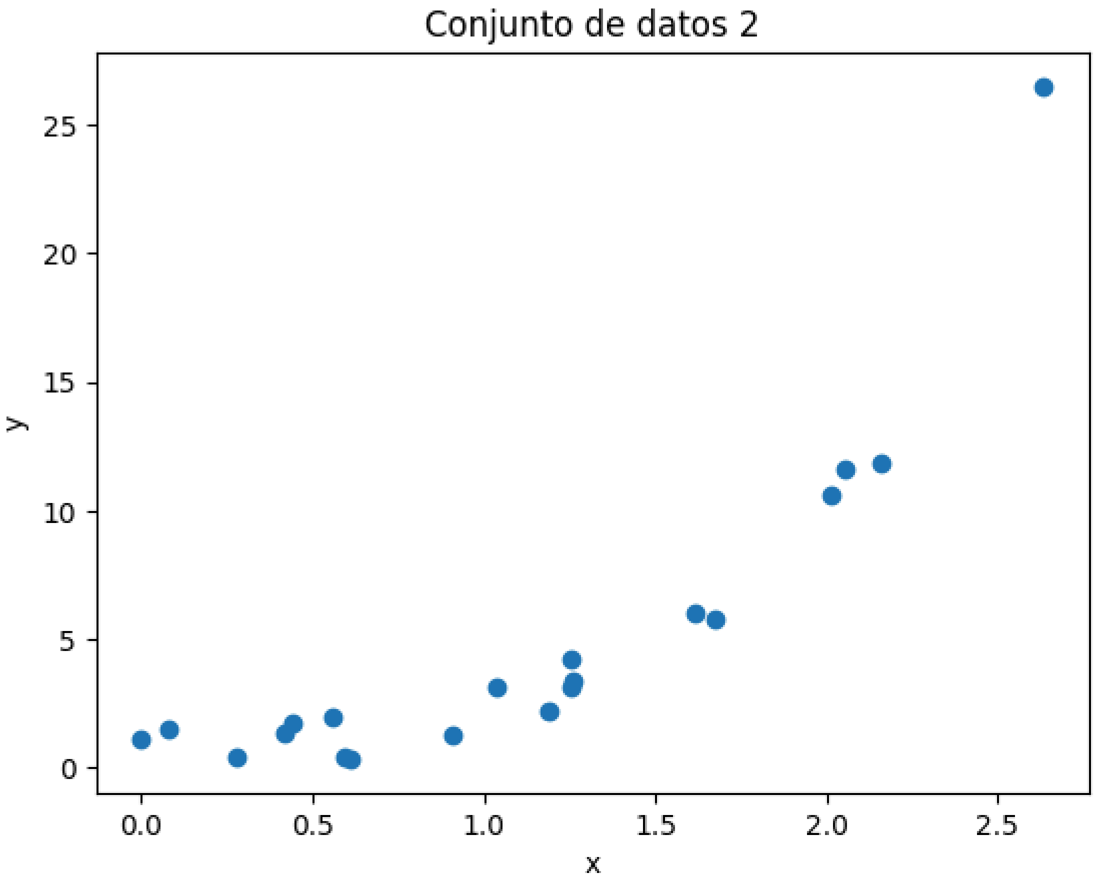
Interpole el conjunto de datos 1 usando la función cuadrática.

Conjunto de datos 2

In [9]:
```
xs2 = [
    0.0003,
    0.0822,
    0.2770,
    0.4212,
    0.4403,
    0.5588,
    0.5943,
    0.6134,
    0.9070,
    1.0367,
    1.1903,
    1.2511,
    1.2519,
    1.2576,
    1.6165,
    1.6761,
    2.0114,
    2.0557,
    2.1610,
    2.6344,
]
ys2 = [
    1.1017,
```

```
    1.5021,
    0.3844,
    1.3251,
    1.7206,
    1.9453,
    0.3894,
    0.3328,
    1.2887,
    3.1239,
    2.1778,
    3.1078,
    4.1856,
    3.3640,
    6.0330,
    5.8088,
    10.5890,
    11.5865,
    11.8221,
    26.5077,
]
```

In [10]:
```
plt.scatter(xs2, ys2)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Conjunto de datos 2")
plt.show()
```



Interpole el conjunto de datos 2 usando la función exponencial.

Modificaciones:

Conjunto de Datos 1

Conocemos que la funcion ideona para interpolar estos puntos es: $y = a_2x^2 + a_1^x + a_0$

Entonces calculamos las derivadas parciales respecto a a_0, a_1ya_2, de donde tenemos que:

$E = \sum_{i=1}^{n} [y_i - (a_0 + a_1^x + a_2x^2)]^2$

$\frac{\partial E}{\partial a_0} = \sum_{i=1}^{n}(a_0 + x_ia_1 + x_i^2a_2 = y_i)$

$\frac{\partial E}{\partial a_1} = \sum_{i=1}^{n}(x_ia_0 + x_i^2a_1 + x_i^3a_2 = x_i * y_i)$

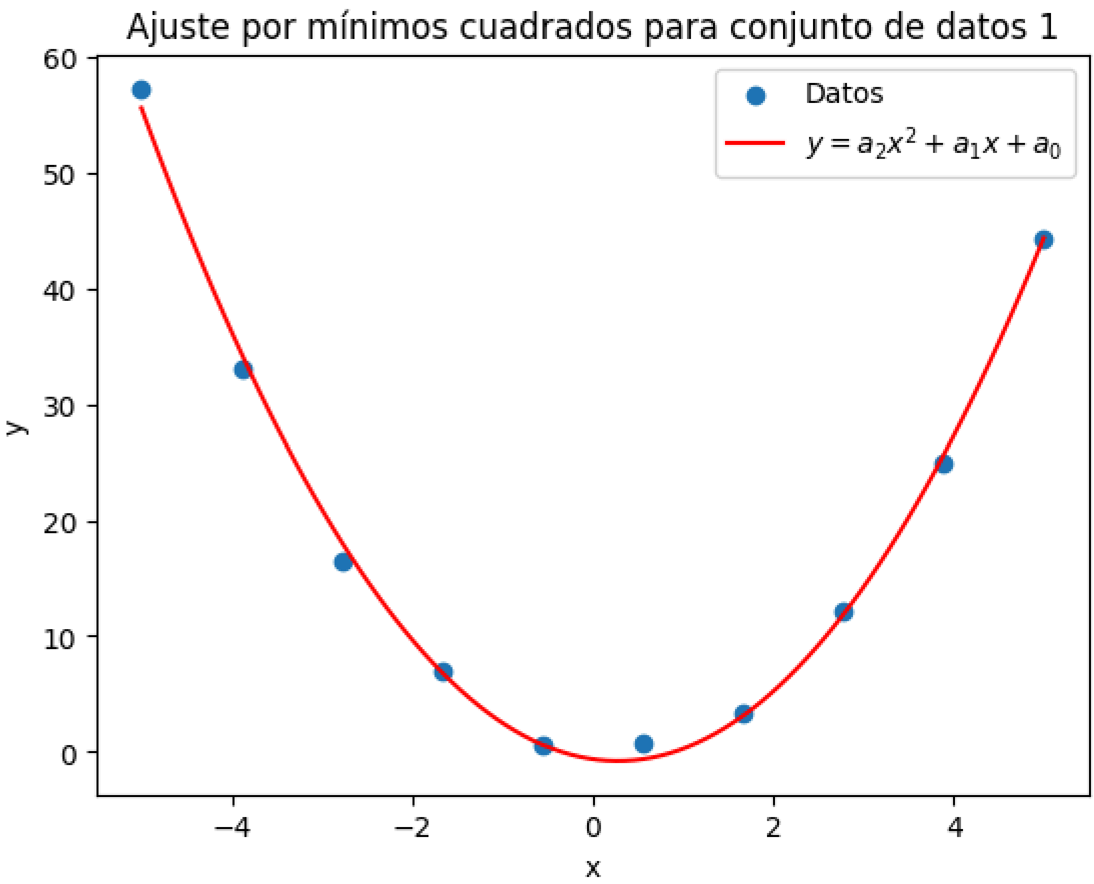
$\frac{\partial E}{\partial a_2} = \sum_{i=1}^{n}(x_i^2a_0 + x_i^3a_1 + x_i^4a_2 = x_i^2 * y_i)$

In [20]:
```
def der_parcial_2(xs: list, ys: list) -> tuple[float, float, float, float]:
    c_2 = sum(xi ** 4 for xi in xs)
    c_1 = sum(xi ** 3 for xi in xs)
    c_0 = sum(xi ** 2 for xi in xs)
    c_ind = sum(yi * xi ** 2 for yi, xi in zip(ys, xs))
    return (c_2, c_1, c_0, c_ind)

def der_parcial_1(xs: list, ys: list) -> tuple[float, float, float, float]:
    c_2 = sum(xi ** 3 for xi in xs)
    c_1 = sum(xi ** 2 for xi in xs)
    c_0 = sum(xi for xi in xs)
    c_ind = sum(yi * xi for yi, xi in zip(ys, xs))
    return (c_2, c_1, c_0, c_ind)

def der_parcial_0(xs: list, ys: list) -> tuple[float, float, float, float]:
    c_2 = sum(xi ** 2 for xi in xs)
    c_1 = sum(xi for xi in xs)
    c_0 = len(xs)
    c_ind = sum(ys)
    return (c_2, c_1, c_0, c_ind)
```

In [19]:
```
from src import ajustar_min_cuadrados

# Obtenemos los parametros
parametros = ajustar_min_cuadrados(xs1, ys1, gradiente=[der_parcial_2, der_parci

a2, a1, a0 = parametros

# Graficamos
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-5, 5, 100)
y = a2 * x ** 2 + a1 * x + a0

print("\n Los parametros obtenidos son: ", a0 , a1, a2)
```

```
plt.scatter(xs1, ys1, label="Datos")
plt.plot(x, y, color="red", label=r"$ y = a_2 x^2 + a_1 x + a_0 $")
plt.xlabel("x")
plt.ylabel("y")
plt.title("Ajuste por mínimos cuadrados para conjunto de datos 1")
plt.legend()
plt.show()
```

```
[01-22 22:30:54][INFO] Se ajustarán 3 parámetros.
[01-22 22:30:54][INFO]
[[ 1.01852593e+02  0.00000000e+00  1.00000000e+01  1.99808900e+02]
 [ 0.00000000e+00  1.01852593e+02  0.00000000e+00 -1.14413577e+02]
 [-2.27373675e-13  0.00000000e+00 -7.90113041e+01  5.04294087e+01]]
[01-22 22:30:54][INFO]
[[ 1.01852593e+02  0.00000000e+00  1.00000000e+01  1.99808900e+02]
 [ 0.00000000e+00  1.01852593e+02  0.00000000e+00 -1.14413577e+02]
 [-2.27373675e-13  0.00000000e+00 -7.90113041e+01  5.04294087e+01]]

 Los parametros obtenidos son:  -0.6382556172537739 -1.123325129575543 2.02441048
2925083
```



Finalmente tenemos que la ecuacion es $y = 2.02x^2 - 1.12x - 0.64$

$y(2.25) = 7.083$
$y(-2.25) = 12.138$

Conjuntos de Datos 2

Conocemos que la funcion ideona para interpolar estos puntos es: $y = a * e^{bx}$

Entonces calculamos las derivadas parciales respecto a $a_0, a_1 y a_2$, de donde tenemos que:

$$E = \sum_{i=1}^{n} [y_i - (a * e^{bx})]^2$$

$$\frac{\partial E}{\partial a} = -2\sum_{i=1}^{n} [y_i - (ae^{bx_i})]e^{bx_i}$$

$$\frac{\partial E}{\partial b} = -2\sum_{i=1}^{n} [y_i - (ae^{bx_i})](ax_i e^{bx_i})$$

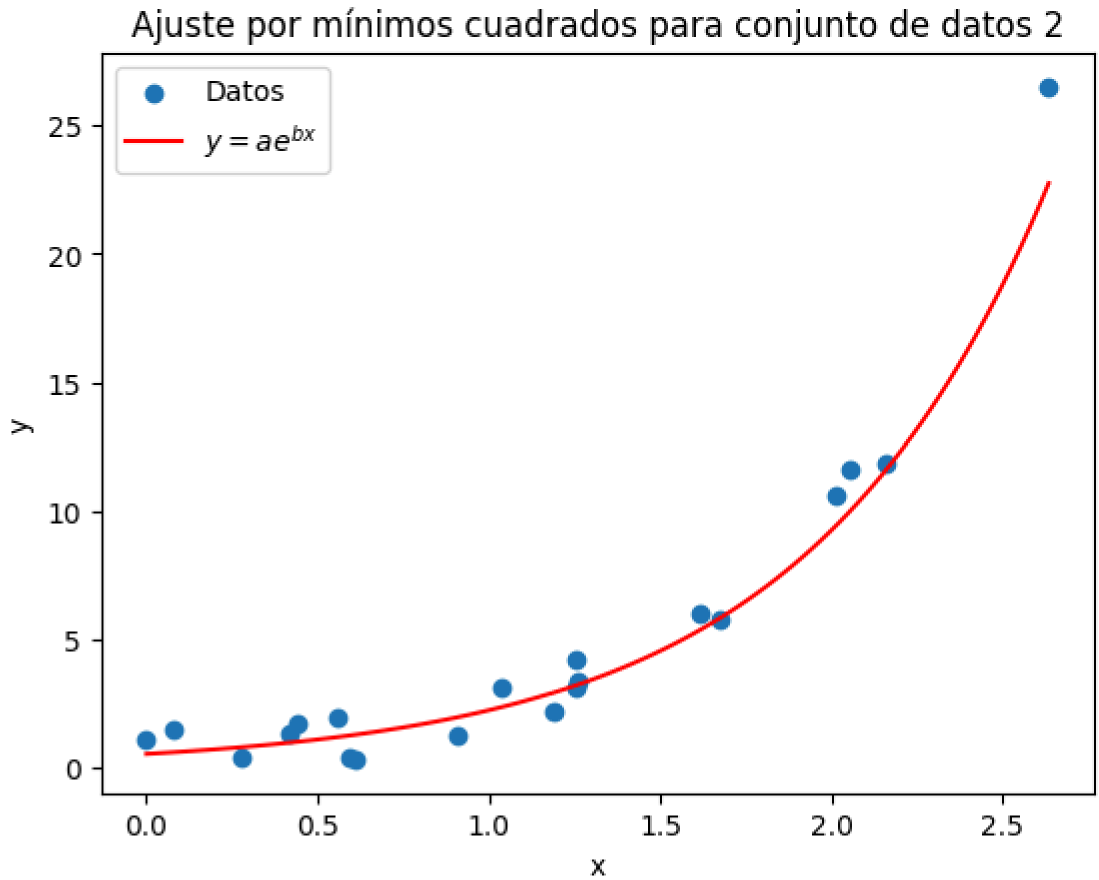
In [16]:

```
import numpy as np

def der_parcial_exp_1(xs: list, ys: list) -> tuple[float, float]:
    log_ys = np.log(ys)
    c_1 = sum(xs)
    c_0 = len(xs)
    c_ind = sum(log_ys)
    return (c_1, c_0, c_ind)

def der_parcial_exp_0(xs: list, ys: list) -> tuple[float, float]:
    log_ys = np.log(ys)
    c_1 = sum(xi * xi for xi in xs)
    c_0 = sum(xs)
    c_ind = sum(xi * yi for xi, yi in zip(xs, log_ys))
    return (c_1, c_0, c_ind)
```

In [22]:

```
from src import ajustar_min_cuadrados

# Calculamos lo parametros
parametros = ajustar_min_cuadrados(xs2, ys2, gradiente=[der_parcial_exp_0, der_p

b, log_a = parametros
a = np.exp(log_a)

print("\n Los parametros obtenidos son: ", a, b)

# Graficamos
x = np.linspace(min(xs2), max(xs2), 100)
y = a * np.exp(b * x)

plt.scatter(xs2, ys2, label="Datos")
plt.plot(x, y, color="red", label=r"$ y = a e^{bx} $")
plt.xlabel("x")
plt.ylabel("y")
plt.title("Ajuste por mínimos cuadrados para conjunto de datos 2")
plt.legend()
plt.show()
```

```
[01-22 23:04:14][INFO] Se ajustarán 2 parámetros.
[01-22 23:04:14][INFO]
[[22.0372     20.         19.05727035]
 [ 0.         -9.57184451  5.82589171]]

 Los parametros obtenidos son:  0.5440855388147081 1.4171603667055415
```



Finalmente tenemos que la ecuacion es $y = 0.544e^{1.417x}$

$y(5) = 650.1174$
$y(1) = 2.2446$