

Лабораторна робота 1.1 студента С. О. Семерікова

Варіант 10: Споживання електроенергії

Опис: Залежність споживання електроенергії (кВт·год) від середньодобової температури (°C). Набір даних:

x_train = np.array([-5, 0, 5, 10, 15, 20, 25, 30, 35]) # температура в °C

y_train = np.array([320, 280, 240, 200, 170, 150, 170, 210, 260]) # споживання в кВт·год

Рекомендовані параметри для початку:

- w_init = -4
- b_init = 200
- alpha = 0.01
- iterations = 10000

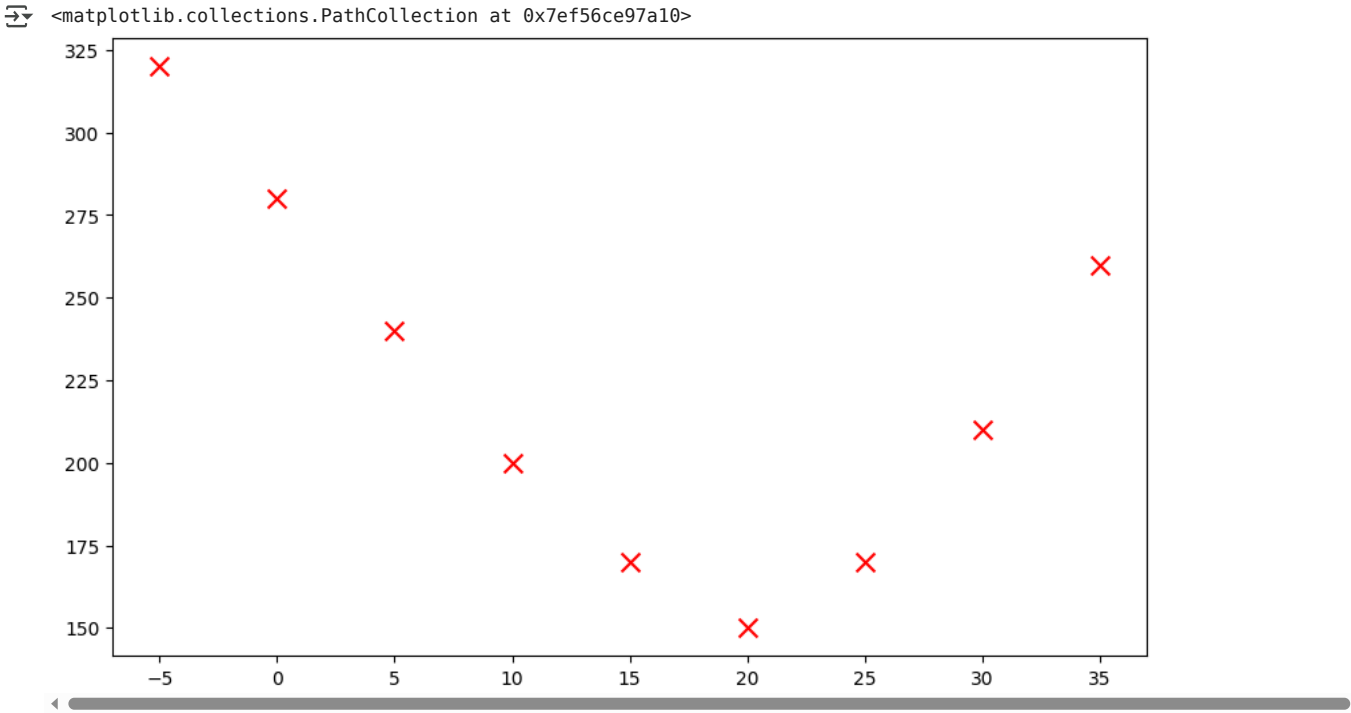
```
w2_init = -4
w1_init = -4
b_init = 200
alpha = 0.01
iterations = 10000
```

2. Підготовка даних
- * Імпортуйте необхідні бібліотеки
 - * Завантажте дані відповідно до вашого варіанту
 - * Візуалізуйте дані для розуміння їх структури

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

x_train = np.array([-5, 0, 5, 10, 15, 20, 25, 30, 35]) # температура в °C
y_train = np.array([320, 280, 240, 200, 170, 150, 170, 210, 260]) # споживання в кВт·год

plt.figure(figsize=(10, 6))
plt.scatter(x_train, y_train, marker='x', c='r', s=100, label='Навчальні дані')
```



- 1: $y_{\text{обч}} = f(x) = wx + b$ - погано
- 2: $y_{\text{обч}} = w_2x^2 + w_1x + b$ - можливо, краще

3. Реалізація функцій
- * Реалізуйте функцію обчислення вихідних значень моделі compute_model_output
 - * Реалізуйте функцію обчислення вартості compute_cost
 - * Реалізуйте функцію обчислення градієнту compute_gradient
 - * Реалізуйте функцію градієнтного спуску gradient_descent

```
# 1. Функція обчислення моделі
def compute_model_output(x, w1, w2, b):
    """
    Обчислює прогноз лінійної моделі
    Аргументи:
        x (ndarray (m,)): Дані, m прикладів
        w, b (скаляри): параметри моделі
    Повертає:
        y (ndarray (m,)): цільові значення
    """
    m = x.shape[0]
    f_wb = np.zeros(m)
    for i in range(m):
        f_wb[i] = w2 * x[i]**2 + w1 * x[i] + b

    return f_wb
```

```
y_hat = compute_model_output(x_train, w1_init, w2_init, b_init)
```

```
y_hat
array([ 120.,  200.,   80., -240., -760., -1480., -2400., -3520.,
       -4840.]
```

```
y_train
array([320, 280, 240, 200, 170, 150, 170, 210, 260])
```


```
# 2. Функція обчислення вартості
def compute_cost(x, y, w1, w2, b):
    """
```

```
Обчислює функцію вартості для лінійної регресії
Аргументи:
  x (ndarray (m,)): Дані, m прикладів
  y (ndarray (m,)): цільові значення
  w1, w2, b (скаляри): параметри моделі
Повертає:
  total_cost (float): вартість використання w,b як параметрів для лінійної регресії
"""
m = x.shape[0]

total_cost = 0
f_wb = compute_model_output(x, w1, w2, b)
for i in range(m):
    total_cost = total_cost + (f_wb[i] - y[i]) ** 2
total_cost = (1 / (2 * m)) * total_cost

return total_cost
```

```
J = compute_cost(x_train, y_train, w1_init, w2_init, b_init)
J
```

 np.float64(2795288.8888888885)


```
# 3. Функція обчислення градієнта
def compute_gradient(x, y, w1, w2, b):
    """
    Обчислює градієнт для лінійної регресії
    Аргументи:
      x (ndarray (m,)): Дані, m прикладів
      y (ndarray (m,)): цільові значення
      w1, w2, b (скаляри): параметри моделі
    Повертає:
      dj_dw1 (скаляр): Градієнт функції вартості відносно параметра w1
      dj_dw2 (скаляр): Градієнт функції вартості відносно параметра w2
      dj_db (скаляр): Градієнт функції вартості відносно параметра b
    """
    m = x.shape[0]
    dj_dw1 = 0
    dj_dw2 = 0
    dj_db = 0

    f_wb = compute_model_output(x, w1, w2, b)
    for i in range(m):
        dj_dw1 += (f_wb[i] - y[i]) * x[i]
        dj_dw2 += (f_wb[i] - y[i]) * x[i] ** 2
        dj_db += (f_wb[i] - y[i]) * 1

    dj_dw1 = dj_dw1 / m
    dj_dw2 = dj_dw2 / m
    dj_db = dj_db / m

    return dj_dw1, dj_dw2, dj_db
```

```
dJ_dw1, dJ_dw2, dJ_db = compute_gradient(x_train, y_train, w1_init, w2_init, b_init)
dJ_dw1, dJ_dw2, dJ_db
```

 (np.float64(-45044.444444444445),
np.float64(-1347222.2222222222),
np.float64(-1648.888888888889))

```
# 4. Функція градієнтного спуску
def gradient_descent(x, y, w1_in, w2_in, b_in, alpha, num_iters):
    """
    Виконує градієнтний спуск для пошуку w1, w2, b

    Аргументи:
      x (ndarray (m,)): Дані, m прикладів
      y (ndarray (m,)): цільові значення
      w1_in, w2_in, b_in (скаляри): початкові значення параметрів моделі
      alpha (float): швидкість навчання
      num_iters (int): кількість ітерацій градієнтного спуску

    Повертає:
      w1 (скаляр): Оновлене значення параметра w1 після градієнтного спуску
      w2 (скаляр): Оновлене значення параметра w2 після градієнтного спуску
      b (скаляр): Оновлене значення параметра b після градієнтного спуску
      J_history (List): Історія значень функції вартості
      p_history (List): Історія параметрів [w,b]
    """

    # Масив для збереження значень вартості J та параметрів w, b
    J_history = []
    p_history = []
    b = b_in
    w1 = w1_in
    w2 = w2_in

    for i in range(num_iters):
        # Обчислення градієнту
        dj_dw1, dj_dw2, dj_db = compute_gradient(x, y, w1, w2, b)


        # Оновлення параметрів
        w1 = w1 - alpha * dj_dw1
        w2 = w2 - alpha * dj_dw2
        b = b - alpha * dj_db

        # Збереження історії
        J_history.append(compute_cost(x, y, w1, w2, b))
        p_history.append([w1, w2, b])

        # Вивід проміжних результатів
        if i % (num_iters // 10) == 0 or i == num_iters - 1:
            print(f"Ітерація {i:4}: Вартість {J_history[-1]:0.2e}, ",
                  f"dj_dw1: {dj_dw1: 0.3e}, dj_dw2: {dj_dw2: 0.3e}, dj_db: {dj_db: 0.3e}, ",
                  f"w1: {w1: 0.3e}, w2: {w2: 0.3e}, b: {b: 0.5e}")

    return w1, w2, b, J_history, p_history
```

```
w1, w2, b, J_history, p_history = gradient_descent(x_train, y_train, w1_init, w2_init, b_init, alpha, iterations)
```

 Ітерація 0: Вартість 2.95e+13, dj_dw1: -4.504e+04, dj_dw2: -1.347e+06, dj_db: -1.649e+03, w1: 4.464e+02, w2: 1.347e+04, b: 2.16489e+02
Ітерація 1000: Вартість nan, dj_dw1: nan, dj_dw2: nan, dj_db: nan, w1: nan, w2: nan, b: nan
Ітерація 2000: Вартість nan, dj_dw1: nan, dj_dw2: nan, dj_db: nan, w1: nan, w2: nan, b: nan
Ітерація 3000: Вартість nan, dj_dw1: nan, dj_dw2: nan, dj_db: nan, w1: nan, w2: nan, b: nan
Ітерація 4000: Вартість nan, dj_dw1: nan, dj_dw2: nan, dj_db: nan, w1: nan, w2: nan, b: nan
Ітерація 5000: Вартість nan, dj_dw1: nan, dj_dw2: nan, dj_db: nan, w1: nan, w2: nan, b: nan
Ітерація 6000: Вартість nan, dj_dw1: nan, dj_dw2: nan, dj_db: nan, w1: nan, w2: nan, b: nan

```
alpha /= 10
w1, w2, b, J_history, p_history = gradient_descent(x_train, y_train, w1_init, w2_init, b_init, alpha, iterations)
```

```
alpha /= 10
w1, w2, b, J history, p history = gradient_descent(x_train, y_train, w1_init, w2_init, b_init, alpha, iterations)
```

```
alpha /= 10
w1, w2, b, J_history, p_history = gradient_descent(x_train, y_train, w1_init, w2_init, b_init, alpha, iterations)
```

```
alpha /= 10
w1, w2, b, J history, p history = gradient_descent(x_train, y_train, w1_init, w2_init, b_init, alpha, iterations)
```

$$\alpha = 0.00003$$

```
w1, w2, b, J history, p history = gradient_descent(x_train, y_train, w1_init, w2_init, b_init, alpha, iterations)
```

<https://colab.research.google.com/drive/1aPGsPJ7zuhqshijjUVRy0AoDQ72NFadC?authuser=1#scrollTo=Bl0dFoEIUtl&printMode=true>

```
4. Запуск градієнтного спуску
* Ініціалізуйте початкові значення параметрів
* Виконайте алгоритм градієнтного спуску для знаходження оптимальних параметрів
```

```
iterations = 1600000
```

```
alpha = 0.0000061
```

```
w1, w2, b, J_history, p_history = gradient_descent(x_train, y_train, w1_init, w2_init, b_init, alpha, iterations)
```

```
↗ Ітерація 0: Вартість 2.70e+06, dj_dw1: -4.504e+04, dj_dw2: -1.347e+06, dj_db: -1.649e+03, w1: -3.725e+00, w2: 4.218e+00, b: 2.00010e+02
Ітерація 160000: Вартість 5.48e+02, dj_dw1: 1.350e+00, dj_dw2: -2.178e-02, dj_db: -1.944e+01, w1: -7.872e+00, w2: 2.499e-01, b: 2.23003e+02
Ітерація 320000: Вартість 2.90e+02, dj_dw1: 9.169e-01, dj_dw2: -1.479e-02, dj_db: -1.320e+01, w1: -8.965e+00, w2: 2.675e-01, b: 2.38731e+02
Ітерація 480000: Вартість 1.71e+02, dj_dw1: 6.226e-01, dj_dw2: -1.004e-02, dj_db: -8.960e+00, w1: -9.707e+00, w2: 2.795e-01, b: 2.49411e+02
Ітерація 640000: Вартість 1.16e+02, dj_dw1: 4.227e-01, dj_dw2: -6.817e-03, dj_db: -6.084e+00, w1: -1.021e+01, w2: 2.876e-01, b: 2.56662e+02
Ітерація 800000: Вартість 9.10e+01, dj_dw1: 2.870e-01, dj_dw2: -4.629e-03, dj_db: -4.131e+00, w1: -1.055e+01, w2: 2.931e-01, b: 2.61586e+02
Ітерація 960000: Вартість 7.94e+01, dj_dw1: 1.949e-01, dj_dw2: -3.143e-03, dj_db: -2.805e+00, w1: -1.078e+01, w2: 2.968e-01, b: 2.64929e+02
Ітерація 1120000: Вартість 7.40e+01, dj_dw1: 1.323e-01, dj_dw2: -2.134e-03, dj_db: -1.905e+00, w1: -1.094e+01, w2: 2.994e-01, b: 2.67199e+02
Ітерація 1280000: Вартість 7.15e+01, dj_dw1: 8.985e-02, dj_dw2: -1.449e-03, dj_db: -1.293e+00, w1: -1.105e+01, w2: 3.011e-01, b: 2.68740e+02
Ітерація 1440000: Вартість 7.04e+01, dj_dw1: 6.101e-02, dj_dw2: -9.838e-04, dj_db: -8.781e-01, w1: -1.112e+01, w2: 3.023e-01, b: 2.69786e+02
Ітерація 1599999: Вартість 6.99e+01, dj_dw1: 4.142e-02, dj_dw2: -6.680e-04, dj_db: -5.962e-01, w1: -1.117e+01, w2: 3.031e-01, b: 2.70497e+02
```

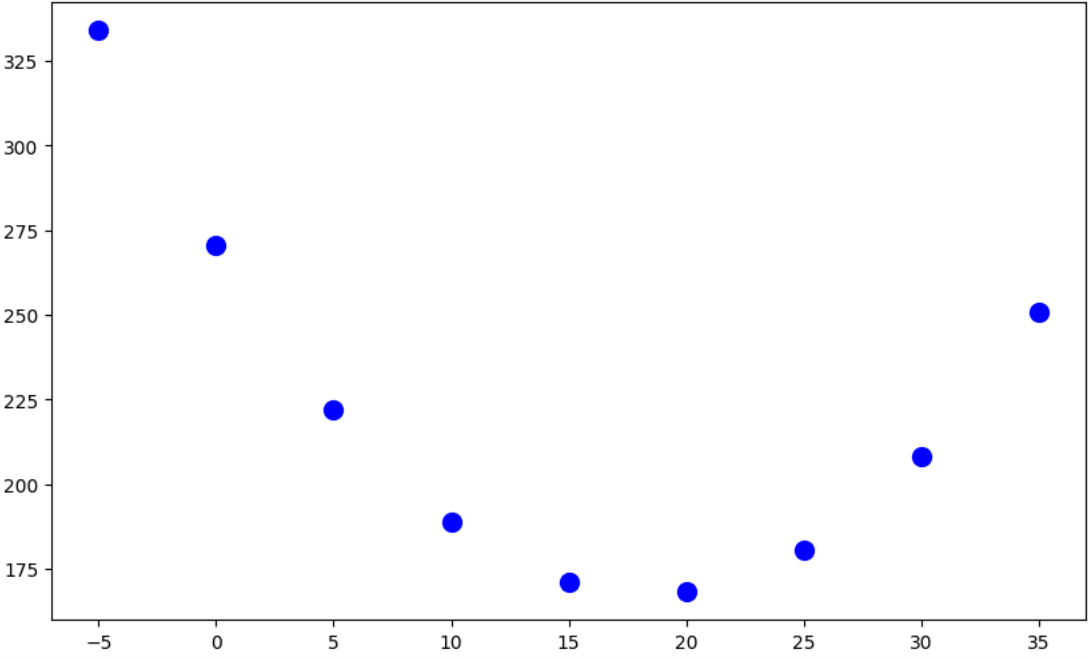
```
y_hat = compute_model_output(x_train, w1, w2, b)
```

```
w1, w2, b
```

```
↗ (np.float64(-11.171761228854031),
 np.float64(0.30307779298607374),
 np.float64(270.4969789048082))
```

```
plt.figure(figsize=(10, 6))
plt.scatter(x_train, y_hat, marker='o', c='b', s=100, label='Обчислені дані')
```

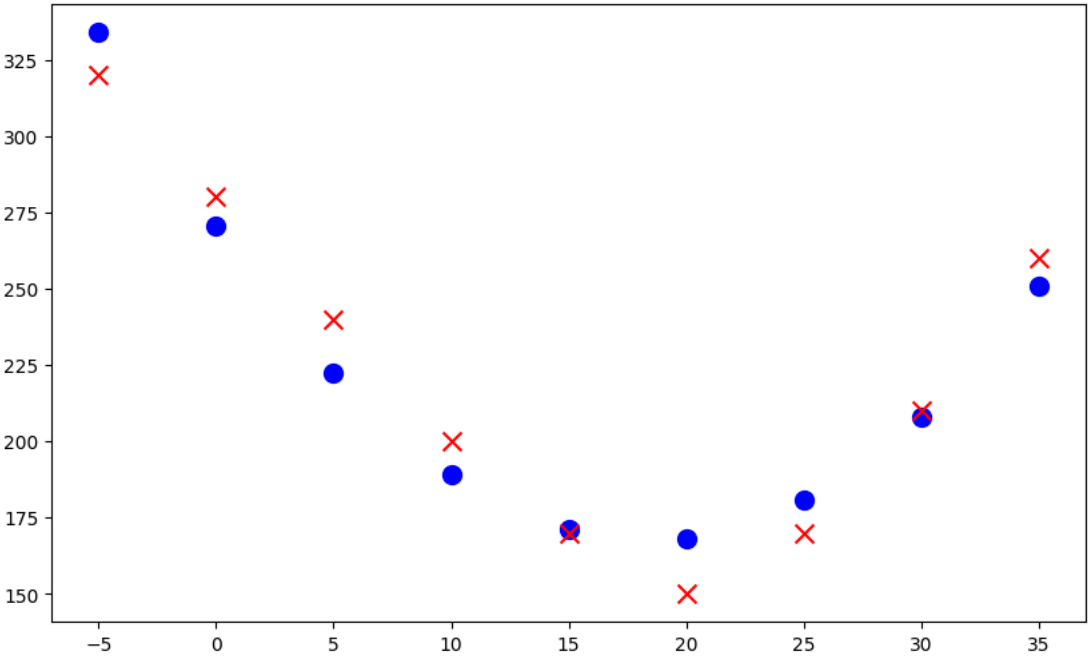
```
↗ <matplotlib.collections.PathCollection at 0x7ef54160dd90>
```



```
plt.figure(figsize=(10, 6))
plt.scatter(x_train, y_hat, marker='o', c='b', s=100, label='Обчислені дані') + plt.scatter(x_train, y_train, marker='x', c='r', s=100, label='Навчальні дані')
```

```
↗ -----
TypeError                                Traceback (most recent call last)
<ipython-input-34-7bf35d6e3e2f> in <cell line: 0>()
      1 plt.figure(figsize=(10, 6))
----> 2 plt.scatter(x_train, y_hat, marker='o', c='b', s=100, label='Обчислені дані') + plt.scatter(x_train, y_train, marker='x', c='r', s=100, label='Навчальні дані')

TypeError: unsupported operand type(s) for +: 'PathCollection' and 'PathCollection'
```

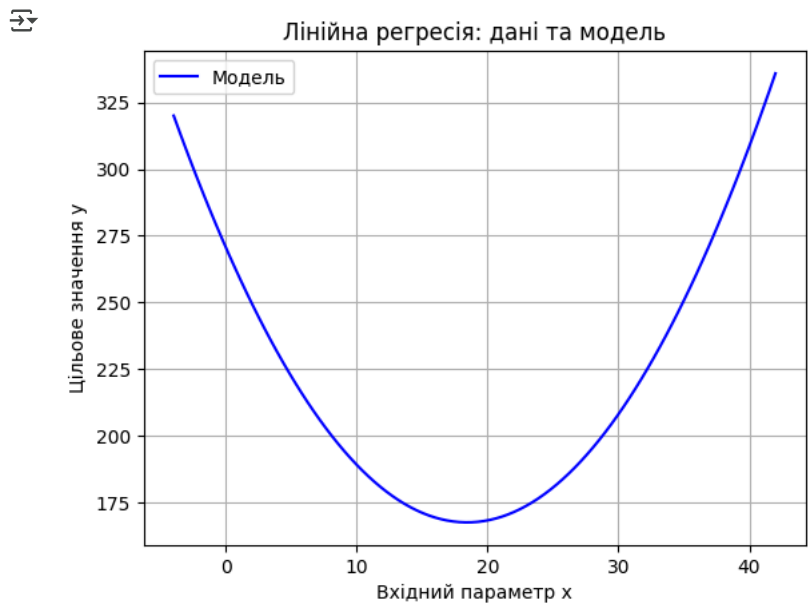


```
5. Аналіз та візуалізація результатів
  • Візуалізуйте дані та отриману модель
  • Проаналізуйте процес навчання (зміну функції вартості)
  • Використайте модель для прогнозування
```

```
# Обчислення значень моделі для виводу лінії
x_line = np.linspace(np.min(x_train) * 0.8, np.max(x_train) * 1.2, 100)
y_line = w1 * x_line + w2 * x_line ** 2 + b

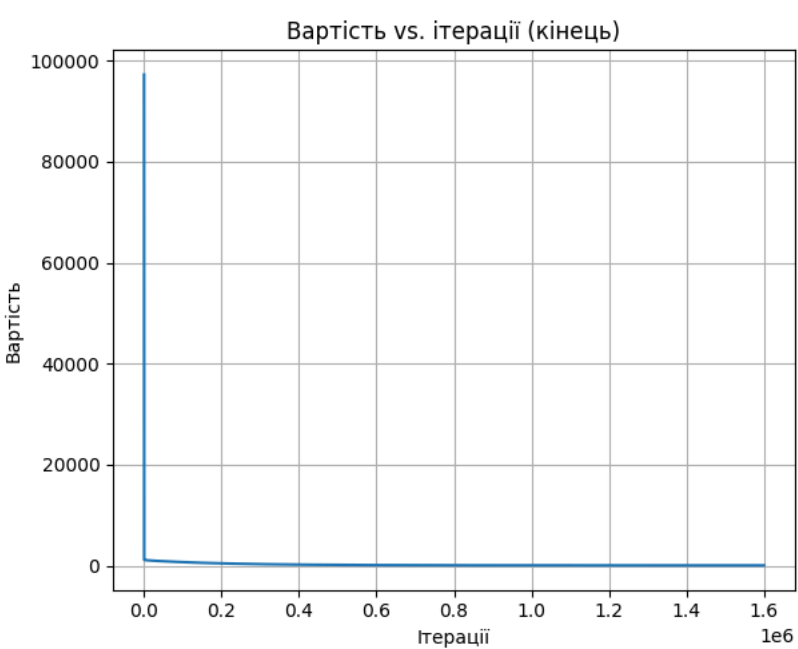
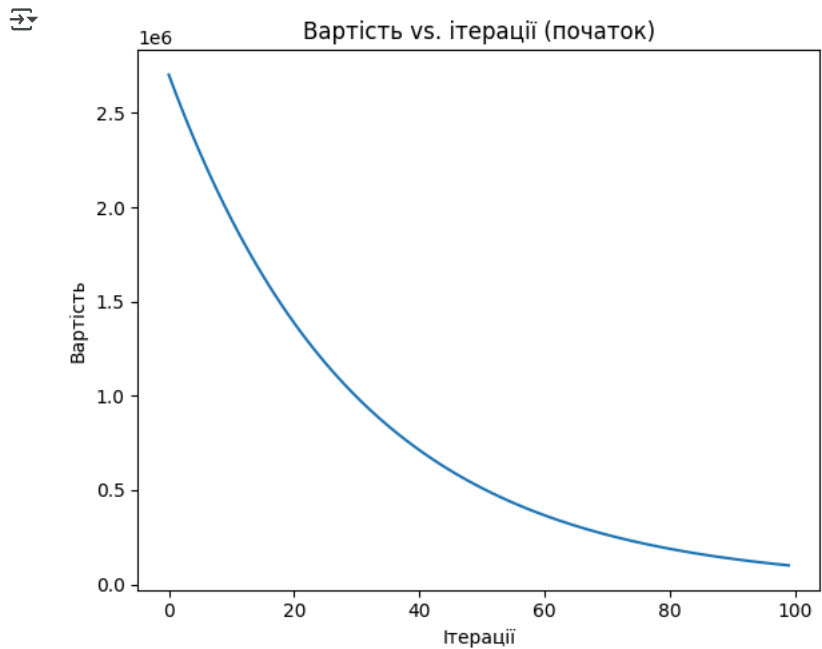
plt.plot(x_line, y_line, 'b-', label='Модель')
```

```
plt.xlabel('Вхідний параметр x')
plt.ylabel('Цільове значення y')
plt.title('Лінійна регресія: дані та модель')
plt.legend()
plt.grid(True)
plt.show()
```



```
# 6.2 Візуалізація зміни функції вартості
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(J_history[:100])
plt.title('Вартість vs. ітерації (початок)')
plt.xlabel('Ітерації')
plt.ylabel('Вартість')

plt.subplot(1, 2, 2)
plt.plot(range(100, len(J_history)), J_history[100:])
plt.title('Вартість vs. ітерації (кінець)')
plt.xlabel('Ітерації')
plt.ylabel('Вартість')
plt.grid(True)
plt.tight_layout()
plt.show()
```



```
# 7. Використання моделі для прогнозування
def predict(x, w1, w2, b):
    """
    Прогнозування за допомогою лінійної моделі

    Аргументи:
    x (скаляр): вхідне значення для прогнозування
    w, b (скаляри): параметри моделі

    Повертає:
    y_pred (скаляр): прогнозоване значення
    """
    return w1 * x + w2 * x*x + b
```

```
# Приклади прогнозування
test_values = [np.min(x_train), np.max(x_train), (np.min(x_train) + np.max(x_train)) / 2]
print("\nПрогнози моделі:")
for x_value in test_values:
    y_pred = predict(x_value, w1, w2, b)
    print(f"При x = {x_value:.2f}, прогноз y = {y_pred:.2f}")

# Додатковий аналіз - коефіцієнт детермінації R²
def r_squared(y_true, y_pred):
    """
    Обчислює коефіцієнт детермінації R²

    Аргументи:
    y_true (ndarray): фактичні значення
    y_pred (ndarray): прогнозовані значення

    Повертає:
    r2 (скаляр): коефіцієнт детермінації
    """
    ss_total = np.sum((y_true - np.mean(y_true))**2)
    ss_residual = np.sum((y_true - y_pred)**2)
    r2 = 1 - (ss_residual / ss_total)
    return r2

# Обчислення прогнозів для навчальних даних
v_pred_train = w1 * x_train + w2 * x_train**2 + b
```

```
# Обчислення R²
r2 = r_squared(y_train, y_pred_train)
print(f"\nКоефіцієнт детермінації (R²): {r2:.4f}")
```

```
↵
Прогнози моделі:
При x = -5.00, прогноз y = 333.93
При x = 35.00, прогноз y = 250.76
При x = 15.00, прогноз y = 171.11

Коефіцієнт детермінації (R²): 0.9515
```