```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import matplotlib.pyplot as plt


# Завантаження даних
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00291/airfoil_self_noise.dat"
column_names = ['Frequency', 'Angle_of_Attack', 'Chord_Length',
                'Free_Stream_Velocity', 'Suction_Side_Displacement_Thickness', 'Sound_Pressure_Level']
df = pd.read_csv(url, sep='\t', names=column_names)

# Вибір ознак і цільової змінної
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

# Нормалізація
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Розділення на тренувальні та тестові дані
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Ініціалізація параметрів для градієнтного спуску
m, n = X_train.shape
w = np.zeros(n)
b = 0
alpha = 0.01
iterations = 1000


# Функція обчислення виходу моделі
def compute_model_output(X, w, b):
    return np.dot(X, w) + b

# Функція обчислення вартості
def compute_cost(X, y, w, b):
    m = len(y)
    f_wb = compute_model_output(X, w, b)
    return np.sum((f_wb - y)**2) / (2 * m)

# Функція обчислення градієнтів
def compute_gradient(X, y, w, b):
    m = len(y)
    error = compute_model_output(X, w, b) - y
    dj_dw = np.dot(X.T, error) / m
    dj_db = np.sum(error) / m
    return dj_dw, dj_db


# Функція градієнтного спуску
def gradient_descent(X, y, w, b, alpha, iterations):
    J_history = []
    for i in range(iterations):
        dj_dw, dj_db = compute_gradient(X, y, w, b)
        w -= alpha * dj_dw
        b -= alpha * dj_db
        if i % 10 == 0:
            J_history.append(compute_cost(X, y, w, b))
    return w, b, J_history

# Навчання моделі за допомогою градієнтного спуску
w_final, b_final, J_hist = gradient_descent(X_train, y_train, w, b, alpha, iterations)

print("Оптимізовані ваги (w_final):", w_final)
print("Оптимізований зсув (b_final):", b_final)
```

```
Оптимізовані ваги (w_final): [-3.96521023 -2.05962944 -3.08659513  1.49968085 -2.09245022]
Оптимізований зсув (b_final): 124.82825891554798
```

```
# Прогнозування на тренувальних та тестових даних
y_pred_train = compute_model_output(X_train, w_final, b_final)
y_pred_test = compute_model_output(X_test, w_final, b_final)

# Оцінка моделі за допомогою метрик
print("\nМетрики на тренувальних даних:")
print("Train R²:", r2_score(y_train, y_pred_train))

print("\nМетрики на тестових даних:")
```

```
print("Test R²:", r2_score(y_test, y_pred_test))
print("Test MSE:", mean_squared_error(y_test, y_pred_test))
print("Test MAE:", mean_absolute_error(y_test, y_pred_test))
```

```
Метрики на тренувальних даних:
Train R²: 0.5027186597799145

Метрики на тестових даних:
Test R²: 0.5566092300110091
Test MSE: 22.213247065298958
Test MAE: 3.673774227019394
```

```
# Візуалізація функції вартості
plt.figure(figsize=(10, 5))
plt.plot(J_hist)
plt.xlabel("Ітерації (×10)")
plt.ylabel("Функція вартості")
plt.title("Збіжність градієнтного спуску")
plt.grid(True)
plt.show()

# Порівняння прогнозів і справжніх значень на тестових даних
plt.figure(figsize=(10, 5))
plt.scatter(y_test, y_pred_test, alpha=0.6)
plt.xlabel("Справжні значення")
plt.ylabel("Прогнозовані значення")
plt.title("Прогноз vs Реальність (Тестові дані)")
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red')  # ідеальна пряма
plt.grid(True)
plt.show()
```