Experiment 3

Aim: To implement gradient descent for regression.

Tools Used: Python

Theory: Gradient descent is an optimization algorithm used to minimize the cost function in regression problems by iteratively adjusting the model's parameters. It calculates the gradient of the cost function with respect to each parameter and updates the parameters in the opposite direction of the gradient to reduce the error. The learning rate controls the step size during this process. The algorithm continues until it converges to the minimum, improving the accuracy of the model with each iteration.

Code:

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

def compute_cost(x, y, w, b):
    m = x.shape[0]
    cost = 0
    for i in range(m):
        f_wb = w * x[i] + b
        cost += (f_wb - y[i]) ** 2
    total_cost = 1 / (2 * m) * cost
    return total_cost

def compute_gradient(x, y, w, b):
    m = x.shape[0]
    dj_dw = 0
    dj_db = 0
    for i in range(m):
        f_wb = w * x[i] + b
        dj_dw_i = (f_wb - y[i]) * x[i]
        dj_db_i = f_wb - y[i]
        dj_dw += dj_dw_i
        dj_db += dj_db_i
    dj_dw /= m
    dj_db /= m
    return dj_dw, dj_db

# Dummy data for x_train and y_train
x_train = np.array([1.0, 2.0, 3.0, 4.0])
y_train = np.array([2.0, 2.5, 3.5, 4.0])

# Plotting gradients
def plot_gradients(x_train, y_train, cost_function, gradien
t_function):
    w_init = 0
    b_init = 0
    iterations = 10000
    alpha = 0.01

    w_final, b_final, J_hist, p_hist =
        gradient_descent(x_train,
        y_train,
        w_init,
        b_init,
        alpha,
        iterations,
        cost_function,
        gradient_function)

    print(f"(w, b) found by gradient descent: {w_final:.4
f}, {b_final:.4f}")

    fig, (ax1, ax2) = plt.subplots(1, 2, constrained_layout
=True, figsize=(12, 4))
    ax1.plot(J_hist[:100])
    ax2.plot(1000 + np.arange(len(J_hist[1000:])), J_hist[1
000:])
    ax1.set_title("Cost vs. iteration (start)")
    ax2.set_title("Cost vs. iteration (end)")
    ax1.set_ylabel('Cost')
    ax2.set_ylabel('Cost')
    ax1.set_xlabel('iteration step')
    ax2.set_xlabel('iteration step')
    plt.show()

def gradient_descent(x,
y,
w_in,
b_in,
alpha,
num_iters,
cost_function,
gradient_function):
    j_history = []
    p_history = []
    w = w_in
    b = b_in

    for i in range(num_iters):
        dj_dw, dj_db = gradient_function(x, y, w, b)
        w = w - alpha * dj_dw
        b = b - alpha * dj_db

        j_history.append(cost_function(x, y, w, b))
        p_history.append([w, b])

        if i % math.ceil(num_iters/10) == 0:
            print(f"Iteration {i:4}: Cost {j_history[-1]:0.
2e} ",
                  f"dj_dw: {dj_dw: 0.3e}, dj_db: {dj_db: 0.
3e} ",
                  f"w: {w: 0.3e}, b: {b: 0.5e}")

    return w, b, j_history, p_history

# Initialize parameters
w_init = 0
b_init = 0
iterations = 10000
alpha = 1.0e-2
```

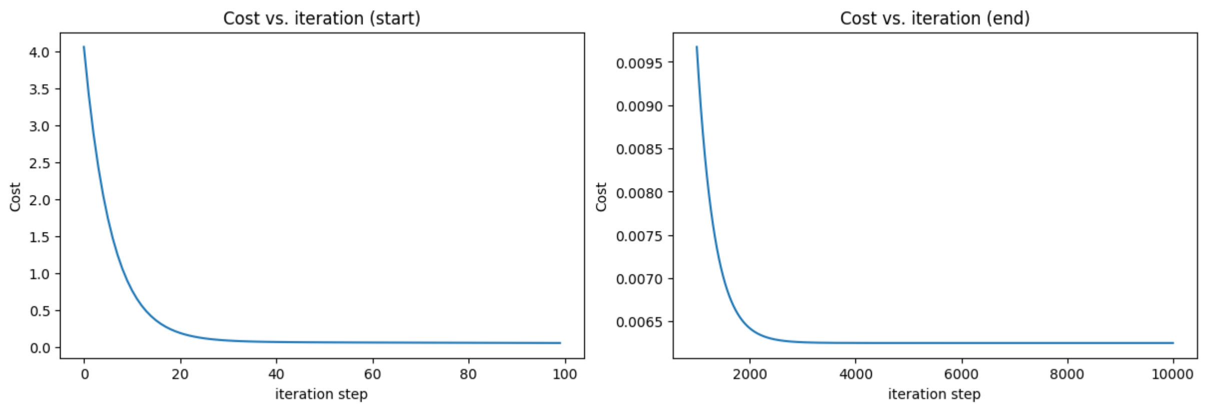
```
# Perform gradient descent and plot the results
w_final, b_final, J_hist, p_hist = gradient_descent(x_trai
n,
y_train,
w_init,
b_init,
alpha,
iterations,
compute_cost,
compute_gradient)

# Plot the cost vs iteration steps
fig, (ax1, ax2) = plt.subplots(1, 2, constrained_layout=Tru
e, figsize=(12,4))
ax1.plot(J_hist[:100])
ax2.plot(1000 + np.arange(len(J_hist[1000:])), J_hist[100
0:])
ax1.set_title("Cost vs. iteration (start)")
ax2.set_title("Cost vs. iteration (end)")
ax1.set_ylabel('Cost')
ax2.set_ylabel('Cost')
ax1.set_xlabel('iteration step')
ax2.set_xlabel('iteration step')
plt.show()
```

Output:

```
Iteration    0: Cost 4.05e+00  dj_dw: -8.375e+00, dj_db: -
3.000e+00  w:  8.375e-02, b:  3.00000e-02
Iteration 1000: Cost 9.67e-03  dj_dw:  1.032e-02, dj_db: -
3.035e-02  w:  7.689e-01, b:  1.04754e+00
Iteration 2000: Cost 6.42e-03  dj_dw:  2.308e-03, dj_db: -
6.786e-03  w:  7.154e-01, b:  1.20474e+00
Iteration 3000: Cost 6.26e-03  dj_dw:  5.160e-04, dj_db: -
1.517e-03  w:  7.034e-01, b:  1.23988e+00
Iteration 4000: Cost 6.25e-03  dj_dw:  1.154e-04, dj_db: -
3.391e-04  w:  7.008e-01, b:  1.24774e+00
```

```
Iteration 5000: Cost 6.25e-03  dj_dw:  2.579e-05, dj_db: -
7.582e-05  w:  7.002e-01, b:  1.24949e+00
Iteration 6000: Cost 6.25e-03  dj_dw:  5.765e-06, dj_db: -
1.695e-05  w:  7.000e-01, b:  1.24989e+00
Iteration 7000: Cost 6.25e-03  dj_dw:  1.289e-06, dj_db: -
3.789e-06  w:  7.000e-01, b:  1.24997e+00
Iteration 8000: Cost 6.25e-03  dj_dw:  2.881e-07, dj_db: -
8.472e-07  w:  7.000e-01, b:  1.24999e+00
Iteration 9000: Cost 6.25e-03  dj_dw:  6.442e-08, dj_db: -
1.894e-07  w:  7.000e-01, b:  1.25000e+00
```



Result: Gradient descent for regression has been successfully implemented.

Criteria	Total Marks	Marks Obtained	Comments
Concept (A)			
Implementation (B)			
Performance (C)			
Total			