Assignment 1 – Linear Regression with Gradient Descent

Name: Raj Patel
Student ID: 801334777
Assignment: Assignment 1

GitHub Repository:
https://github.com/RajP-17/Intro-To-ML-Assignments

Use of Gen AI: The author acknowledges the use of Chat gpt and Gemini in the preparation or completion of this assignment. The Chat GPT and Gemini were used in the following way's in this assignment: brainstorming, grammatical correction, citation, compiler errors, optimisation tips, and proper syntax.

---

1. Introduction

In this assignment, we implemented linear regression using the gradient descent algorithm from scratch—without using built-in functions from libraries like scikit-learn. We used the dataset "D3.csv" where the first three columns represent the explanatory variables (x_1, x_2, x_3) and the fourth column represents the dependent variable (y). Two main problems were addressed:

- **Problem 1:** Training a separate linear regression model for each explanatory variable individually.
- **Problem 2:** Training a multivariate linear regression model using all three explanatory variables.

The GitHub repository containing the source code can be found at the URL listed above.

---

2. Problem 1: Single-Feature Linear Regression

2.1 Methodology

For each feature (x_1, x_2, and x_3), we implemented gradient descent to minimize the cost function:

$$J(\theta_0, \theta_1) = (1/(2m)) \Sigma (\theta_0 + \theta_1 \cdot x_i - y_i)^2$$

We initialized both parameters (θ_0 and θ_1) to zero and experimented with three different learning rates (0.1, 0.05, and 0.01). For each feature, the learning rate yielding the lowest final cost was chosen. In our experiments, a learning rate of 0.1 provided the best performance for all features.

2.2 Source Code

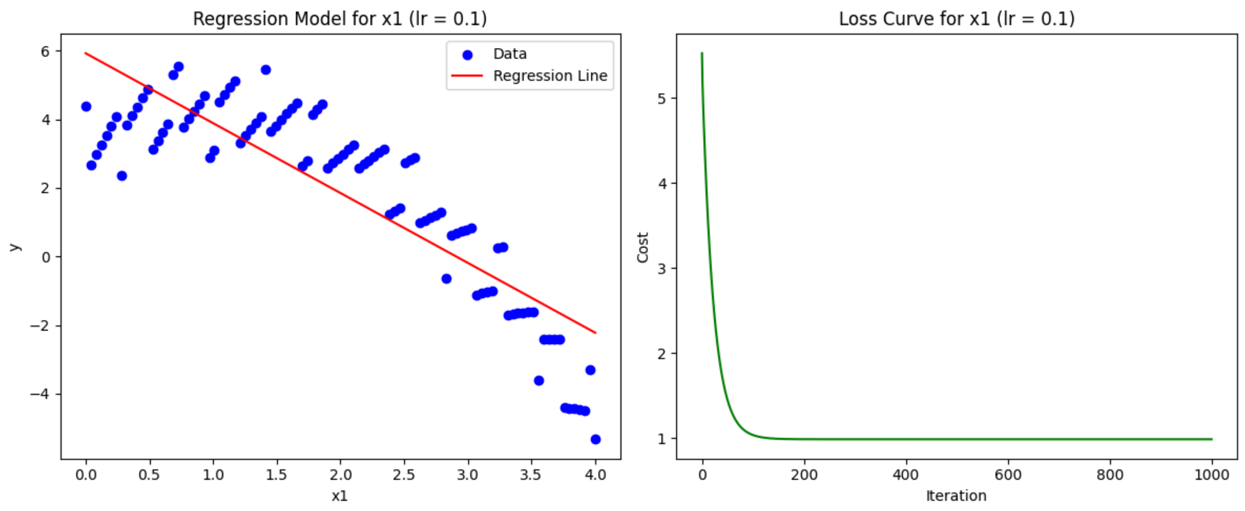
Below is the key portion of the Python code used for Problem 1:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Load dataset from D3.csv
data = pd.read_csv('D3.csv')
x1 = data.iloc[:, 0].values
x2 = data.iloc[:, 1].values
x3 = data.iloc[:, 2].values
y  = data.iloc[:, 3].values

def gradient_descent_single(x, y, lr=0.1, iterations=1000):
    m = len(y)
    theta0, theta1 = 0.0, 0.0
    loss_history = []
    for i in range(iterations):
        predictions = theta0 + theta1 * x
        error = predictions - y
        cost = (1/(2*m)) * np.sum(error ** 2)
        loss_history.append(cost)
        grad0 = (1/m) * np.sum(error)
        grad1 = (1/m) * np.sum(error * x)
        theta0 = theta0 - lr * grad0
        theta1 = theta1 - lr * grad1
    return theta0, theta1, loss_history

learning_rates = [0.1, 0.05, 0.01]
iterations = 1000
features = [(x1, 'x1'), (x2, 'x2'), (x3, 'x3')]
models = {}

for feature, name in features:
    best_theta0, best_theta1 = None, None
    best_loss = float('inf')
    best_lr = None
    best_loss_history = None
    for lr in learning_rates:
        theta0, theta1, loss_history =
gradient_descent_single(feature, y, lr=lr, iterations=iterations)
        if loss_history[-1] < best_loss:
            best_loss = loss_history[-1]
            best_theta0 = theta0
            best_theta1 = theta1
            best_lr = lr
            best_loss_history = loss_history
    models[name] = {'theta0': best_theta0, 'theta1': best_theta1,
'lr': best_lr, 'loss_history': best_loss_history}

    # Plotting code (regression line and loss curve) is included in
the notebook.
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.scatter(feature, y, color='blue', label='Data')
    x_line = np.linspace(feature.min(), feature.max(), 100)
    y_line = best_theta0 + best_theta1 * x_line
    plt.plot(x_line, y_line, color='red', label='Regression Line')
    plt.xlabel(name)
    plt.ylabel('y')
    plt.title(f'Regression Model for {name} (lr = {best_lr})')
    plt.legend()
    plt.subplot(1, 2, 2)
    plt.plot(best_loss_history, color='green')
    plt.xlabel('Iteration')
    plt.ylabel('Cost')
    plt.title(f'Loss Curve for {name} (lr = {best_lr})')
    plt.tight_layout()
    plt.show()

    print(f"For {name}: y = {best_theta0:.4f} + {best_theta1:.4f} *
{name} with final cost = {best_loss:.4f} (lr = {best_lr})")
```

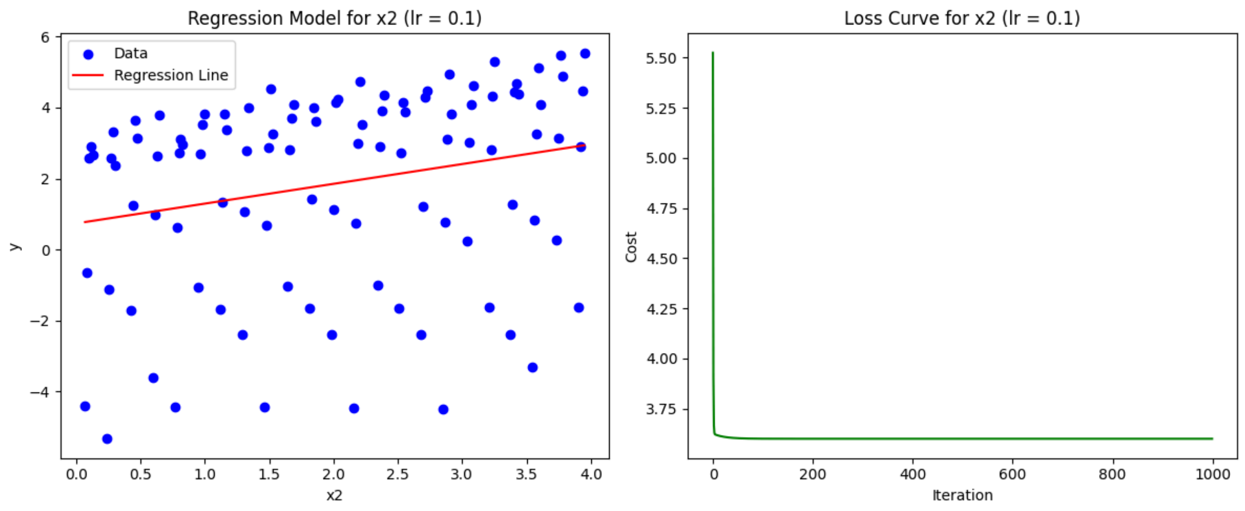
2.3 Results and Plots

The experiments yielded the following models:

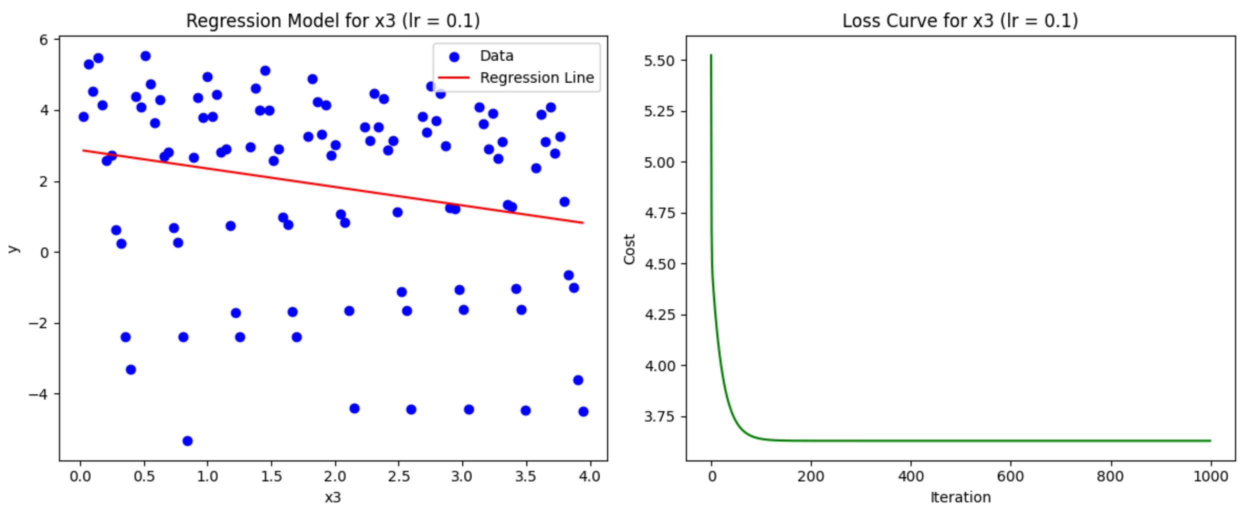
- **x_1 Model:**
  $y = 5.9279 - 2.0383 \cdot x_1$ (Final cost = 0.9850, Learning rate = 0.1)



- **x_2 Model:**
  $y = 0.7361 + 0.5576 \cdot x_2$ (Final cost = 3.5994, Learning rate = 0.1)



- **x_3 Model:**
  $y = 2.8714 - 0.5205 \cdot x_3$ (Final cost = 3.6295, Learning rate = 0.1)



2.4 Observations and Answers

- The model using x_1 as the predictor achieved the lowest final cost (0.9850), indicating that x_1 is the strongest single predictor of y.
- The loss curves demonstrated a steady decrease in cost over iterations with a learning rate of 0.1.
- A higher learning rate (0.1) resulted in rapid convergence for these models.

---

3. Problem 2: Multi-Feature Linear Regression

3.1 Methodology

For the multi-feature model, we combined x_1, x_2, and x_3 into a single design matrix with an additional bias column. The cost function used was:

$$J(\theta) = (1/(2m)) \Sigma (\theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + \theta_3 \cdot x_3 - y)^2$$

Gradient descent was applied to update all parameters simultaneously. As before, different learning rates were tested, and a learning rate of 0.1 produced the best results.

3.2 Source Code

Below is the key code excerpt for the multi-feature model:

```
def gradient_descent_multi(X, y, lr=0.1, iterations=1000):
    m, n = X.shape
    theta = np.zeros(n)
    loss_history = []
    for i in range(iterations):
        predictions = X.dot(theta)
        error = predictions - y
        cost = (1/(2*m)) * np.sum(error ** 2)
        loss_history.append(cost)
        gradient = (1/m) * (X.T.dot(error))
        theta = theta - lr * gradient
    return theta, loss_history

X_multi = np.column_stack((np.ones(len(x1)), x1, x2, x3))
best_theta = None
best_loss = float('inf')
best_lr = None
best_loss_history = None

for lr in learning_rates:
    theta, loss_history = gradient_descent_multi(X_multi, y, lr=lr,
iterations=iterations)
    if loss_history[-1] < best_loss:
        best_loss = loss_history[-1]
        best_theta = theta
        best_lr = lr
        best_loss_history = loss_history

print(f"Final multi-feature model: y = {best_theta[0]:.4f} +
{best_theta[1]:.4f}*x1 + {best_theta[2]:.4f}*x2 +
{best_theta[3]:.4f}*x3")
print(f"(Learning rate: {best_lr}, Final cost: {best_loss:.4f})")

plt.figure(figsize=(6, 5))
plt.plot(best_loss_history, color='purple')
plt.xlabel('Iteration')
plt.ylabel('Cost')
plt.title(f'Multi-Feature Loss Curve (lr = {best_lr})')
plt.show()

new_inputs = np.array([[1, 1, 1],
                       [2, 0, 4],
                       [3, 2, 1]])
new_inputs_bias = np.column_stack((np.ones(len(new_inputs)),
new_inputs))
predictions = new_inputs_bias.dot(best_theta)

print("\nPredictions for new (x1, x2, x3) values:")
for inp, pred in zip(new_inputs, predictions):
    print(f"Input {inp} -> Predicted y = {pred:.4f}")
```

3.3 Results, Predictions, and Plots

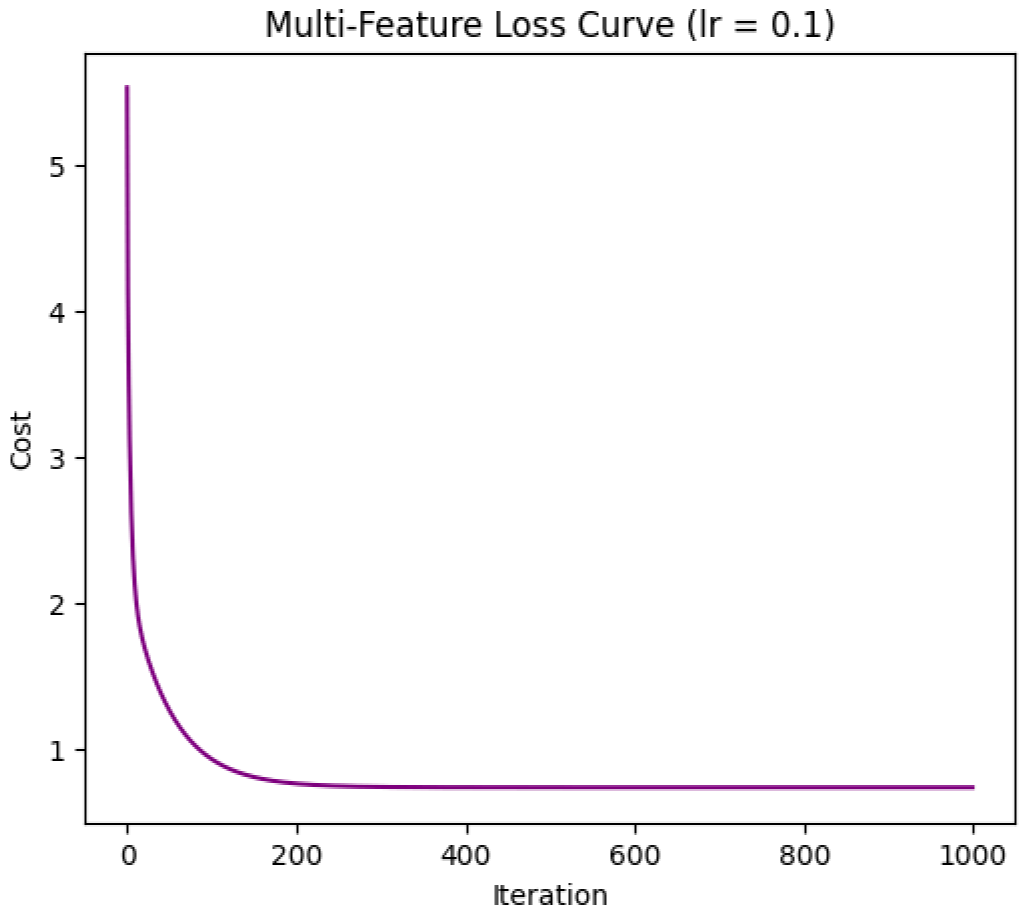
The final multi-feature model is:

$y = 5.3139 – 2.0037 \cdot x_1 + 0.5326 \cdot x_2 – 0.2656 \cdot x_3$

(Final cost = 0.7385, Learning rate = 0.1)

The loss curve for the multi-feature model shows a steady decline in cost over the iterations. Predictions on new input data are as follows:

- For input [1, 1, 1] → Predicted y = 3.5773
- For input [2, 0, 4] → Predicted y = 0.2443
- For input [3, 2, 1] → Predicted y = 0.1025



3.4 Observations and Answers

- The multi-feature model achieved a lower final cost (0.7385) compared to any of the individual models, indicating a better overall fit.
- The loss curve confirms steady convergence with the chosen learning rate of 0.1.
- The predictions on new data illustrate that the model generalizes well.

---

4. Conclusions

- **Problem 1:**
  Separate models for each explanatory variable were developed. The x_1 model ($y = 5.9279 – 2.0383 \cdot x_1$) provided the best fit with the lowest cost, suggesting that x_1 is the strongest single predictor of y.
- **Problem 2:**
  The combined multi-feature regression model yielded an improved prediction performance (final cost 0.7385) compared to individual models. The loss curves and

prediction outputs confirm that using all three explanatory variables offers a more accurate overall model.
- The experiments highlighted the importance of selecting an appropriate learning rate, where 0.1 provided effective and stable convergence.

---

5. Assumptions

- The dataset “D3.csv” is correctly formatted with four columns.
- A fixed number of iterations (1000) was used for gradient descent; this parameter can be adjusted if necessary.
- The learning rates tested (0.1, 0.05, and 0.01) are considered representative. In these experiments, a learning rate of 0.1 yielded the best performance.
- All computations were performed on the entire dataset without splitting into separate training and testing sets.
