Practical-6: Develop a Gradient descent of linear regression using sample dataset.

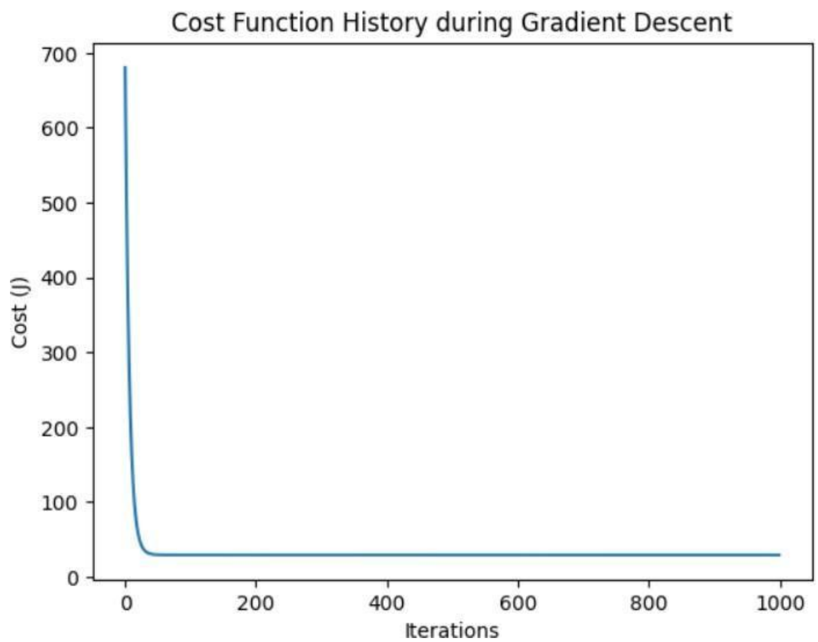
Output:

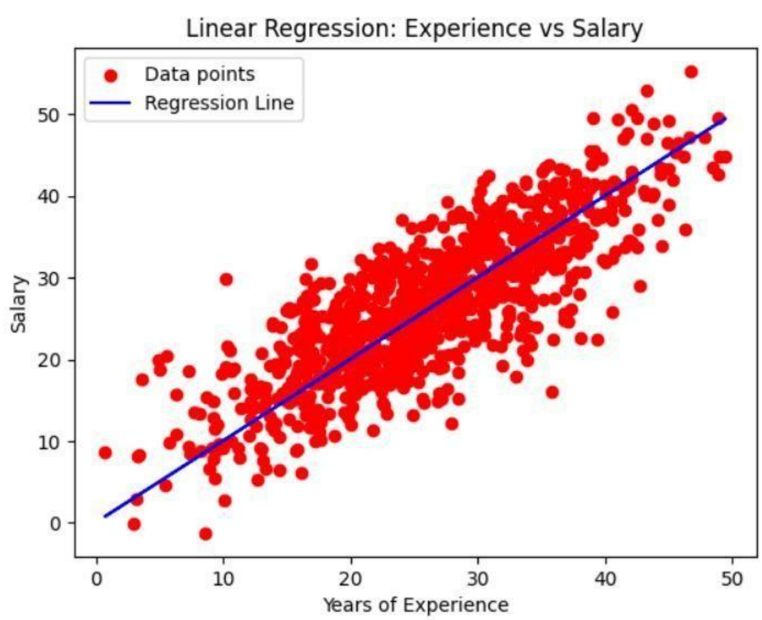
```
Initial cost (theta = zeros): 792.2487672102134
```

```
Iteration 1: Cost = 680.5572602312128
Iteration 2: Cost = 585.2192664846582
Iteration 3: Cost = 503.84035572724207
Iteration 4: Cost = 434.3766827006044
Iteration 5: Cost = 375.08365557468636
Iteration 6: Cost = 324.4721201970103
Iteration 7: Cost = 281.2709597070975
Iteration 8: Cost = 244.3951701983308
Iteration 9: Cost = 212.91861064152818
Iteration 10: Cost = 186.05074267945125
```

```
Optimized parameters: theta_0 = 0.08569249990135377, theta_1 = 0.9988176396752504
```

```
Final cost after optimization: 29.364303963905186
```


Cost Function History during Gradient Descent
Cost (J)
Iterations
0
100
200
300
400
500
600
700
0
200
400
600
800
1000


Linear Regression: Experience vs Salary
Data points
Regression Line
Salary
Years of Experience
0
10
20
30
40
50

Code:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

 # 1. Load dataset

df= pd.read_csv("Experience-Salary.csv")

 # Step 2: Extract input (x) and output (y)

data x = data['exp(in months)'].values

y = data['salary(in thousands)'].values

m = len(x) **# Number of data points**

 # Step 3: Prepare the data (add column of ones for intercept

term)

X = np.c_[np.ones(m), x] **# Shape: (m, 2)**

 # Step 4: Initialize parameters (theta_0 and

theta_1) theta = np.zeros(2) **# [theta_0, theta_1]**

 # Step 5: Learning rate and number of iterations for gradient descent

learning_rate = 0.0001

iterations = 1000

 # Step 6: Define the cost function (Mean Squared Error)

def cost_function(X, y, theta):

    predictions = X.dot(theta)

    cost = (1 / m) * np.sum((predictions - y) ** 2)

    return cost

 # Step 7: Gradient descent function to optimize theta

```
def gradient_descent(X, y, theta, learning_rate, iterations):
    cost_history = []
  for i in range(iterations):
        predictions = X.dot(theta)
        errors = predictions - y
        gradient = (1 / m) * X.T.dot(errors)
       theta -= learning_rate * gradient
       current_cost = cost_function(X, y, theta)
       cost_history.append(current_cost)
       # Optional: print cost at each iteration
        print(f"Iteration {i+1}: Cost = {current_cost}")
    return theta, cost_history
# Step 8: Print initial cost before training
initial_cost = cost_function(X, y, theta)
print(f"Initial cost (theta = zeros): {initial_cost}")

# Step 9: Run gradient descent
theta_optimized, cost_history = gradient_descent(X, y, theta, learning_rate, iterations)
# Step 10: Output optimized parameters
print(f"Optimized parameters: theta_0 = {theta_optimized[0]}, theta_1 = {theta_optimized[1]}")
# Step 11: Print final cost
final_cost = cost_function(X, y, theta_optimized)
print(f"Final cost after optimization: {final_cost}")
```

 # Step 12: Plot cost function history

```
plt.plot(range(iterations), cost_history)

plt.xlabel('Iterations')

plt.ylabel('Cost (J)')

plt.title('Cost Function History during Gradient Descent') plt.show()
```

 # Step 13: Plot data points and regression line

```
plt.scatter(x, y, color='red', label='Data points')

plt.plot(x, X.dot(theta_optimized), label='Regression Line', color='blue')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.title('Linear Regression: Experience vs Salary')

plt.legend()

plt.show()
```