# LP3 Group B Assignment 4

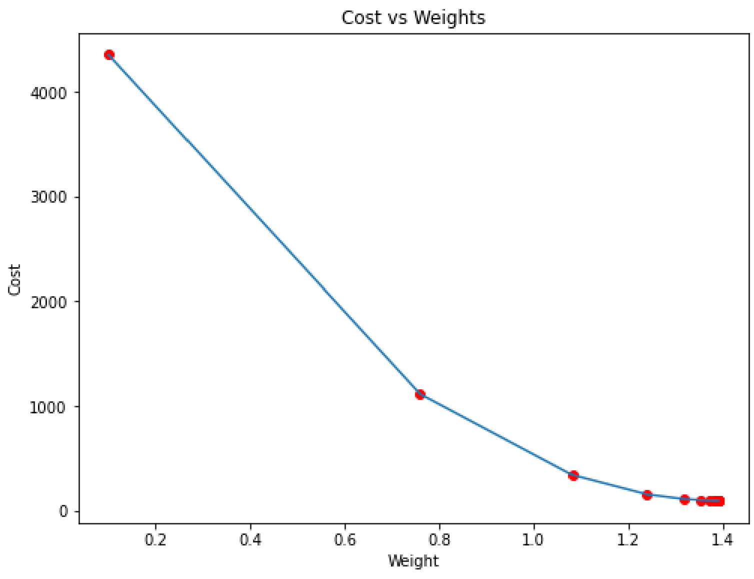
## Implement Gradient Descent Algorithm to find the local minima of a function.

### For example, find the local minima of the function $y=(x+3)^2$ starting from the point x=2 .

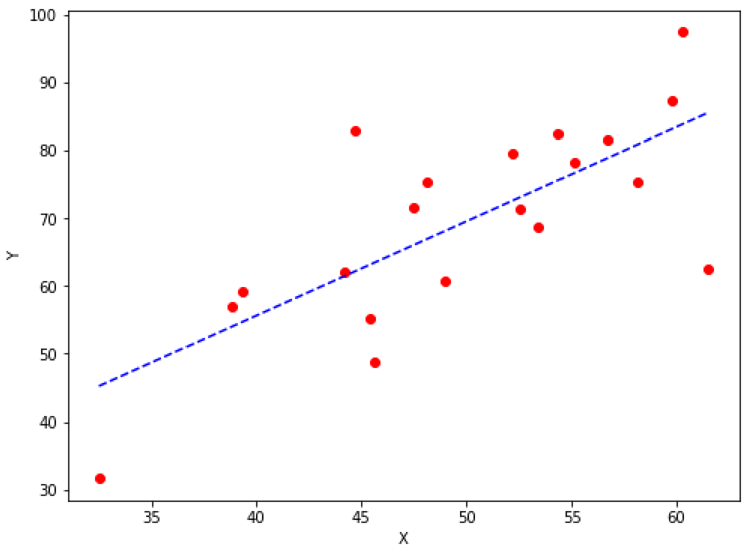
In [1]:

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

def mean_squared_error(y_true, y_predicted):
    cost = np.sum((y_true-y_predicted)**2) / len(y_true) #Calculating the loss or cost
    return cost

def gradient_descent(x, y, iterations = 1000, learning_rate = 0.0001, stopping_threshold = 1e-6): #Gradient Descent
    #Initializing weight, bias, learning rate and iterations
    current_weight = 0.1
    current_bias = 0.01
    iterations = iterations
    learning_rate = learning_rate
    n = float(len(x))
    costs = []
    weights = []
    previous_cost = None

    for i in range(iterations): #Estimation of optimal parameters
        y_predicted = (current_weight * x) + current_bias #Making predictions
        current_cost = mean_squared_error(y, y_predicted) #Calculationg the current cost
        #If the change in cost is less than or equal to stopping_threshold we stop the gradient descent
        if previous_cost and abs(previous_cost-current_cost)<=stopping_threshold:
            break
        previous_cost = current_cost
        costs.append(current_cost)
        weights.append(current_weight)
        #Calculating the gradients
        weight_derivative = -(2/n) * sum(x * (y-y_predicted))
        bias_derivative = -(2/n) * sum(y-y_predicted)
        #Updating weights and bias
        current_weight = current_weight - (learning_rate * weight_derivative)
        current_bias = current_bias - (learning_rate * bias_derivative)
        #Printing the parameters for each 1000th iteration
        print(f"Iteration{i+1}:Cost{current_cost},Weight\{current_weight},Bias{current_bias}")
    #Visualizing the weights and cost at for all iterations
    plt.figure(figsize = (8,6))
    plt.plot(weights, costs)
    plt.scatter(weights, costs, marker='o', color='red')
    plt.title("Cost vs Weights")
    plt.ylabel("Cost")
    plt.xlabel("Weight")
    plt.show()
    return current_weight, current_bias

#Data
X = np.array([32.50234527, 53.42680403, 61.53035803, 47.47563963, 59.81320787,
          55.14218841, 52.21179669, 39.29956669, 48.10504169, 52.55001444,
          45.41973014, 54.35163488, 44.1640495 , 58.16847072, 56.72720806,
          48.95588857, 44.68719623, 60.29732685, 45.61864377, 38.81681754])
Y = np.array([31.70700585, 68.77759598, 62.5623823 , 71.54663223, 87.23092513,
          78.21151827, 79.64197305, 59.17148932, 75.3312423 , 71.30087989,
          55.16567715, 82.47884676, 62.00892325, 75.39287043, 81.43619216,
          60.72360244, 82.89250373, 97.37989686, 48.84715332, 56.87721319])

#Estimating weight and bias using gradient descent
estimated_weight, eatimated_bias = gradient_descent(X, Y, iterations=2000)
print(f"Estimated Weight: {estimated_weight}\nEstimated Bias: {eatimated_bias}")
Y_pred = estimated_weight*X + eatimated_bias #Making predictions using estimated parameters
#Plotting the regression line
plt.figure(figsize = (8,6))
plt.scatter(X, Y, marker='o', color='red')
plt.plot([min(X), max(X)], [min(Y_pred), max(Y_pred)], color='blue',markerfacecolor='red',markersize=10,linestyle='
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```

```
Iteration1:Cost4352.088931274409,Weight\0.7593291142562117,Bias0.02288558130709
Iteration2:Cost1114.8561474350017,Weight\1.081602958862324,Bias0.029180147485695113
Iteration3:Cost341.42912086804455,Weight\1.2391274084945083,Bias0.03225308846928192
Iteration4:Cost156.64495290904443,Weight\1.3161239281746984,Bias0.03375132986012604
Iteration5:Cost112.49704004742098,Weight\1.3537591652024805,Bias0.034479873154934775
Iteration6:Cost101.9493925395456,Weight\1.3721549833978113,Bias0.034832195392868505
Iteration7:Cost99.4293893333546,Weight\1.3811467575154601,Bias0.03500062439068245
Iteration8:Cost98.82731958262897,Weight\1.3855419247507244,Bias0.03507916814736111
Iteration9:Cost98.68347500997261,Weight\1.3876903144657764,Bias0.035113776874486774
Iteration10:Cost98.64910780902792,Weight\1.3887405007983562,Bias0.035126910596389935
Iteration11:Cost98.64089651459352,Weight\1.389253895811451,Bias0.03512954755833985
Iteration12:Cost98.63893428729509,Weight\1.38950491235671,Bias0.035127053821718185
Iteration13:Cost98.63846506273883,Weight\1.3896276808137857,Bias0.035122052266051224
Iteration14:Cost98.63835254057648,Weight\1.38968776283053,Bias0.03511582492978764
Iteration15:Cost98.63832524036214,Weight\1.3897172043139192,Bias0.03510899846107016
Iteration16:Cost98.63831830104695,Weight\1.389731668997059,Bias0.035101879159522745
Iteration17:Cost98.63831622628217,Weight\1.389738813163012,Bias0.03509461674147458
```



```
Estimated Weight: 1.389738813163012
Estimated Bias: 0.03509461674147458
```



```
In [ ]:
```