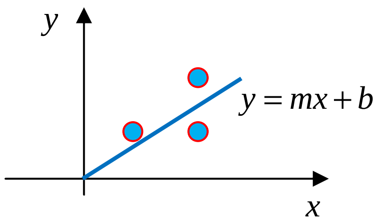
Örnek Uygulama:

Lineer Regresyon probleminin gradyan iniş yöntemi ile çözümü:

$y = mx + b$ doğru denklemi yandaki dataya fit ederek sistemin bu veri aralığı için doğrusal bir modelini elde etmek istiyoruz.



Bunu $T = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), ..., (x_N, y_N)\}$ veri kümesi üzerinde ortalama karesel hatayı minimize eden bir optimizasyon problemi olarak ifade edelim:
Veri başına hatayı yazalım:

$$e_1 = y_1 - y = y_1 - (mx_1 + b)$$
$$e_2 = y_2 - y = y_2 - (mx_2 + b)$$
...
..
$$e_N = y_N - y = y_N - (mx_N + b)$$

Bütün verileri dikkate alabilen ortalama karesel hatayı yazalım.

$$E(m,b) = \frac{1}{N}(e_1^{\ 2} + e_2^{\ 2} + e_3^{\ 2} + .. + e_N^{\ 2}) = \frac{1}{N}\sum_{i=1}^{N} e_i^{\ 2} = \frac{1}{N}\sum_{i=1}^{N}(y_i - (mx_i + b))^2$$

[$e_i = y_i - y = y_i - (mx_i + b)$, ayni x_i noktası için verinin y_i değeri ile doğrusal modelin aynı noktadaki değeri ($y = mx_i + b$) farkını veri başına hata olarak tanımladık.]

$$E(m,b) = \frac{1}{N}\sum_{i=1}^{N}(y_i - (mx_i + b))^2$$

Burada ortalama karesel hata konveks bir aram yüzeyi ifade eder. Gradyan iniş konveks arama uzaylarında oldukça etkindir.

Burada optimizasyon problemi, bütün veriler için en az hata ile fit eden bir lineer model oluşturulması:

$$\min E(m,b) = \frac{1}{N}\sum_{i=1}^{N}(y_i - (mx_i + b))^2$$

Burada (x_i, y_i) veri değerleri ve reel sayıdır. m ve b optimize edilmesi gereken öğrenme parametreleridir.
[Bu problemde kısıt tanımlamaya ihtiyaç duyulmamıştır. Bu optimizasyon problemi, için batch-mode gradiyan iniş ile çözülmesi daha sağlıklı olur. Çünkü bütün verilere modelin en iyi şekilde fit etmesini istiyoruz.] Buna göre parametrelerinin (m,b) gradyan iniş rekürsif çözümü için,

$$m_n = m_{n-1} - \eta\frac{dE}{dm}$$
$$b_n = b_{n-1} - \eta\frac{dE}{db}$$

yazılır. Bu çözümü gradyan iniş ile çözebilmek için $\frac{dE}{dm}$ ve $\frac{dE}{db}$ türevlerini, şöyle elde ederiz.

[Burada türev operatörü toplam sembolü içine girebilir. $\left(u(x)^n\right)' = n.u'.u(x)^{n-1}$ türev özelliği kullanılır. Diğer bir ifade ile $\left(u^2\right)' = n.u'.u$]

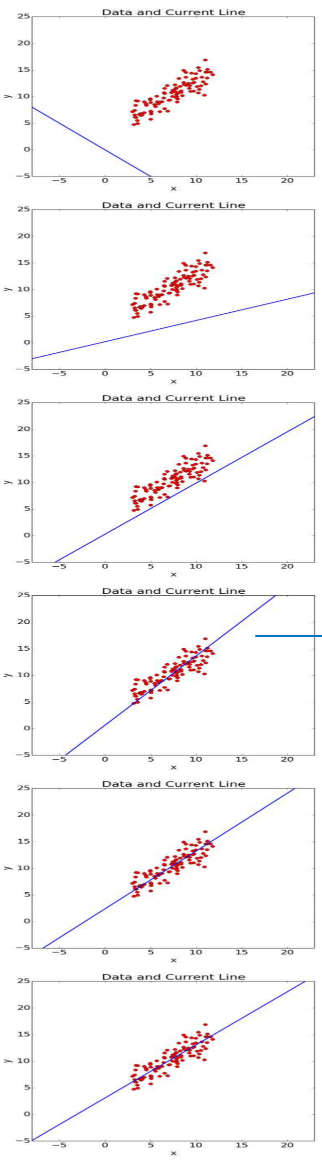
$$\frac{dE(m,b)}{dm} = \frac{1}{N}\sum_{i=1}^{N} 2(-x_i)(y_i - (mx_i + b)) = -\frac{2}{N}\sum_{i=1}^{N} x_i(y_i - (mx_i + b))$$

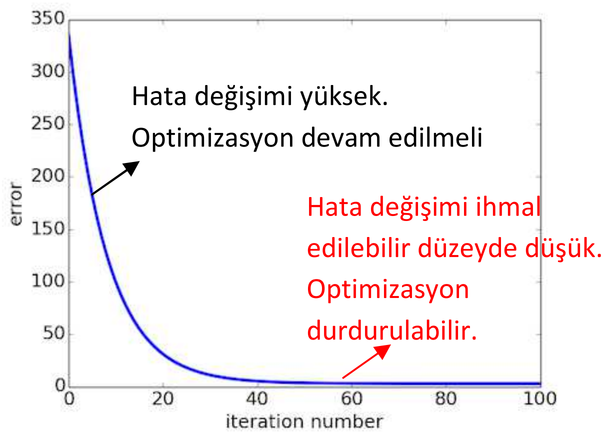
$$\frac{dE(m,b)}{db} = \frac{1}{N}\sum_{i=1}^{N} 2(-1)(y_i - (mx_i + b)) = -\frac{2}{N}\sum_{i=1}^{N} (y_i - (mx_i + b))$$

parametrelerin gradyan iniş güncelleme çözümleri

$$\boxed{\begin{aligned} m_n &= m_{n-1} - \eta(-\frac{2}{N}\sum_{i=1}^{N} x_i(y_i - (mx_i + b))) \\ b_n &= b_{n-1} - \eta(-\frac{2}{N}\sum_{i=1}^{N} (y_i - (mx_i + b))) \end{aligned}}$$

elde edilir. Bu iteratif çözüm bilgisayarda hesaplanmış ve aşağıdaki çözümler elde edilmiştir. Solda





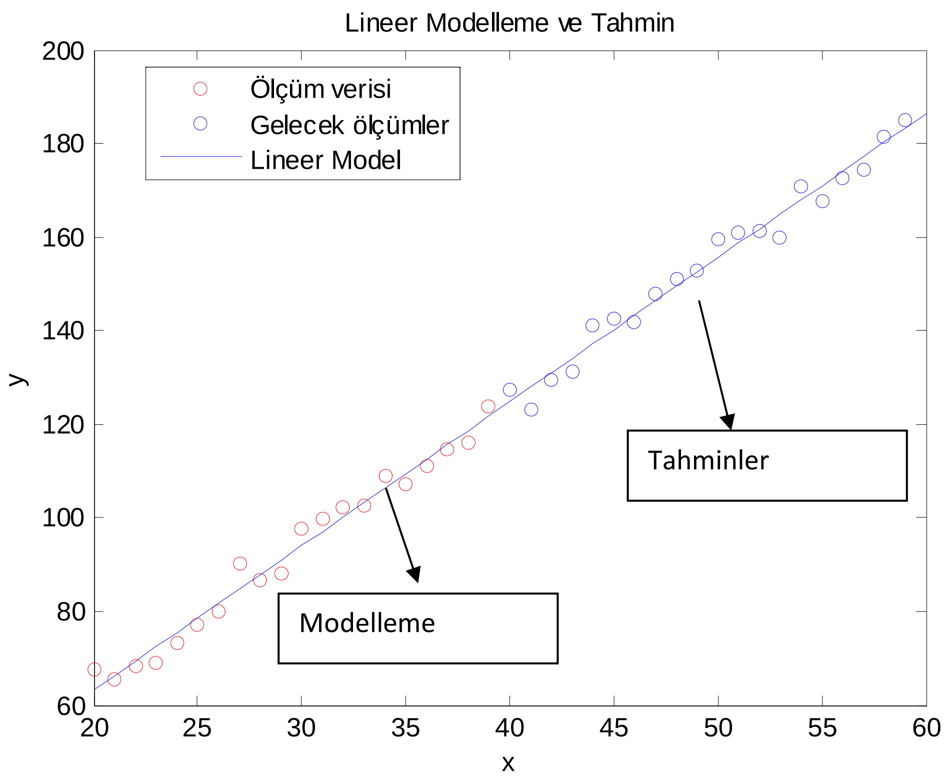
İterasyon süresince, m_n ve b_n çözümlerin ile elde edilmiş olan doğru ve verinin durumları görünüyor. İlk karede başlangıçta doğru denklemi veriden çok uzakta ve hata değeri üst grafikte 300 üzerinde. Ancak, iterasyonlar yapıldıkça, hatayı azaltan m_n ve b_n çözümlerinin ifade ettiği doğru modelinin veriye daha çok yaklaştırdığı görülür.

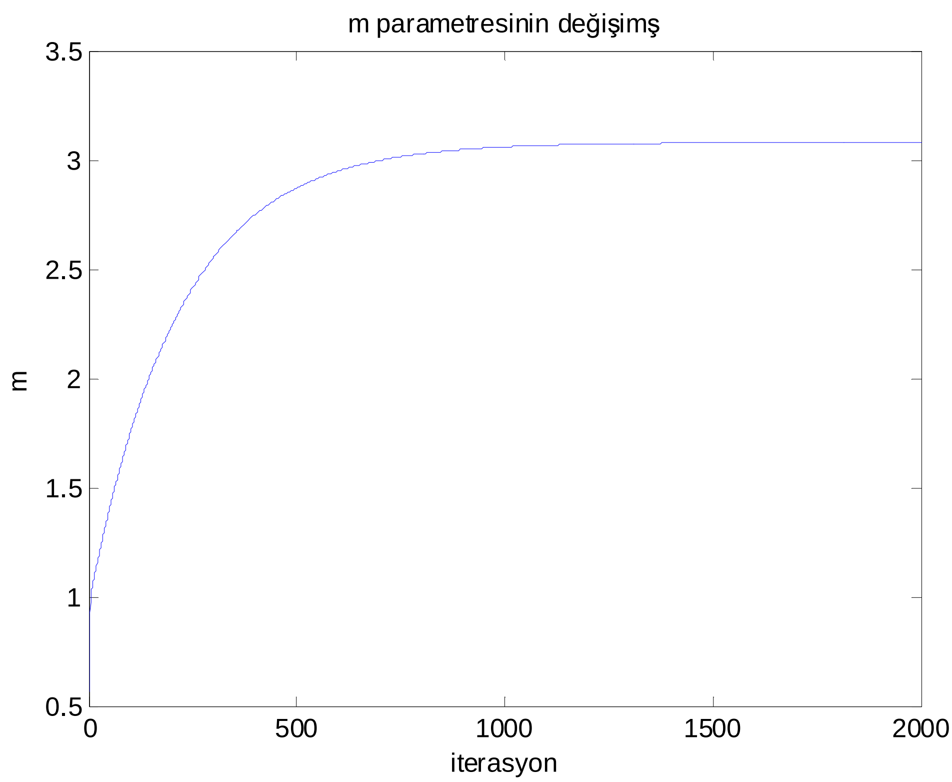
Üstte hatanın iterasyon boyunca değişimine bakıldığında sıfıra yakınsadıkça hatanın değişim hızının azaldığı görülür.

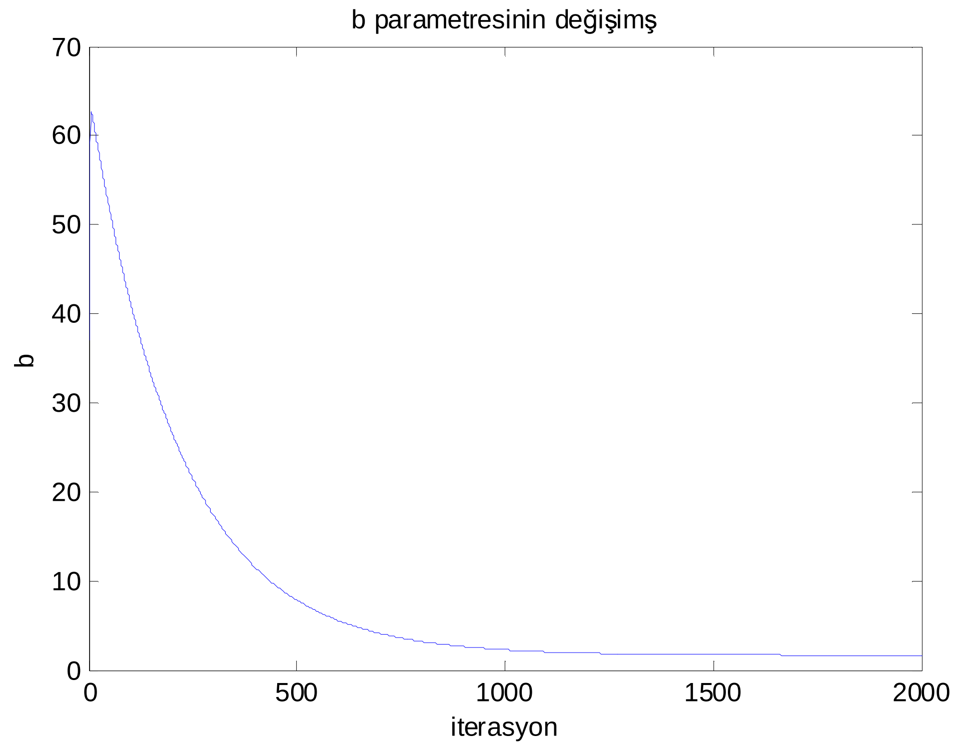
Lineer Regresyonun Ölçüm Verisinden Lineer Modelleme ve Tahmin Amaçlı Uygulaması:

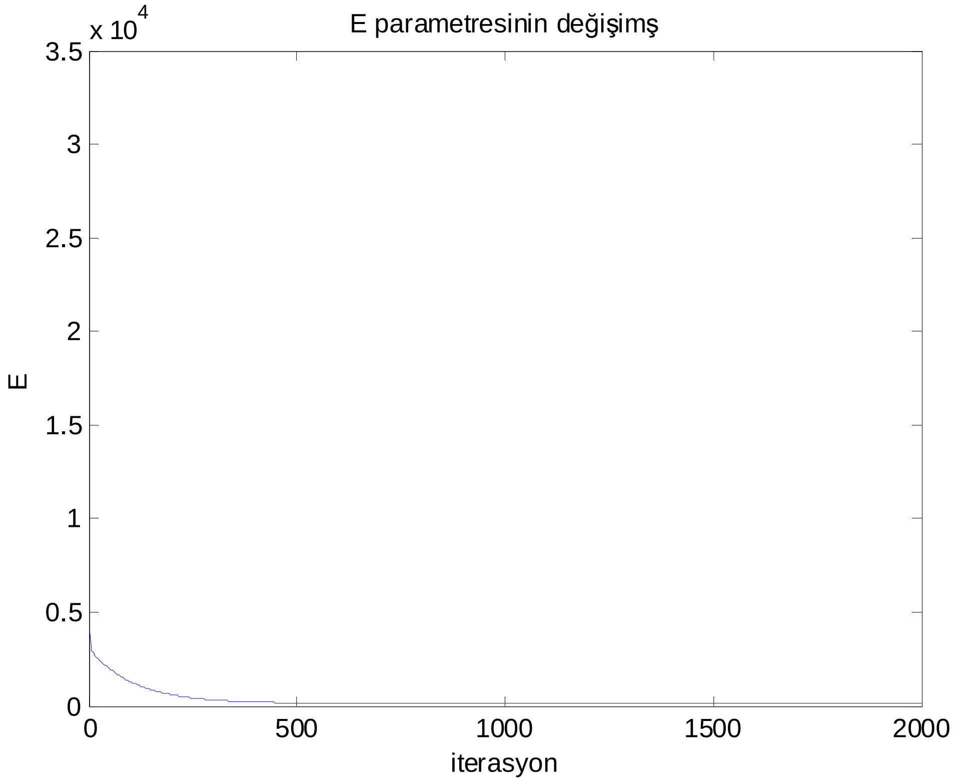
Modelleme, gelecek verileri tahmin edebiliyorsa bu bir öğrenme faaliyeti ifade eder.

```
clear all;
% Data generation
x1=20:60;
y1=3*x1+5;
gurultu=10*(rand(1,length(x1))-0.5);
y2=y1+gurultu;

x=x1(1:20);
y=y2(1:20);
% Öğrenme katsayısı
etam=1e-4;
etab=2e-1;
m=0;
b=0;
Maxit=2000;

for i=1:Maxit
% Bacth-Mode Gradient Descent
mde_dx=-(2/length(x))*sum(x.*(y-(m*x+b)));
bde_dx=-(2/length(x))*sum((y-(m*x+b)));

m=m-etam*mde_dx;
b=b-etab*bde_dx;

mh(i)=m;
bh(i)=b;
E(i)=(2/length(x))*sum((y-(m*x+b)).^2);
end

yg=(m*x1+b);
%yyol=(mh.*x1+bh);

figure(1)
plot(x,y,'or',x1(21:40),y2(21:40),'og',x1,yg)
xlabel('x')
ylabel('y')
title('Lineer Modelleme ve Tahmin')
legend('Ölçüm verisi','Gelecek ölçümler','Lineer Model')

figure(2)
plot(1:Maxit,mh)
xlabel('iterasyon')
ylabel('m')
title('m parametresinin değişimi')

figure(3)
plot(1:Maxit,bh)
xlabel('iterasyon')
ylabel('b')
title('b parametresinin değişimi')

figure(4)
plot(1:Maxit,E)
xlabel('iterasyon')
ylabel('E')
title('E parametresinin değişimi')
```


Lineer Modelleme ve Tahmin
Ölçüm verisi
Gelecek ölçümler
Lineer Model
Tahminler
Modelleme
y
x
200
180
160
140
120
100
80
60
20
25
30
35
40
45
50
55
60


m parametresinin değişimş
m
iterasyon
3.5
3
2.5
2
1.5
1
0.5
0
500
1000
1500
2000


b parametresinin değişimş
70
60
50
40
30
20
10
0
b
0
500
1000
1500
2000
iterasyon

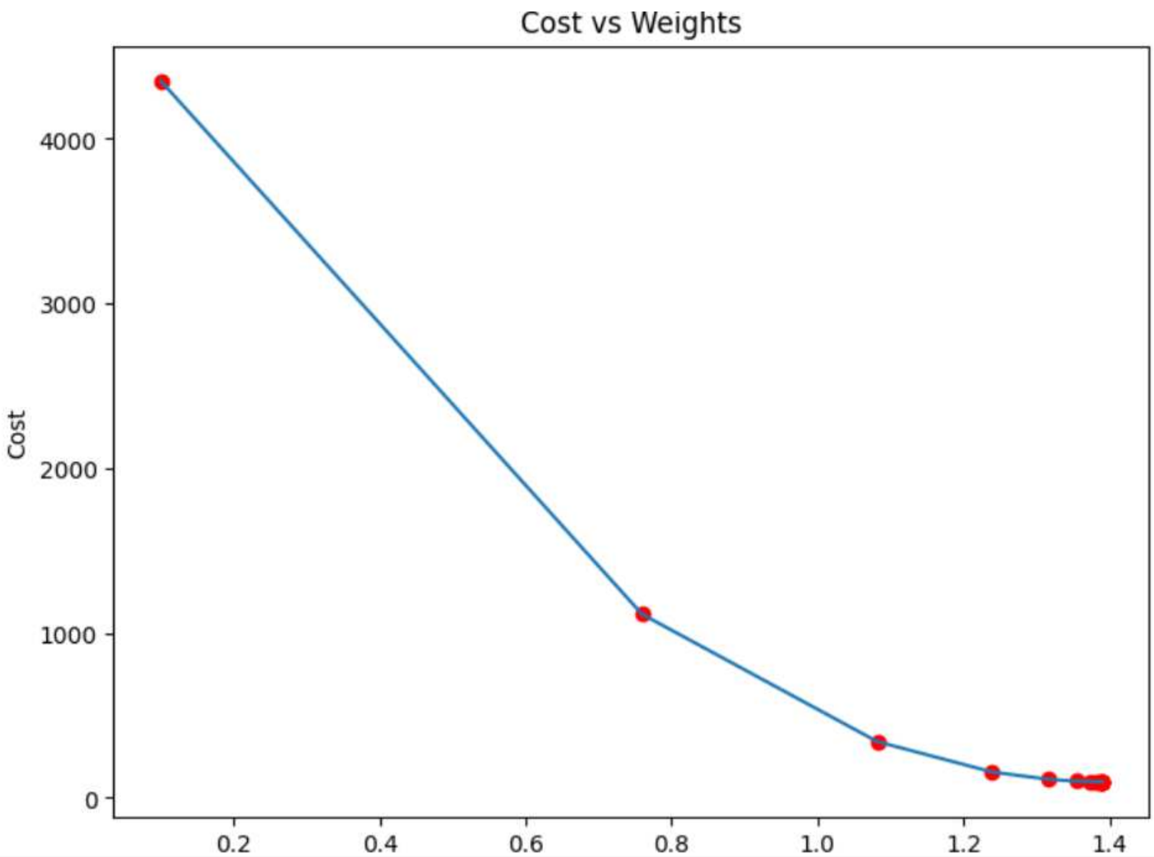

x 10^4
E parametresinin değişimş
3.5
3
2.5
2
1.5
1
0.5
0
E
0
500
1000
1500
2000
iterasyon

Phyton kodu

```
# Importing Libraries
import numpy as np
import matplotlib.pyplot as plt

def mean_squared_error(y_true, y_predicted):

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

# Gradient Descent Function
# Here iterations, learning_rate, stopping_threshold
# are hyperparameters that can be tuned
def gradient_descent(x, y, iterations = 1000, learning_rate = 0.0001,
          stopping_threshold = 1e-6):

  # 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

  # Estimation of optimal parameters
  for i in range(iterations):

    # Making predictions
    y_predicted = (current_weight * x) + current_bias

    # Calculating the current cost
    current_cost = mean_squared_error(y, y_predicted)

    # 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


def main():

  # Data
  X = np.array([32.50234527, 53.42680403, 61.53035803, 47.47563963, 59.8132
0787,
    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, 87.2309
2513,
    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, estimated_bias = gradient_descent(X, Y, iterations=2000
)
  print(f"Estimated Weight: {estimated_weight}\nEstimated Bias: {estimated_
bias}")

  # Making predictions using estimated parameters
  Y_pred = estimated_weight*X + estimated_bias

  # 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',marke
rfacecolor='red',
      markersize=10,linestyle='dashed')
  plt.xlabel("X")
  plt.ylabel("Y")
  plt.show()


if __name__=="__main__":
  main()
```

```
Iteration 1: Cost 4352.088931274409, Weight 0.7593291142562117, Bias 0.02288558130709
Iteration 2: Cost 1114.8561474350017, Weight 1.081602958862324, Bias 0.02918014748569513
Iteration 3: Cost 341.42912086804455, Weight 1.2391274084945083, Bias 0.03225308846928192
Iteration 4: Cost 156.64495290904443, Weight 1.3161239281746984, Bias 0.03375132986012604
Iteration 5: Cost 112.49704004742098, Weight 1.3537591652024805, Bias 0.034479873154934775
Iteration 6: Cost 101.9493925395456, Weight 1.3721549833978113, Bias 0.034832195392868505
Iteration 7: Cost 99.4293893333546, Weight 1.3811467575154601, Bias 0.03500062439068245
Iteration 8: Cost 98.82731958262897, Weight 1.3855419247507244, Bias 0.03507916814736111
Iteration 9: Cost 98.68347500997261, Weight 1.3876903144657764, Bias 0.035113776874486774
Iteration 10: Cost 98.64910780902792, Weight 1.3887405007983562, Bias 0.035126910596389935
Iteration 11: Cost 98.64089651459352, Weight 1.389253895811451, Bias 0.03512954755833985
Iteration 12: Cost 98.63893428729509, Weight 1.38950491235671, Bias 0.035127053821718185
Iteration 13: Cost 98.63846506273883, Weight 1.3896276808137857, Bias 0.035122052266051224
Iteration 14: Cost 98.63835254057648, Weight 1.38968776283053, Bias 0.03511582492978764
Iteration 15: Cost 98.63832524036214, Weight 1.3897172043139192, Bias 0.03510899846107016
Iteration 16: Cost 98.63831830104695, Weight 1.389731668997059, Bias 0.035101879159522745
Iteration 17: Cost 98.63831622628217, Weight 1.389738813163012, Bias 0.03509461674147458
```



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