House price prediction

import libraries

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

Load the Data

Function load_house_data to convert the text to float and split the data

```
def load_house_data():
    data = np.loadtxt("houses.txt", delimiter=',', skiprows=1)
    X = data[:,:4]
    y = data[:,4]
    return X, y
```

```
X_train,y_train = load_house_data()
```

```
X_train
```

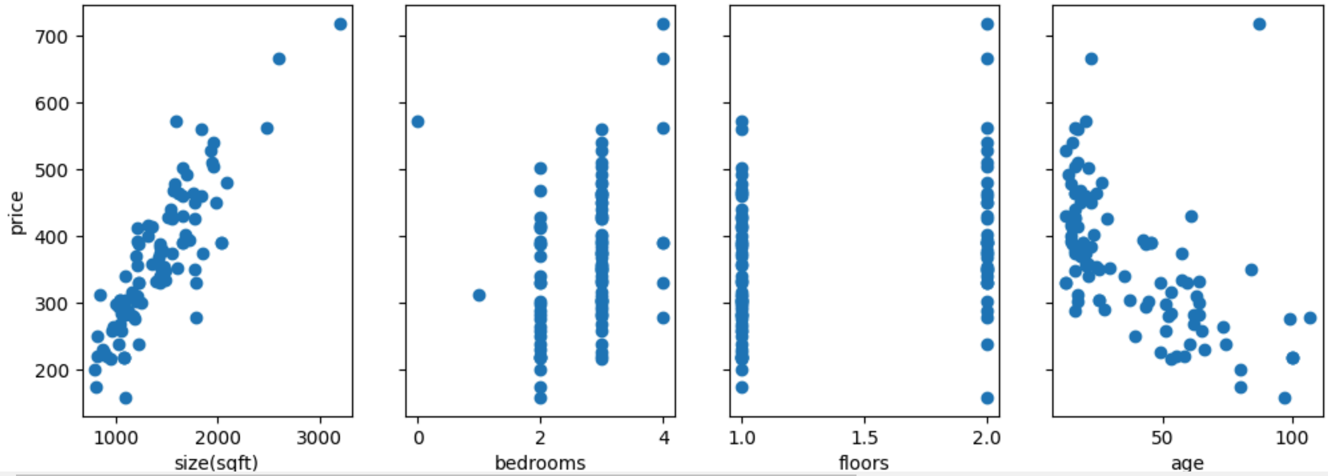
```
array([[1.244e+03, 3.000e+00, 1.000e+00, 6.400e+01],
       [1.947e+03, 3.000e+00, 2.000e+00, 1.700e+01],
       [1.725e+03, 3.000e+00, 2.000e+00, 4.200e+01],
       [1.959e+03, 3.000e+00, 2.000e+00, 1.500e+01],
       [1.314e+03, 2.000e+00, 1.000e+00, 1.400e+01],
       [8.640e+02, 2.000e+00, 1.000e+00, 6.600e+01],
       [1.836e+03, 3.000e+00, 1.000e+00, 1.700e+01],
       [1.026e+03, 3.000e+00, 1.000e+00, 4.300e+01],
       [3.194e+03, 4.000e+00, 2.000e+00, 8.700e+01],
       [7.880e+02, 2.000e+00, 1.000e+00, 8.000e+01],
       [1.200e+03, 2.000e+00, 2.000e+00, 1.700e+01],
       [1.557e+03, 2.000e+00, 1.000e+00, 1.800e+01],
       [1.430e+03, 3.000e+00, 1.000e+00, 2.000e+01],
       [1.220e+03, 2.000e+00, 1.000e+00, 1.500e+01],
       [1.092e+03, 2.000e+00, 1.000e+00, 6.400e+01],
       [8.480e+02, 1.000e+00, 1.000e+00, 1.700e+01],
       [1.682e+03, 3.000e+00, 2.000e+00, 2.300e+01],
       [1.768e+03, 3.000e+00, 2.000e+00, 1.800e+01],
       [1.040e+03, 3.000e+00, 1.000e+00, 4.400e+01],
       [1.652e+03, 2.000e+00, 1.000e+00, 2.100e+01],
       [1.088e+03, 2.000e+00, 1.000e+00, 3.500e+01],
       [1.316e+03, 3.000e+00, 1.000e+00, 1.400e+01],
       [1.593e+03, 0.000e+00, 1.000e+00, 2.000e+01],
       [9.720e+02, 2.000e+00, 1.000e+00, 7.300e+01],
       [1.097e+03, 3.000e+00, 1.000e+00, 3.700e+01],
       [1.004e+03, 2.000e+00, 1.000e+00, 5.100e+01],
       [9.040e+02, 3.000e+00, 1.000e+00, 5.500e+01],
       [1.694e+03, 3.000e+00, 1.000e+00, 1.300e+01],
       [1.073e+03, 2.000e+00, 1.000e+00, 1.000e+02],
       [1.419e+03, 3.000e+00, 2.000e+00, 1.900e+01],
       [1.164e+03, 3.000e+00, 1.000e+00, 5.200e+01],
       [1.935e+03, 3.000e+00, 2.000e+00, 1.200e+01],
       [1.216e+03, 2.000e+00, 2.000e+00, 7.400e+01],
       [2.482e+03, 4.000e+00, 2.000e+00, 1.600e+01],
       [1.200e+03, 2.000e+00, 1.000e+00, 1.800e+01],
       [1.840e+03, 3.000e+00, 2.000e+00, 2.000e+01],
       [1.851e+03, 3.000e+00, 2.000e+00, 5.700e+01],
       [1.660e+03, 3.000e+00, 2.000e+00, 1.900e+01],
       [1.096e+03, 2.000e+00, 2.000e+00, 9.700e+01],
       [1.775e+03, 3.000e+00, 2.000e+00, 2.800e+01],
       [2.030e+03, 4.000e+00, 2.000e+00, 4.500e+01],
       [1.784e+03, 4.000e+00, 2.000e+00, 1.070e+02],
       [1.073e+03, 2.000e+00, 1.000e+00, 1.000e+02],
       [1.552e+03, 3.000e+00, 1.000e+00, 1.600e+01],
       [1.953e+03, 3.000e+00, 2.000e+00, 1.600e+01],
       [1.224e+03, 2.000e+00, 2.000e+00, 1.200e+01],
       [1.616e+03, 3.000e+00, 1.000e+00, 1.600e+01],
       [8.160e+02, 2.000e+00, 1.000e+00, 5.800e+01],
       [1.349e+03, 3.000e+00, 1.000e+00, 2.100e+01],
       [1.571e+03, 3.000e+00, 1.000e+00, 1.400e+01],
       [1.486e+03, 3.000e+00, 1.000e+00, 5.700e+01],
       [1.506e+03, 2.000e+00, 1.000e+00, 1.600e+01],
       [1.097e+03, 3.000e+00, 1.000e+00, 2.700e+01],
       [1.764e+03, 3.000e+00, 2.000e+00, 2.400e+01],
       [1.208e+03, 2.000e+00, 1.000e+00, 1.400e+01],
```

```
 [1.470e+03, 3.000e+00, 2.000e+00, 2.400e+01],
 [1.768e+03, 3.000e+00, 2.000e+00, 8.400e+01],
 [1.654e+03, 3.000e+00, 1.000e+00, 1.900e+01],
```

```
y_train
```

```
array([300.   , 509.8  , 394.   , 540.   , 415.   , 230.   , 560.   ,
       294.   , 718.2  , 200.   , 302.   , 468.   , 374.2  , 388.   ,
       282.   , 311.8  , 401.   , 449.8  , 301.   , 502.   , 340.   ,
       400.282, 572.   , 264.   , 304.   , 298.   , 219.8  , 490.7  ,
       216.96 , 368.2  , 280.   , 526.87 , 237.   , 562.426, 369.8  ,
       460.   , 374.   , 390.   , 158.   , 426.   , 390.   , 277.774,
       216.96 , 425.8  , 504.   , 329.   , 464.   , 220.   , 358.   ,
       478.   , 334.   , 426.98 , 290.   , 463.   , 390.8  , 354.   ,
       350.   , 460.   , 237.   , 288.304, 282.   , 249.   , 304.   ,
       332.   , 351.8  , 310.   , 216.96 , 666.336, 330.   , 480.   ,
       330.3  , 348.   , 304.   , 384.   , 316.   , 430.4  , 450.   ,
       284.   , 275.   , 414.   , 258.   , 378.   , 350.   , 412.   ,
       373.   , 225.   , 390.   , 267.4  , 464.   , 174.   , 340.   ,
       430.   , 440.   , 216.   , 329.   , 388.   , 390.   , 356.   ,
       257.8  ])
```

```
X_features = ['size(sqft)','bedrooms','floors','age']
fig,ax = plt.subplots(1,4, figsize=(12,4), sharey=True)
for i in range(len(ax)):
  ax[i].scatter(X_train[:,i],y_train)
  ax[i].set_xlabel(X_features[i])
ax[0].set_ylabel("price")
plt.show()
```



Compute the cost Function

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

Compute Gradient Descent

```
def compute_gradient(X,y,w,b):
  m,n = X.shape
  d_w = np.zeros((n,))
  d_b = 0
  for i in range(m):
    f_wb = (np.dot(w,X[i]) + b) - y[i]
    for j in range(n):
      d_w[j] += f_wb*X[i,j]
    d_b = d_b + f_wb
  d_w = d_w/m
  d_b = d_b/m
  return d_w,d_b
```

```
def gradient_descent(X,y,w_in,b_in,cost_function,gradient_function,alpha,num_iters):
  J_his = []
  w = w_in
  b = b_in
  for i in range(num_iters):
    d_w,d_b = gradient_function(X,y,w,b)
    w = w - alpha*d_w
    b = b - alpha*d_b
    if i<100000:
      J_his.append(cost_function(X,y,w,b))
    if i%math.ceil(num_iters/10) == 0:
      print(f"Iteration {i:4}: Cost {J_his[-1]:.2f} , w = {w}, b = {b}")
  return w,b,J_his
```

Scale The Features

Using Zero_score_normalization

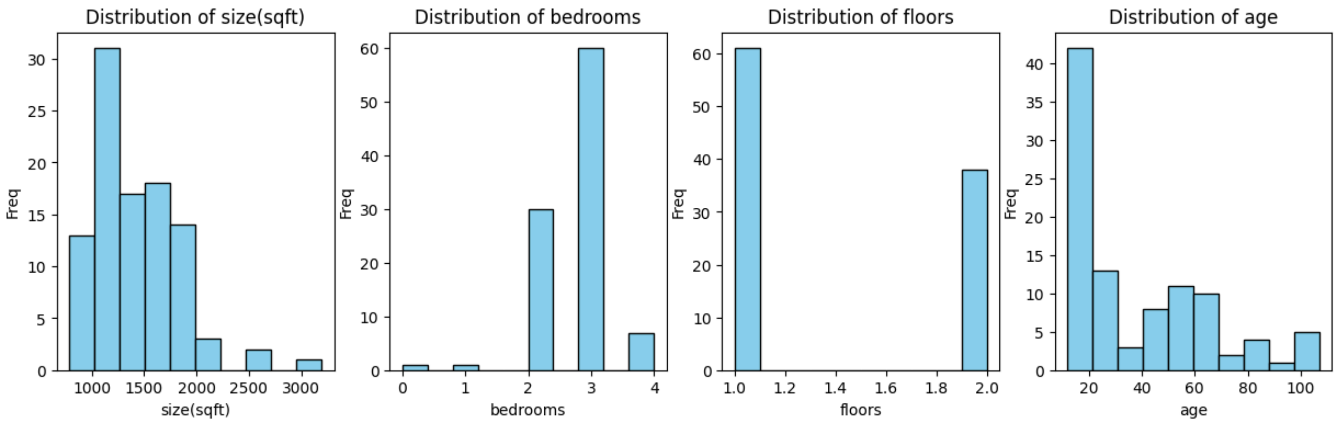
```
# def z_score_normalization(X):
#   m,n = X.shape
#   u = np.zeros((n,))
#   s = np.zeros((n,))
#   X_scaled = np.zeros((m,n))
#   for j in range(n):
#     for i in range(m):
#       u[j] += X[i,j]
#     u[j] = u[j]/m
#   for j in range(n):
#     for i in range(m):
#       s[j] += (X[i,j]-u[j])**2
#     s[j] = s[j]/m
#   s = np.sqrt(s)
#   for j in range(n):
#     for i in range(m):
#       X_scaled[i,j] = (X[i,j]-u[j])/s[j]
#   return X_scaled, u, s
```

```
def zscore_normalize_features(X):
    mu     = np.mean(X, axis=0)
    sigma  = np.std(X, axis=0)
    X_norm = (X - mu) / sigma
    return (X_norm, mu, sigma)
```

```
X_scaled, u, s = zscore_normalize_features(X_train)
Y_scaled , u_y, s_y = zscore_normalize_features(y_train)
```

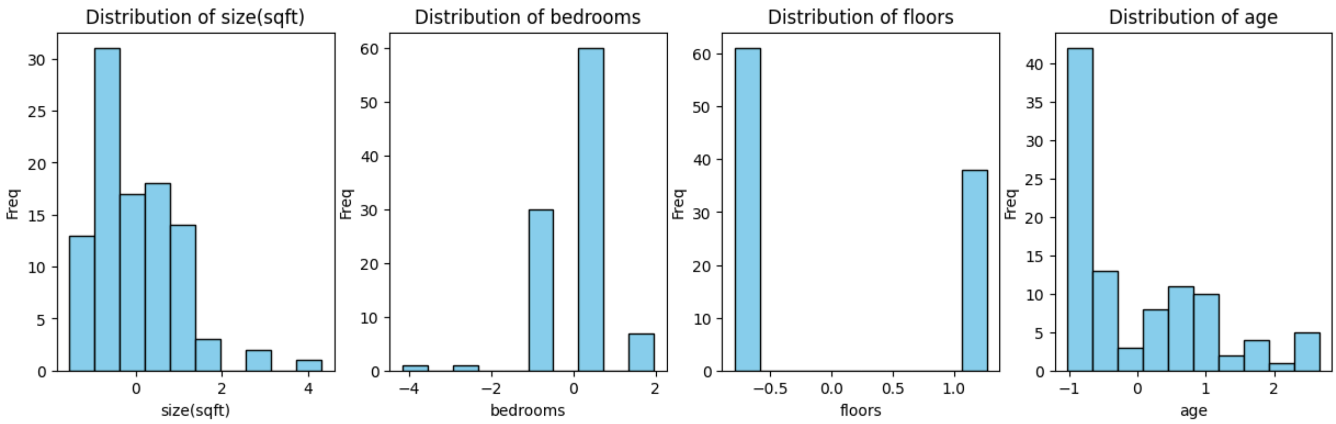
Before Scale

```
fig, axes = plt.subplots(1 ,4, figsize=(15,4))

for i in range(len(X_features)):
    axes[i].hist(X_train[:, i], color='skyblue', edgecolor='black')
    axes[i].set_xlabel(X_features[i])
    axes[i].set_ylabel('Freq')
    axes[i].set_title(f'Distribution of {X_features[i]}')
plt.show()
```



After Scale

```
fig, axes = plt.subplots(1 ,4, figsize=(15,4))

for i in range(len(X_features)):
    axes[i].hist(X_scaled[:, i], color='skyblue', edgecolor='black')
    axes[i].set_xlabel(X_features[i])
    axes[i].set_ylabel('Freq')
    axes[i].set_title(f'Distribution of {X_features[i]}')
plt.show()
```



Histogram shows the data before scaling, where there is a large range of values across the features. After scaling, the values of all the features are centered around zero.

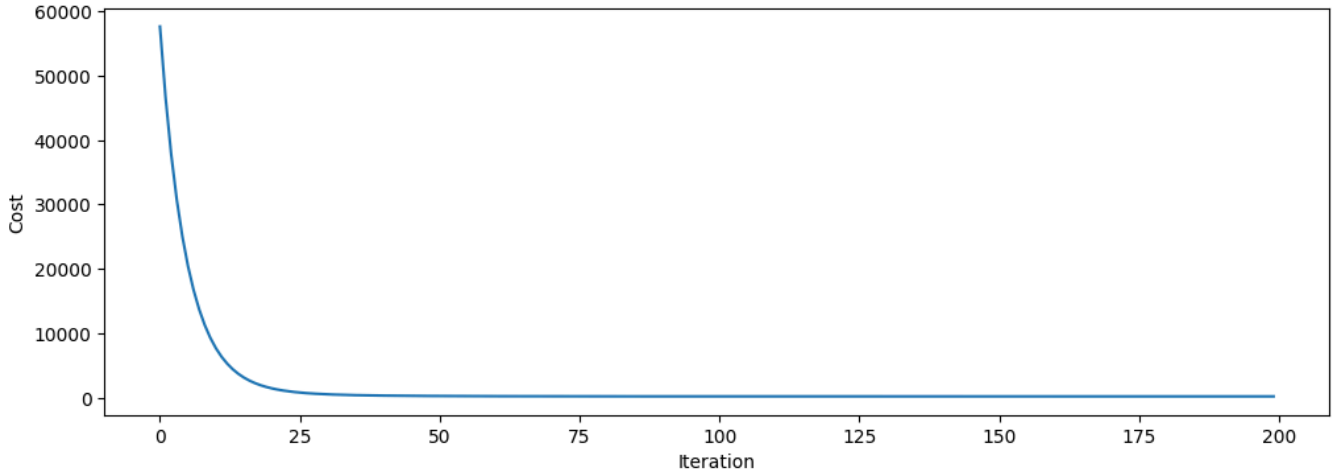
Train the model

```
w_init = np.zeros((4,))
b_init = 3.6e-04
iterations = 200
alpha = 1.0e-1
w_final,b_final,J_hist = gradient_descent(X_scaled,y_train,w_init,b_init,compute_cost,compute_gradient,alpha,iterations)
print(f"w_final = {w_final}, b_final = {b_final}" )
```

```
Iteration    0: Cost 57616.93 , w = [ 8.91428163  2.95182478  3.28229064 -5.96084411], b = 36.31593208080814
Iteration   20: Cost 1462.59 , w = [ 70.33772601  -2.41571178  -6.77185213 -40.09494761], b = 323.41994893993495
Iteration   40: Cost 343.31 , w = [ 90.22024072 -11.66719423 -20.27717684 -40.9990357 ], b = 358.3250948291819
Iteration   60: Cost 248.04 , w = [100.25109553 -16.29905477 -26.64039505 -39.80221069], b = 362.5687456943923
Iteration   80: Cost 226.54 , w = [105.33903931 -18.70544998 -29.70149487 -38.95066663], b = 363.084674569835
Iteration  100: Cost 221.09 , w = [107.91666103 -19.95469161 -31.20467374 -38.47753825], b = 363.147399476518
Iteration  120: Cost 219.69 , w = [109.22193552 -20.59791022 -31.95212918 -38.22963748], b = 363.155025360832
Iteration  140: Cost 219.33 , w = [109.8827985  -20.92691532 -32.32657716 -38.10220447], b = 363.15595249033527
Iteration  160: Cost 219.24 , w = [110.21737013 -21.09448721 -32.51498401 -38.03719541], b = 363.15606520763856
Iteration  180: Cost 219.21 , w = [110.38674583 -21.17961866 -32.61002485 -38.00414919], b = 363.1560789114312
w_final = [110.46944627 -21.22126909 -32.65633572 -37.9879772 ], b_final = 363.1560805518715
```

```
fig,ax = plt.subplots(1,1, figsize=(12,4), sharey=True)
ax.plot(J_hist)
ax.set_xlabel("Iteration")
ax.set_ylabel("Cost")
```

```
Text(0, 0.5, 'Cost')
```




```
def predict(x,w,b):
  p = np.dot(x,w) + b
  return p
```

```
y_pred = predict(X_scaled,w_final,b_final)
print(y_train)
print(y_pred)
```

```
[300.    509.8   394.    540.    415.    230.    560.    294.    718.2
 200.    302.    468.    374.2   388.    282.    311.8   401.    449.8
 301.    502.    340.    400.282 572.    264.    304.    298.    219.8
 490.7   216.96  368.2   280.    526.87  237.    562.426 369.8   460.
 374.    390.    158.    426.    390.    277.774 216.96  425.8   504.
 329.    464.    220.    358.    478.    334.    426.98  290.    463.
 390.8   354.    350.    460.    237.    288.304 282.    249.    304.
 332.    351.8   310.    216.96  666.336 330.    480.    330.3   348.
 304.    384.    316.    430.4   450.    284.    275.    414.    258.
 378.    350.    412.    373.    225.    390.    267.4   464.    174.
 340.    430.    440.    216.    329.    388.    390.    356.    257.8   ]
[295.17674922 485.96026268 389.53823791 492.12815525 420.19633329
 222.79467182 523.32005756 267.61686461 684.92396576 181.76646469
 318.03001754 479.51803654 409.93673744 393.49493764 286.93266736
 323.25990109 405.99754062 436.446558   269.90052045 500.59314971
 328.59545068 388.1833992  551.33177768 241.46406378 295.51384604
 282.4728714  217.19050388 491.10474504 228.78148207 341.30823524
 291.39032847 490.10802964 238.3251574  598.46759093 383.70633858
 452.82259436 401.24914662 405.98784087 172.22523477 423.58856228
 434.42347911 277.03470288 228.78148207 448.573773   489.04420897
 331.83946832 465.75009981 221.70175188 386.72428318 456.62032632
 370.44039448 468.77798241 310.2505025  426.5310437  391.74804201
 347.62729243 339.18462534 471.52754691 243.36968894 298.03327468
 272.89635052 249.70139916 297.90016773 334.89700496 375.95351834
 288.86739179 228.78148207 621.02606949 352.73215734 511.07562707
 364.117591   363.17393909 297.90016773 407.25778626 288.57476229
 385.97654837 488.25361828 260.93161134 258.9718191  427.58406103
 238.11445242 355.65929611 339.71250423 390.27437637 381.69216643
 220.12813542 434.42347911 243.37453882 465.75009981 185.79216628
 341.31308511 410.17034998 445.62159183 231.94655985 331.83946832
 409.12888209 405.98784087 351.42987426 274.18703725]
```

```
fig,ax = plt.subplots(1,4, figsize=(12,3), sharey=True)
for i in range(len(ax)):
  ax[i].scatter(X_scaled[:,i], y_train,color='g')
  ax[i].scatter(X_scaled[:,i], y_pred)
  ax[i].set_xlabel(X_features[i])
  ax[i].legend(['Training data', 'prediction'])
ax[0].set_ylabel('Price (1000s of dollars)')
```

```
Text(0, 0.5, 'Price (1000s of dollars)')
```