In [37]:
```
import pandas as pd
```

In [38]:
```
url = "SquareFeet_Data2.csv"
data = pd.read_csv(url)
data
```

Out[38]:

	SquareFeet	Density	City	price
0	850	low	CityA	467500
1	779	medium	CityB	363014
2	990	medium	CityA	594000
3	665	medium	CityC	266000
4	550	medium	CityC	220000
5	880	medium	CityB	478720
6	567	low	CityB	264222
7	1020	low	CityB	497760
8	2067	high	CityC	756522
9	577	high	CityA	375050
10	989	medium	CityA	581532
11	720	low	CityC	165600
12	585	high	CityA	321750
13	656	medium	CityC	196800
14	788	high	CityC	253736
15	1222	high	CityB	596336
16	565	high	CityA	326005
17	844	high	CityC	196652
18	744	low	CityA	415152
19	1356	high	CityB	737664
20	1555	low	CityB	622000
21	2000	high	CityC	560000
22	647	low	CityA	355850
23	769	low	CityC	196095
24	855	medium	CityB	470250
25	900	medium	CityA	509400
26	456	medium	CityC	151848
27	669	low	CityB	347880
28	1899	high	CityA	1044450
29	633	low	CityC	212055

	SquareFeet	Density	City	price
30	890	medium	CityB	400500
31	946	low	CityC	272448
32	1235	low	CityA	679250

In [39]:
```
#pre-data processing
#convert sqft to m^2
#convert price to k

data["SquareFeet"] = data["SquareFeet"] * 0.092903
data['price'] = data['price'] / 1000
```

In [40]:
```
data["Density"] = data["Density"].map({"low": 0, "medium": 1, "high" : 2})
data
```

Out[40]:

	SquareFeet	Density	City	price
0	78.967550	0	CityA	467.500
1	72.371437	1	CityB	363.014
2	91.973970	1	CityA	594.000
3	61.780495	1	CityC	266.000
4	51.096650	1	CityC	220.000
5	81.754640	1	CityB	478.720
6	52.676001	0	CityB	264.222
7	94.761060	0	CityB	497.760
8	192.030501	2	CityC	756.522
9	53.605031	2	CityA	375.050
10	91.881067	1	CityA	581.532
11	66.890160	0	CityC	165.600
12	54.348255	2	CityA	321.750
13	60.944368	1	CityC	196.800
14	73.207564	2	CityC	253.736
15	113.527466	2	CityB	596.336
16	52.490195	2	CityA	326.005
17	78.410132	2	CityC	196.652
18	69.119832	0	CityA	415.152
19	125.976468	2	CityB	737.664
20	144.464165	0	CityB	622.000
21	185.806000	2	CityC	560.000
22	60.108241	0	CityA	355.850
23	71.442407	0	CityC	196.095
24	79.432065	1	CityB	470.250
25	83.612700	1	CityA	509.400
26	42.363768	1	CityC	151.848
27	62.152107	0	CityB	347.880
28	176.422797	2	CityA	1044.450
29	58.807599	0	CityC	212.055

	SquareFeet	Density	City	price
30	82.683670	1	CityB	400.500
31	87.886238	0	CityC	272.448
32	114.735205	0	CityA	679.250

One Hot Encoding

In [41]:
```
#Convert City to CityA, CityB, CityC
#not need CityC is because if CityA and CityB are 0. it means it is CityC
```

In [42]:
```
from sklearn.preprocessing import OneHotEncoder

onehot_encoder = OneHotEncoder()
onehot_encoder.fit(data[["City"]])
city_encoded = onehot_encoder.transform(data[["City"]]).toarray()

data[["CityA","CityB","CityC"]] = city_encoded
data = data.drop(["City","CityC"],axis=1)
data
```

Out[42]:

	SquareFeet	Density	price	CityA	CityB
0	78.967550	0	467.500	1.0	0.0
1	72.371437	1	363.014	0.0	1.0
2	91.973970	1	594.000	1.0	0.0
3	61.780495	1	266.000	0.0	0.0
4	51.096650	1	220.000	0.0	0.0
5	81.754640	1	478.720	0.0	1.0
6	52.676001	0	264.222	0.0	1.0
7	94.761060	0	497.760	0.0	1.0
8	192.030501	2	756.522	0.0	0.0
9	53.605031	2	375.050	1.0	0.0
10	91.881067	1	581.532	1.0	0.0
11	66.890160	0	165.600	0.0	0.0
12	54.348255	2	321.750	1.0	0.0
13	60.944368	1	196.800	0.0	0.0
14	73.207564	2	253.736	0.0	0.0
15	113.527466	2	596.336	0.0	1.0
16	52.490195	2	326.005	1.0	0.0
17	78.410132	2	196.652	0.0	0.0
18	69.119832	0	415.152	1.0	0.0
19	125.976468	2	737.664	0.0	1.0
20	144.464165	0	622.000	0.0	1.0
21	185.806000	2	560.000	0.0	0.0
22	60.108241	0	355.850	1.0	0.0
23	71.442407	0	196.095	0.0	0.0
24	79.432065	1	470.250	0.0	1.0
25	83.612700	1	509.400	1.0	0.0
26	42.363768	1	151.848	0.0	0.0
27	62.152107	0	347.880	0.0	1.0
28	176.422797	2	1044.450	1.0	0.0
29	58.807599	0	212.055	0.0	0.0

	SquareFeet	Density	price	CityA	CityB
30	82.683670	1	400.500	0.0	1.0
31	87.886238	0	272.448	0.0	0.0
32	114.735205	0	679.250	1.0	0.0

split train and test data

In [43]:
```
from sklearn.model_selection import train_test_split

x = data[["SquareFeet","Density","CityA","CityB"]]
y = data["price"]

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2,random_state=
len(x),len(x_train),len(x_test)
```

Out[43]: (33, 26, 7)

In [44]:
```
x_train = x_train.to_numpy()
x_test = x_test.to_numpy()
```

In [45]:
```
#y_pred = w1 * x1 + w2 * x2 + w3 * x3 ... + b
#y_pred = w1*SquareFeet + w2*Density + w3*CityA + w4*CityB + b
```

optimizer - gradient descent

In [46]:
```
#y_pred = w1 * x1 + w2 * x2 + w3 * x3 ... + b

#cost = (y - y_pred)**2
#cost = (y - w1 * x1 + w2 * x2 + w3 * x3 ... + b)**2

# w1_gradient = 2*x1*(w*x+b - y)

# w1_gradient = 2*x1*(y_pred - y)
# w2_gradient = 2*x2*(y_pred - y)
# w3_gradient = 2*x3*(y_pred - y)
# w4_gradient = 2*x4*(y_pred - y)

# b_gradient = 2*(y_pred - y)
```

In [47]:
```
import numpy as np

w = np.array([80.5,1,1,0])
b = 1

y_pred = (w*x_train).sum(axis = 1) + b
```

In [48]:
```
def compute_cost(x,y,w,b):
    y_pred = (w*x).sum(axis = 1) + b
```

```
    cost = ((y - y_pred)**2).mean()
    return cost
```

In [49]:
```
w = np.array([80.5,1,1,0])
b = 1
compute_cost(x_train,y_train,w,b)
```

Out[49]:
```
np.float64(52763482.698299974)
```

In [50]:
```
w_gradient = np.zeros(x_train.shape)
x_train.shape
```

Out[50]:
```
(26, 4)
```

In [51]:
```
x_train.shape[1]
```

Out[51]:
```
4
```

In [52]:
```
y_pred = (x_train*w).sum(axis = 1) + b
w_gradient = np.zeros(x_train.shape[1])
b_gradient = (y_pred - y_train).mean()

# w1_gradient = 2*x1*(w*x+b - y)
#             = 2*x1*(y_pred - y)

for i in range(x_train.shape[1]):
    w_gradient[i] = (2 *(x_train[:,i])* (y_pred - y_train)).mean()

w_gradient,b_gradient
```

Out[52]:
```
(array([1388716.77998974,   13554.49597623,    3516.80135812,
           4085.41275908]),
 np.float64(6659.75281325))
```

In [53]:
```
def compute_gradient(x,y,w,b):
    y_pred = (x*w).sum(axis = 1) + b
    w_gradient = np.zeros(x.shape[1])
    b_gradient = (y_pred - y).mean()

    for i in range(x.shape[1]):
        w_gradient[i] = (2 *(x[:,i])* (y_pred - y)).mean()

    return w_gradient,b_gradient
```

In [54]:
```
w = np.array([80.5,1,1,0])
b = 1
learning_rate = 0.0000001

w_gradient,b_gradient = compute_gradient(x_train,y_train,w,b)
print(compute_cost(x_train,y_train,w,b))

w = w - w_gradient * learning_rate
b = b - b_gradient * learning_rate
```

```
print(compute_cost(x_train,y_train,w,b))
```

```
52763482.698299974
52570775.4437354
```

In [55]:
```
def gradient_descent(x,y,w_init,b_init,learning_rate,cost_function,gradient_functio
    c_record = []
    b_record = []
    w_record = []

    w = w_init
    b = b_init

    for i in range(run_iter):
        w_gradient,b_gradient = compute_gradient(x,y,w,b)

        w = w - w_gradient * learning_rate
        b = b - b_gradient * learning_rate
        cost = compute_cost(x,y,w,b)

        c_record.append(cost)
        b_record.append(b)
        w_record.append(w)

        if i%itra == 0:
            print(f"Iteration {i:5}: Cost {cost:.2f}: w {w}: b {b:.2f} :w_gradient

    return w,b,c_record,w_record,b_record
```

In [56]:
```
w_init = np.array([80.5,1,1,0])
b_init = 1
learning_rate = 0.0000001
run_iter = 10000

w_final,b_final,c_record,w_record,b_record = gradient_descent(x_train,y_train,w_ini
```

```
Iteration     0: Cost 52570775.44: w [ 8.03611283e+01  9.98644550e-01  9.99648320e-0
1 -4.08541276e-04]: b 1.00 :w_gradient [1388716.77998974   13554.49597623    3516.80
135812    4085.41275908]: b_gradient 6659.75
Iteration  1000: Cost 1366316.69: w [16.70559463  0.37702112  0.84254717 -0.1857068
6]: b 0.69 :w_gradient [222777.19810896   2179.3184498     500.59672869    624.88185
15 ]: b_gradient 1057.04
Iteration  2000: Cost 48603.49: w [ 6.49399109  0.27680915  0.82370156 -0.2123826 ]:
b 0.65 :w_gradient [3.57380632e+04 3.54520468e+02 1.67423796e+01 6.97464447e+01]: b_
gradient 158.26
Iteration  3000: Cost 14692.29: w [ 4.85582191  0.26024167  0.82703452 -0.21361225]:
b 0.64 :w_gradient [5733.38838107   61.78706063  -60.8742768   -19.30764072]: b_grad
ient 14.08
Iteration  4000: Cost 13818.89: w [ 4.5929965   0.25709245  0.83392505 -0.2107599 ]:
b 0.64 :w_gradient [920.0614164   14.82627863 -73.32259376 -33.59331738]: b_gradient
-9.05
Iteration  5000: Cost 13795.68: w [ 4.55080268  0.25609584  0.841386   -0.20725274]:
b 0.64 :w_gradient [147.91118253   7.29215315 -75.31665576 -35.88470316]: b_gradient
-12.76
Iteration  6000: Cost 13794.36: w [ 4.54400242  0.25544463  0.84893816 -0.20364057]:
b 0.64 :w_gradient [ 24.0434253    6.08282181 -75.63365433 -36.25197394]: b_gradient
-13.35
Iteration  7000: Cost 13793.60: w [ 4.54287996  0.25484889  0.85650467 -0.20001159]:
b 0.64 :w_gradient [  4.17264655   5.88810919 -75.68161983 -36.31057966]: b_gradient
-13.45
Iteration  8000: Cost 13792.86: w [ 4.54266834  0.25426213  0.8640732  -0.19637994]:
b 0.65 :w_gradient [  0.98498345   5.85616101 -75.68642739 -36.31966953]: b_gradient
-13.46
Iteration  9000: Cost 13792.11: w [ 4.54260283  0.25367688  0.87164175 -0.19274789]:
b 0.65 :w_gradient [  0.47361299   5.85032347 -75.68431174 -36.32081609]: b_gradient
-13.46
```

In [57]:
```
y_pred = (w_final * x_test).sum(axis = 1) + b_final
pd.DataFrame({
    "y_pred": y_pred,
    "y_test": y_test
})
```

Out[57]:

	y_pred	y_test
16	240.473758	326.005
1	329.464094	363.014
11	304.501087	165.600
24	361.537426	470.250
6	239.743286	264.222
25	381.596536	509.400
28	803.445137	1044.450

Standardization

In [58]:
```
## standardization
# y_pred = w1 * x1 + w2 * x2 + w3 * x3 ... + b
# some of data 'x' is a big number, it will hard to get lower cost
# for example, y_pred = w1 * big + w2 * small + w3 * small ... + b
# we can use 'Standardization' to fix this problem
```

In [59]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)
```

In [60]:
```
w = np.array([80.5,1,1,0])
b = 1

compute_cost(x_train,y_train,w,b)
```

Out[60]:
```
np.float64(186320.60341681572)
```

In [61]:
```
w_init = np.array([80.5,1,1,0])
b_init = 1
learning_rate = 0.01
run_inter = 10000

w_final,b_final,w_hist,b_hist,c_hist = gradient_descent(x_train,y_train,w_init,b_in
```

```
Iteration     0: Cost 182565.57: w [81.68970221  1.08549156  2.09796233  1.0372344
]: b 5.13 :w_gradient [-118.97022063   -8.54915583 -109.79623291 -103.7234402 ]: b_g
radient -412.58
Iteration  1000: Cost 2827.86: w [141.12336605   7.38649413 100.94190999  80.0132841
]: b 413.56 :w_gradient [ 0.00058307 -0.00087062 -0.00134113 -0.00151495]: b_gradien
t -0.02
Iteration  2000: Cost 2827.86: w [141.12281613   7.3873072  100.94314778  80.0146879
]: b 413.58 :w_gradient [ 1.35944348e-08 -1.96392132e-08 -2.90523762e-08 -3.32735270
e-08]: b_gradient -0.00
Iteration  3000: Cost 2827.86: w [141.12281611   7.38730721 100.94314781  80.0146879
3]: b 413.58 :w_gradient [ 1.40796776e-12 -3.62923367e-13 -6.98518166e-13 -6.8649359
7e-13]: b_gradient -0.00
Iteration  4000: Cost 2827.86: w [141.12281611   7.38730721 100.94314781  80.0146879
3]: b 413.58 :w_gradient [ 1.40359519e-12 -3.49805655e-14 -6.67910172e-13 -7.0179759
4e-13]: b_gradient -0.00
Iteration  5000: Cost 2827.86: w [141.12281611   7.38730721 100.94314781  80.0146879
3]: b 413.58 :w_gradient [ 1.40359519e-12 -3.49805655e-14 -6.67910172e-13 -7.0179759
4e-13]: b_gradient -0.00
Iteration  6000: Cost 2827.86: w [141.12281611   7.38730721 100.94314781  80.0146879
3]: b 413.58 :w_gradient [ 1.40359519e-12 -3.49805655e-14 -6.67910172e-13 -7.0179759
4e-13]: b_gradient -0.00
Iteration  7000: Cost 2827.86: w [141.12281611   7.38730721 100.94314781  80.0146879
3]: b 413.58 :w_gradient [ 1.40359519e-12 -3.49805655e-14 -6.67910172e-13 -7.0179759
4e-13]: b_gradient -0.00
Iteration  8000: Cost 2827.86: w [141.12281611   7.38730721 100.94314781  80.0146879
3]: b 413.58 :w_gradient [ 1.40359519e-12 -3.49805655e-14 -6.67910172e-13 -7.0179759
4e-13]: b_gradient -0.00
Iteration  9000: Cost 2827.86: w [141.12281611   7.38730721 100.94314781  80.0146879
3]: b 413.58 :w_gradient [ 1.40359519e-12 -3.49805655e-14 -6.67910172e-13 -7.0179759
4e-13]: b_gradient -0.00
```

In [62]:
```
y_pred = (w_final * x_test).sum(axis = 1) + b_final
pd.DataFrame({
    "y_pred": y_pred,
    "y_test": y_test
})
```

Out[62]:

	y_pred	y_test
16	393.841443	326.005
1	420.952385	363.014
11	211.146584	165.600
24	447.355543	470.250
6	338.384846	264.222
25	501.307158	509.400
28	857.286345	1044.450

Check final output here

In [63]:
```
sq = 80.5
density = 1
cityA = 0
cityB = 1

x_realData = np.array([[sq,density,cityA,cityB]])
x_realData = scaler.transform(x_realData)
y_realData = (x_realData * w_final).sum(axis=1) + b_final
price = round(y_realData[0],2)

print(f'Square feet(m^2): {sq} : Density: {density} : cityA : {cityA}: cityB: {city
```

```
Square feet(m^2): 80.5 : Density: 1 : cityA : 0: cityB: 1: The price is: 451.35 k
```

In [ ]: