Simple Linear Regression

August 25, 2025

[5]:
```
import pandas as pd
```

[6]:
```
url = "SquareFeet_Data.csv"
data = pd.read_csv(url)
data
```

[6]:

	SquareFeet	StateA	StateB	StateC	price
0	850	1	0	0	467500
1	779	0	1	0	363014
2	990	1	0	0	594000
3	665	0	0	1	266000
4	550	0	0	1	220000
5	880	0	1	0	478720
6	567	0	1	0	264222
7	1020	0	1	0	497760
8	2067	0	0	1	756522
9	577	1	0	0	375050
10	989	1	0	0	581532
11	720	0	0	1	165600
12	585	1	0	0	321750
13	656	0	0	1	196800
14	788	0	0	1	253736
15	1222	0	1	0	596336
16	565	1	0	0	326005
17	844	0	0	1	196652
18	744	1	0	0	415152
19	1356	0	1	0	737664
20	1555	0	1	0	622000
21	2000	0	0	1	560000
22	647	1	0	0	355850
23	769	0	0	1	196095
24	855	0	1	0	470250
25	900	1	0	0	509400
26	456	0	0	1	151848
27	669	0	1	0	347880
28	1899	1	0	0	1044450
29	633	0	0	1	212055
30	890	0	1	0	400500

```
31          946        0        0        1   272448
32         1235        1        0        0   679250
```

[7]:
```
#convert sqft to m^2

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

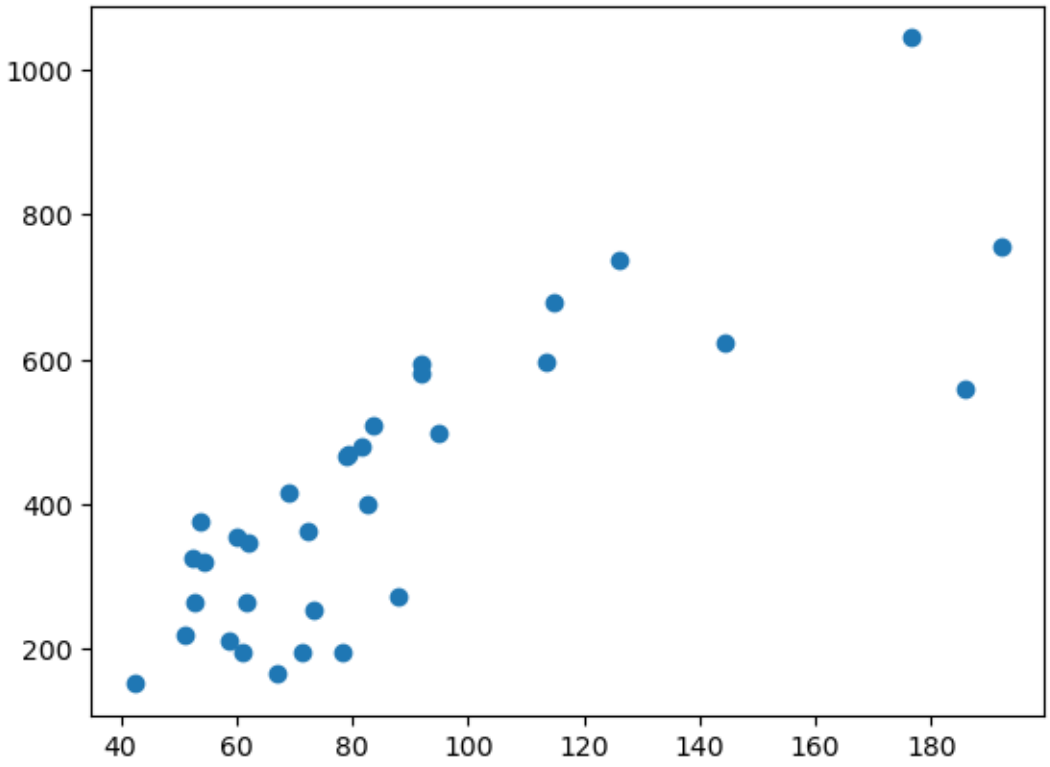
[5]:
```
x = data["SquareFeet"]
y = data["price"]

x
```

[5]:
```
0      78.967550
1      72.371437
2      91.973970
3      61.780495
4      51.096650
5      81.754640
6      52.676001
7      94.761060
8     192.030501
9      53.605031
10     91.881067
11     66.890160
12     54.348255
13     60.944368
14     73.207564
15    113.527466
16     52.490195
17     78.410132
18     69.119832
19    125.976468
20    144.464165
21    185.806000
22     60.108241
23     71.442407
24     79.432065
25     83.612700
26     42.363768
27     62.152107
28    176.422797
29     58.807599
30     82.683670
31     87.886238
32    114.735205
Name: SquareFeet, dtype: float64
```

[6]:
```
import matplotlib.pyplot as plt
```

[7]:
```
plt.scatter(x,y)
```

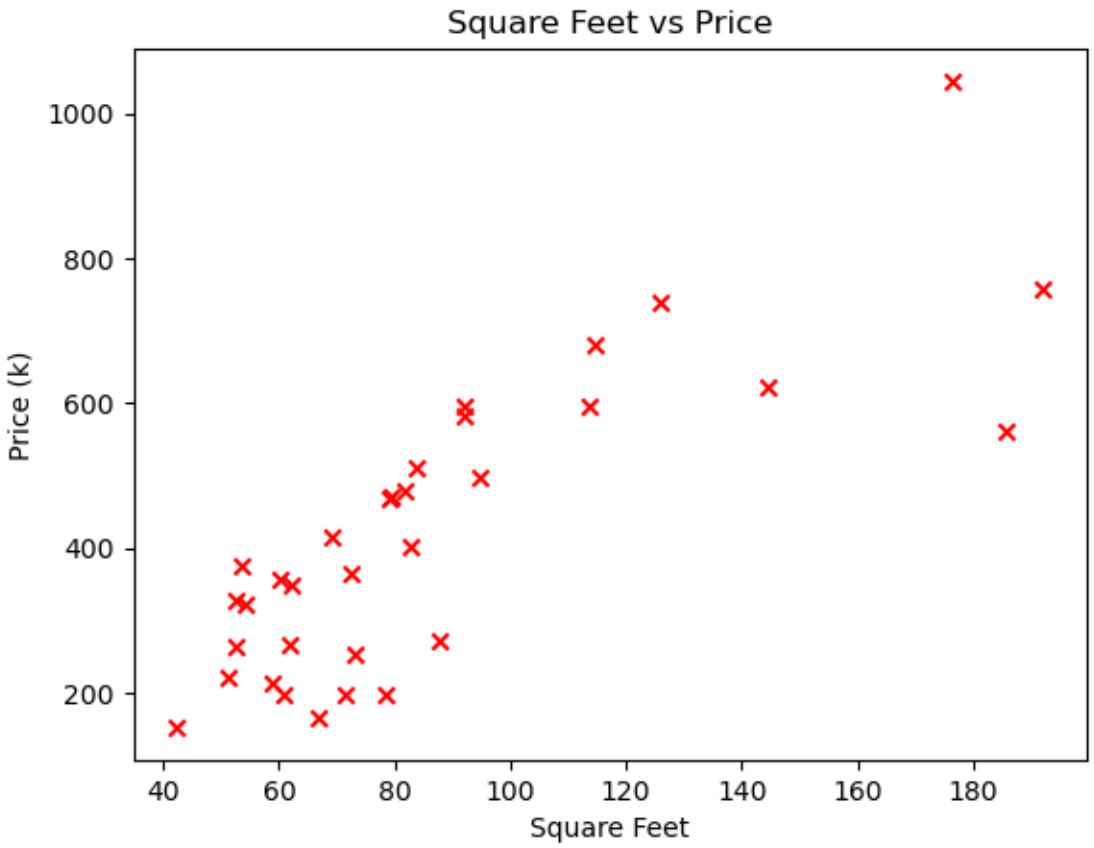
[7]:
```
<matplotlib.collections.PathCollection at 0x23c2de1bdf0>
```



[8]:
```
#m² to sqft → multiply by 10.7639
#sqft to m² → multiply by 0.092903

# x = data['SquareFeet'] * 0.092903
# y = data['price'] / 1000
```

[9]:
```
# x = data.sort_values(by='SquareFeet')
# y = data['price'].replace({',': ''}, regex=True).astype(float)
# y = data.sort_values(by='price')
```

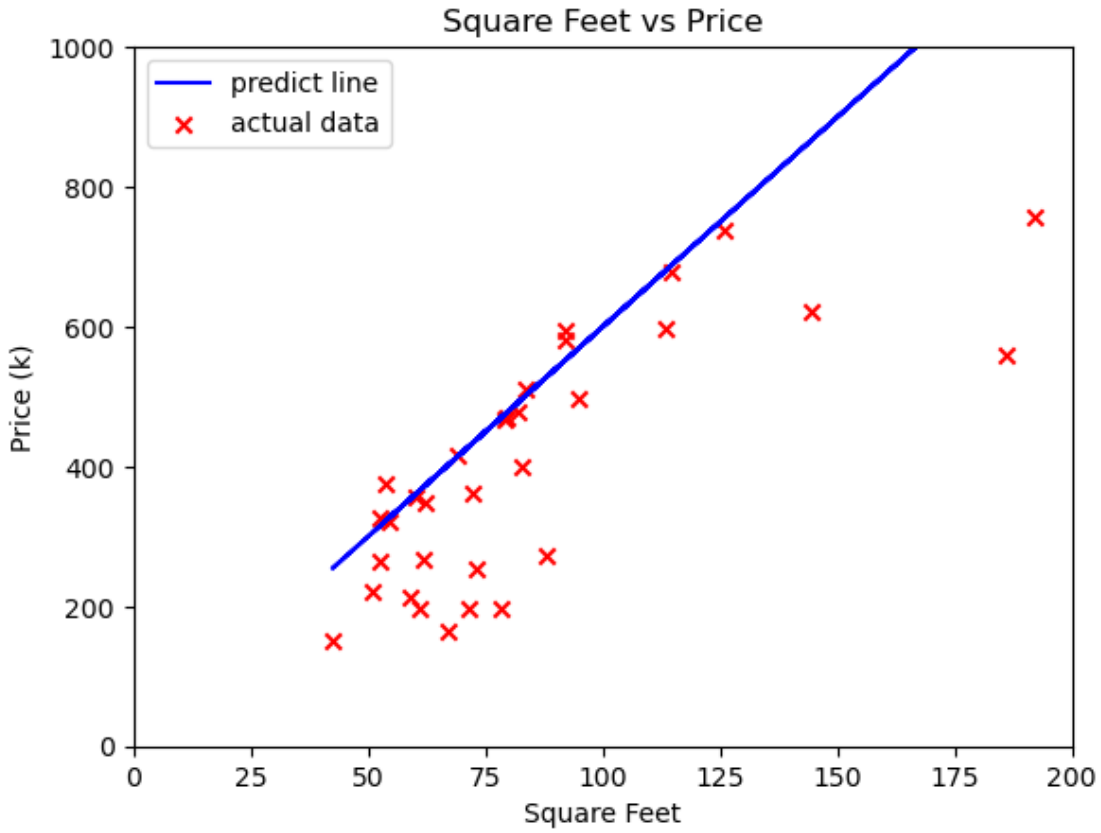
[10]:
```
plt.scatter(x,y,marker='x',color='red')
plt.xlabel("Square Feet")
plt.ylabel("Price (k)")
plt.title("Square Feet vs Price")
plt.show()
```



[11]:
```
#cost function formula
#y_pred = w*x + b

# if w=0,b=0
# y_pred = 0*x + 0
# y_pred = 0

# (y-y_pred)^2 + (y-y_pred)^2 + (y-y_pred)^2 = ?
# (1-0)^2 + (2-0)^2 + (3-0)^2 = 14

# if w=2,b=0
# y_pred = 2*x + 0
# y_pred = 2x

# (1-2)^2 + (2-4)^2 + (3-6)^2 = 14

# if w=1,b=0
# y_pred = 1*x + 0
# y_pred = x
```

```
# (1-1)^2 + (2-2)^2 + (3-3)^2 = 0
```

[12]:
```
w = 0
b = 0

def plot_pred(w,b):
    y_pred = w*x + b
    plt.plot(x,y_pred,color="blue",label="predict line")
    plt.scatter(x,y,marker="x",color="red",label="actual data")
    plt.title("Square Feet vs Price")
    plt.xlabel("Square Feet")
    plt.ylabel("Price (k)")
    plt.xlim([0,200])
    plt.ylim([0,1000])
    plt.legend()
    plt.show()

plot_pred(6,0)
```



[13]:
```
#cost function
w = 6
b = 0

y_pred = w*x + b
cost = (y - y_pred)**2
cost.sum()/len(x)
# cost.mean()
```

[13]:
```
np.float64(27936.753619832147)
```

[14]:
```
def compute_cost(x,y,w,b):
    y_pred = w*x + b
    cost = (y - y_pred)**2
    cost = cost.sum()/len(x)

    return cost
```

[15]:
```
compute_cost(x,y,6,0)
```

[15]:
```
np.float64(27936.753619832147)
```

[16]:
```
#b=0 w=100~100 cost=?

costs = []
for w in range(-100,101):
    cost = compute_cost(x,y,w,0)
    costs.append(cost)

costs
```

[16]:
```
[np.float64(98884393.15876493),
 np.float64(97005804.97023045),
 np.float64(95145235.45044343),
 np.float64(93302684.59940387),
 np.float64(91478152.4171117),
 np.float64(89671638.90356702),
 np.float64(87883144.05876975),
 np.float64(86112667.88271993),
 np.float64(84360210.37541756),
 np.float64(82625771.53686261),
 np.float64(80909351.3670551),
 np.float64(79210949.86599506),
 np.float64(77530567.03368245),
 np.float64(75868202.87011728),
 np.float64(74223857.37529953),
 np.float64(72597530.54922923),
```

```
np.float64(70989222.39190638),
np.float64(69398932.90333098),
np.float64(67826662.08350301),
np.float64(66272409.93242248),
np.float64(64736176.45008939),
np.float64(63217961.63650373),
np.float64(61717765.491665535),
np.float64(60235588.01557476),
np.float64(58771429.20823144),
np.float64(57325289.06963555),
np.float64(55897167.599787116),
np.float64(54487064.7986861),
np.float64(53094980.66633255),
np.float64(51720915.20272642),
np.float64(50364868.407867745),
np.float64(49026840.2817565),
np.float64(47706830.824392706),
np.float64(46404840.03577635),
np.float64(45120867.91590743),
np.float64(43854914.46478594),
np.float64(42606979.68241192),
np.float64(41377063.56878532),
np.float64(40165166.123906165),
np.float64(38971287.34777445),
np.float64(37795427.240390174),
np.float64(36637585.80175334),
np.float64(35497763.03186396),
np.float64(34375958.930722),
np.float64(33272173.498327494),
np.float64(32186406.734680425),
np.float64(31118658.639780797),
np.float64(30068929.213628612),
np.float64(29037218.456223864),
np.float64(28023526.36756656),
np.float64(27027852.94765669),
np.float64(26050198.19649427),
np.float64(25090562.11407929),
np.float64(24148944.70041174),
np.float64(23225345.95549164),
np.float64(22319765.879318982),
np.float64(21432204.47189376),
np.float64(20562661.733215984),
np.float64(19711137.663285647),
np.float64(18877632.26210275),
np.float64(18062145.529667296),
np.float64(17264677.465979278),
np.float64(16485228.0710387),
```

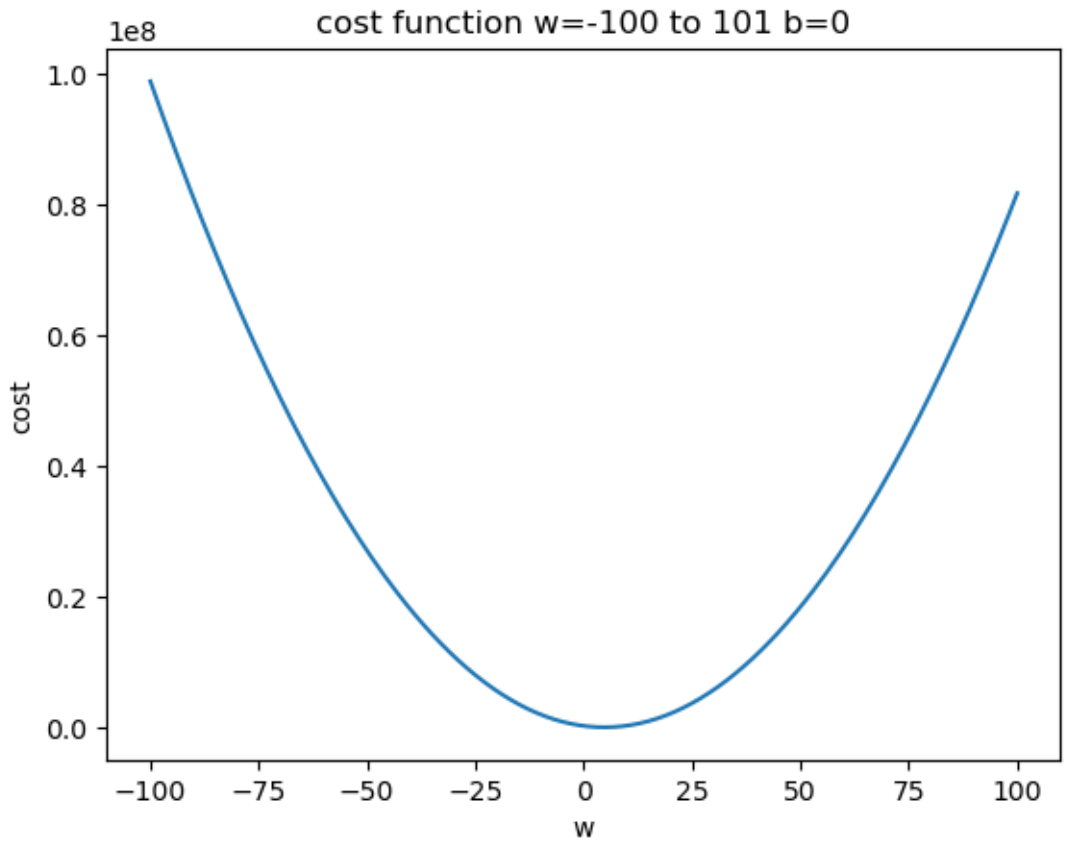
```
np.float64(15723797.344845565),
np.float64(14980385.287399875),
np.float64(14254991.898701621),
np.float64(13547617.178750811),
np.float64(12858261.127547441),
np.float64(12186923.745091513),
np.float64(11533605.031383023),
np.float64(10898304.986421973),
np.float64(10281023.610208368),
np.float64(9681760.9027422),
np.float64(9100516.864023477),
np.float64(8537291.49405219),
np.float64(7992084.792828346),
np.float64(7464896.760351943),
np.float64(6955727.39662298),
np.float64(6464576.70164146),
np.float64(5991444.67540738),
np.float64(5536331.317920739),
np.float64(5099236.62918154),
np.float64(4680160.609189782),
np.float64(4279103.257945464),
np.float64(3896064.575448588),
np.float64(3531044.5616991525),
np.float64(3184043.216697158),
np.float64(2855060.540442603),
np.float64(2544096.5329354904),
np.float64(2251151.1941758175),
np.float64(1976224.5241635859),
np.float64(1719316.522898795),
np.float64(1480427.1903814455),
np.float64(1259556.526611536),
np.float64(1056704.5315890678),
np.float64(871871.2053140402),
np.float64(705056.5477864537),
np.float64(556260.5590063079),
np.float64(425483.23897360277),
np.float64(312724.5876883386),
np.float64(217984.60515051516),
np.float64(141263.29136013263),
np.float64(82560.64631719084),
np.float64(41876.67002168995),
np.float64(19211.36247362986),
np.float64(14564.723673010589),
np.float64(27936.753619832147),
np.float64(59327.452314094524),
np.float64(108736.81975579771),
np.float64(176164.85594494175),
```

```
np.float64(261611.56088152665),
np.float64(365076.9345655523),
np.float64(486560.9769970188),
np.float64(626063.688175926),
np.float64(783585.0681022742),
np.float64(959125.1167760631),
np.float64(1152683.834197293),
np.float64(1364261.2203659634),
np.float64(1593857.275282075),
np.float64(1841471.9989456274),
np.float64(2107105.3913566205),
np.float64(2390757.4525150545),
np.float64(2692428.182420929),
np.float64(3012117.581074245),
np.float64(3349825.6484750006),
np.float64(3705552.3846231983),
np.float64(4079297.789518836),
np.float64(4471061.863161915),
np.float64(4880844.605552434),
np.float64(5308646.016690395),
np.float64(5754466.096575797),
np.float64(6218304.845208638),
np.float64(6700162.262588921),
np.float64(7200038.348716645),
np.float64(7717933.103591812),
np.float64(8253846.527214415),
np.float64(8807778.619584462),
np.float64(9379729.380701948),
np.float64(9969698.810566876),
np.float64(10577686.909179244),
np.float64(11203693.676539054),
np.float64(11847719.112646306),
np.float64(12509763.217500998),
np.float64(13189825.99110313),
np.float64(13887907.433452703),
np.float64(14604007.544549715),
np.float64(15338126.32439417),
np.float64(16090263.772986064),
np.float64(16860419.8903254),
np.float64(17648594.676412176),
np.float64(18454788.131246395),
np.float64(19279000.254828054),
np.float64(20121231.04715715),
np.float64(20981480.508233692),
np.float64(21859748.63805767),
np.float64(22756035.436629098),
np.float64(23670340.90394796),
```

```
 np.float64(24602665.040014263),
 np.float64(25553007.844828006),
 np.float64(26521369.318389192),
 np.float64(27507749.46069782),
 np.float64(28512148.27175388),
 np.float64(29534565.75155739),
 np.float64(30575001.90010834),
 np.float64(31633456.71740672),
 np.float64(32709930.203452557),
 np.float64(33804422.35824583),
 np.float64(34916933.18178654),
 np.float64(36047462.6740747),
 np.float64(37196010.83511029),
 np.float64(38362577.66489332),
 np.float64(39547163.1634238),
 np.float64(40749767.33070171),
 np.float64(41970390.16672707),
 np.float64(43209031.67149986),
 np.float64(44465691.8450201),
 np.float64(45740370.687287785),
 np.float64(47033068.198302895),
 np.float64(48343784.37806547),
 np.float64(49672519.226575464),
 np.float64(51019272.74383291),
 np.float64(52384044.9298378),
 np.float64(53766835.78459012),
 np.float64(55167645.308089875),
 np.float64(56586473.500337094),
 np.float64(58023320.36133174),
 np.float64(59478185.89107382),
 np.float64(60951070.08956334),
 np.float64(62441972.95680033),
 np.float64(63950894.49278473),
 np.float64(65477834.69751657),
 np.float64(67022793.57099587),
 np.float64(68585771.11322258),
 np.float64(70166767.32419679),
 np.float64(71765782.2039184),
 np.float64(73382815.75238743),
 np.float64(75017867.96960394),
 np.float64(76670938.85556787),
 np.float64(78342028.41027924),
 np.float64(80031136.63373807),
 np.float64(81738263.52594432)]
```

[16]:
```
plt.plot(range(-100,101),costs)
plt.title("cost function w=-100 to 101 b=0")
```

```
plt.xlabel("w")
plt.ylabel("cost")
plt.show()
```

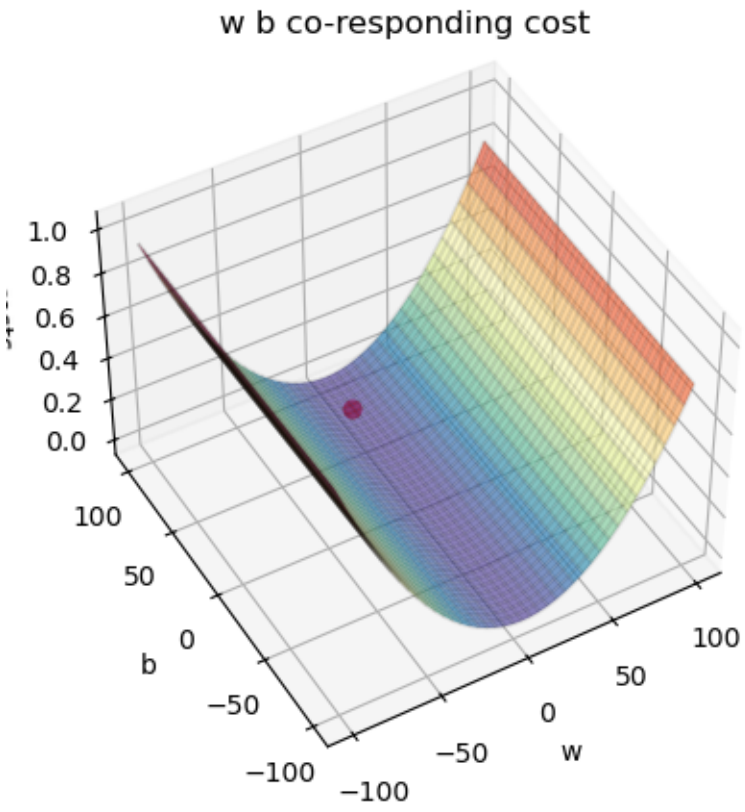


[17]:
```
import numpy as np
```

[18]:
```
ws = np.arange(-100,101)
bs = np.arange(-100,101)
costs = np.zeros((201,201))

i = 0
for w in ws:
    j = 0
    for b in bs:
        cost = compute_cost(x,y,w,b)
        costs[i,j] = cost
        j = j + 1
    i = i + 1

costs
```

[18]:
```
array([[1.00716630e+08, 1.00698208e+08, 1.00679789e+08, …,
        9.71082054e+07, 9.70901800e+07, 9.70721567e+07],
       [9.88206613e+07, 9.88024137e+07, 9.87841682e+07, …,
        9.52466498e+07, 9.52287982e+07, 9.52109487e+07],
       [9.69427116e+07, 9.69246378e+07, 9.69065661e+07, …,
        9.34031128e+07, 9.33854351e+07, 9.33677593e+07],
       …,
       [7.67329891e+07, 7.67489805e+07, 7.67649739e+07, …,
        7.99382909e+07, 7.99546783e+07, 7.99710677e+07],
       [7.84047172e+07, 7.84208824e+07, 7.84370496e+07, …,
        8.16444317e+07, 8.16609929e+07, 8.16775561e+07],
       [8.00944639e+07, 8.01108029e+07, 8.01271439e+07, …,
        8.33685912e+07, 8.33853262e+07, 8.34020632e+07]])
```

[33]:
```
ax = plt.axes(projection = "3d")
ax.view_init(45,-120)
b_grid, w_grid = np.meshgrid(bs,ws)
ax.plot_surface(w_grid,b_grid,costs,cmap="Spectral_r",alpha=0.7)
ax.plot_wireframe(w_grid,b_grid,costs,color="black",alpha=0.1)

ax.set_title("w b co-responding cost")
ax.set_xlabel("w")
ax.set_ylabel("b")
ax.set_zlabel("costs")

w_index, b_index = np.where(costs == np.min(costs))
ax.scatter(ws[w_index],bs[b_index],costs[w_index,b_index],color="red",s=40)

plt.show()
print(f"when w={ws[w_index]},b={bs[b_index]} will appear min cost:
  ↪{costs[w_index,b_index]}")
```




```
when w=[4],b=[73] will appear min cost:[13811.0357929]
```

[ ]: