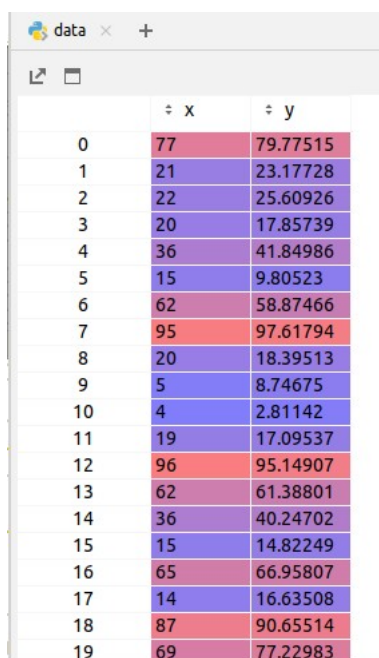


سؤال (۱)

۱. اطلاعات این بخش در فایل q1_a_2 قرار دارد.
(بخش ۱-۱)

ابتدا داده‌ها را با استفاده از دستور read_csv() خوانده و در data frame به اسم data ذخیره می‌کنیم.

```
##%% read data
data = pd.read_csv("test.csv")
```



	x	y
0	77	79.77515
1	21	23.17728
2	22	25.60926
3	20	17.85739
4	36	41.84986
5	15	9.80523
6	62	58.87466
7	95	97.61794
8	20	18.39513
9	5	8.74675
10	4	2.81142
11	19	17.09537
12	96	95.14907
13	62	61.38801
14	36	40.24702
15	15	14.82249
16	65	66.95807
17	14	16.63508
18	87	90.65514
19	69	77.22983

دیتا فریم ذخیره شده به شکل زیر می باشد:

تحلیل داده:

ستون اول مقدار متغیر X

ستون دوم مقدار متغیر Y

با استفاده از تابع خطای حداقل مربعات بهترین خط که این داده‌ها را توصیف کند به دست می‌آوریم.

۲ روش متداول محاسبه تابع خطا در ادامه توضیح داده می‌شود.

خطای حداقل مربعات اختلافات MSE یا Mean Square Error:

در این روش تابع هزینه J را به این صورت تعریف می‌کنیم که میانگین مربعات اختلاف مقادیر y را برای تابع پیشبینی h محاسبه می‌کند و با به حداقل رساندن این مقدار، هزینه را کاهش می‌دهد.
در این سؤال چون رگرسیون خطی داریم تابع h به صورت زیر می باشد:

$$h(\theta) = \theta_0 + \theta_1 \times x$$

در این روش به خطاهای بزرگ به دلیل به توان دو رسیدن وزن زیادی می‌دهد و اگر نسبت به داده‌های دور تر حساس هستیم و می‌خواهیم در محاسبه رگرسیون تأثیر گذار باشند، این روش مناسب است.

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(X^{(i)}) - Y^{(i)})^2$$

خطای قدرمطلق اختلافات MAE یا Mean Absolute Error:

در این روش هم مانند روش قبل عمل می‌کنیم ولی به جای توان دوقدر مطلق داریم. درواقع در این روش مقدار میانگین مجموع قدرمطلق اختلاف‌های بین مقادیر پیش‌بینی شده و مقدار اصلی را محاسبه می‌کنیم. در این روش به ابزارهای پیشرفته‌تری برای محاسبه رگرسیون خطی نیاز داریم. این روش تأثیر داده‌های پرت را به دلیل اینکه به توان دو نمی‌رساند، کمتر می‌کند. فرمول این تابع خطا به صورت زیر می‌شود:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m |h_{\theta}(X^{(i)}) - Y^{(i)}|$$

در این تمرین از روش اول استفاده کردم و تابع هزینه را به صورت زیر تعریف کردم که معادل با همان فرمول ذکر شده در بالا است.

```
## cost function MSE
def cost_function(X, y, theta):
    m = len(y)
    h = np.dot(X, theta)
    error = h - y
    cost = (1/(2*m)) * np.sum(np.power(error,2))
    return cost
```

در اینجا مقادیری که در ادامه برای گرادینان کاهش می‌دهیم به آن‌ها نیاز داریم را set می‌کنیم:

```
m = len(data['x'])
x = (np.array(data['x'])).reshape(m,1)
y = (np.array(data['y'])).reshape(m,1)
X = np.insert(x, 0, np.ones([m]), axis=1)
theta = np.zeros((2,1))
```

m: تعداد داده‌های آموزشی

x: ستون 0 یا همان ستون اول در داده‌ها که آن را به فرمت ndarray(300,1) در می‌آوریم.
y: ستون 1 یا همان ستون دوم در داده‌ها که آن را به فرمت ndarray(300,1) در می‌آوریم.
X: یک ستون یک در اندیس ستون صفر آرایه x اضافه می‌کنیم و این متغیر دارای فرمت ndarray(300,2) می‌باشد که ستون صفر آن اعداد یک می‌باشد.

	÷ 0	÷ 1		÷ 0	Y
0	1	77	0	79.77515	
1	1	21	1	23.17728	
2	1	22	2	25.60926	
3	1	20	3	17.85739	
4	1	36	4	41.84986	
5	1	15	5	9.80523	
6	1	62	6	58.87466	
7	1	95	7	97.61794	
8	1	20	8	18.39513	
9	1	5	9	8.74675	
10	1	4	10	2.81142	
11	1	19	11	17.09537	
12	1	96	12	95.14907	
13	1	62	13	61.38801	
14	1	36	14	40.24702	
15	1	15	15	14.82249	
16	1	65	16	66.95807	
17	1	14	17	16.63508	
18	1	87	18	90.65514	
19	1	69	19	77.22983	

در این حلقه iteration های مورد نیاز برای آپدیت کردن theta انجام می شود.
برای آپدیت کردن همزمان theta از فرمول زیر استفاده می کنیم:

$$\theta_j := \theta_j - \alpha \frac{\partial J(\theta_0, \theta_1)}{\partial \theta_j}$$

که به جای مقدار $\frac{\partial J}{\partial \theta_j}$ مقدار مشتق تابع هزینه را قرار داده ام.

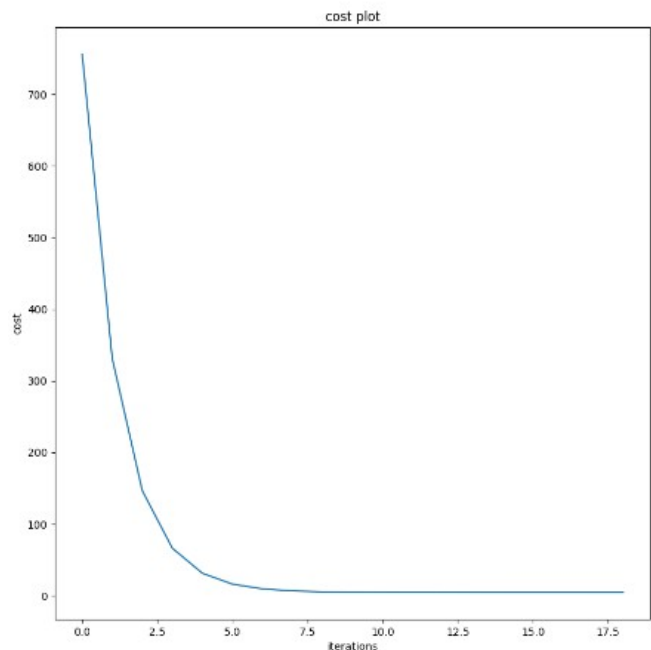
```
prev_cost= 0
tolerance= 100000
costs= []
while(abs(tolerance)>0.0001):
    h = np.dot(X, theta)
    theta = theta - 0.0001/m * np.dot(X.T,h-y)
    cost= cost_function(X,y,theta)
    costs.append(cost)
    tolerance= abs(cost-prev_cost)/cost
    prev_cost= cost;
```

یکی از شرط های پایان حلقه استفاده از فرمول روبه رو است:
به این معنی که اگر اختلاف تابع هزینه در دو مرحله پشت سر هم نسب به تابع هزینه در این مرحله از یک مقداری کوچک تر بود ایتريشن ها را متوقف کند. درواقع نشان دهنده همگرایی تابع هزینه می باشد.

$$\frac{J(\theta)^{(i)} - J(\theta)^{(i-1)}}{J(\theta)^{(i)}} < 10^{-4}$$

راه حل دیگر تشخیص همگرایی رسم تابع هزینه است که با دستور زیر تابع هزینه را بر حسب iteration ها رسم کردم و بر اساس این نمودار مقدار الفا را تا حد امکان بزرگ در نظر گرفتم.

```
##% print cost
plt.figure(figsize=(10, 10))
plt.title("cost plot")
plt.plot(list(range(len(costs))),costs)
plt.xlabel("iterations")
plt.ylabel("cost")
plt.show()
```



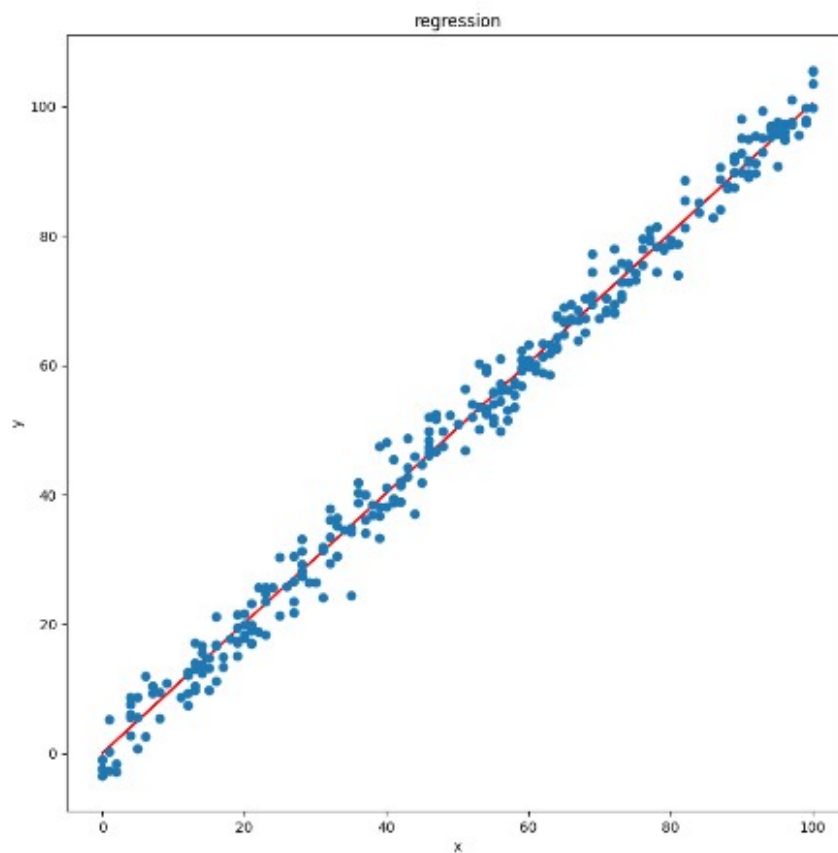
همان طور که مشاهده می شود مقدار هزینه در ۱۷ تکرار به سمت صفر میل میکند و همگرا شده است.

بخش ۲-۱)
مقدار کلی خطا: 4.609400193042552

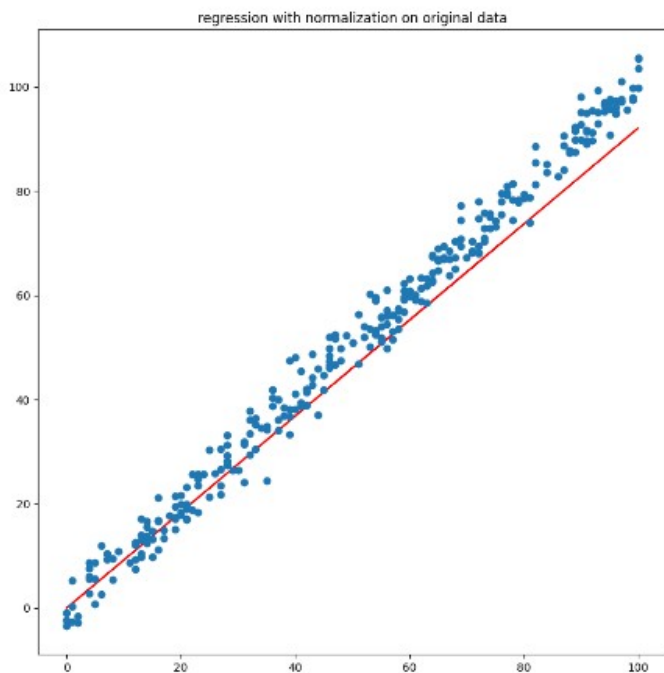
```
### total cost  
total_cost= cost_function(X,y,theta)
```

بخش ۳-۱)
سپس نقاط و تابع رگرسیون را رسم می کنیم:

```
### plot linear regression on data  
plt.figure(figsize=(10, 10))  
plt.title("regression")  
plt.xlabel("x")  
plt.ylabel("y")  
plt.plot(x, h, 'r')  
plt.plot(x, y, 'o')  
plt.show()
```



توضیحات زیر در فایل q1_a_1 قرار دارند.
برای این سؤال ابتدا داده‌ها را نرمال کردم و رگرسیون را محاسبه کردم ولی میزان کلی خطا برابر با 17.35905004521532 شد و خط از داده‌ها انحراف داشت.



دلیل این اتفاق این بود که داده‌ها در یک اسکیل قرار داشتند و نرمال کردن در این شرایط دقت را کاهش می‌دهد و برای مواقعی که داده‌ها خودشان توزیع نرمال دارند، استاندارد سازی مناسب‌تر است.

بخش ۱-۴)

فایل ها: q1_b2

در این بخش به جای مقادیر x مقادیر $x*5$ را قرار دادم.

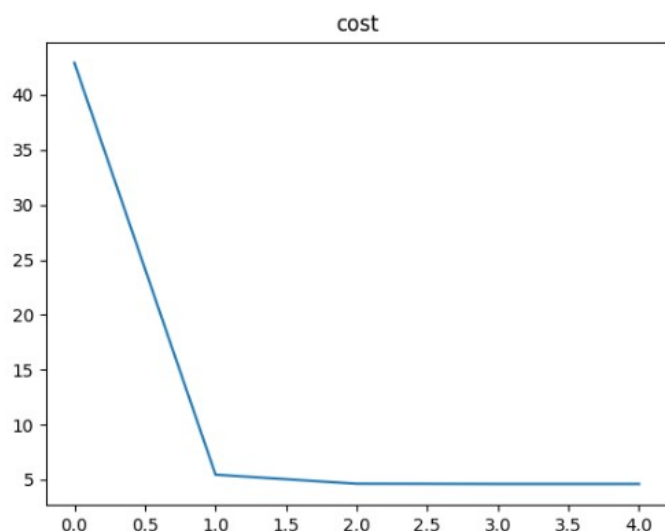
```
data['x'] = data['x']*5
```

همان طور که مشاهده می شود مقادیر ستون x مقدار ۵ برابر قبل دارند:

	÷ 0	÷ 1
0	1	385
1	1	105
2	1	110
3	1	100
4	1	180
5	1	75
6	1	310
7	1	475
8	1	100
9	1	25
10	1	20
11	1	95
12	1	480
13	1	310
14	1	180
15	1	75
16	1	325
17	1	70
18	1	435
19	1	345
20	1	445

بقیه مراحل دقیقاً مشابه قبل تکرار می شوند.

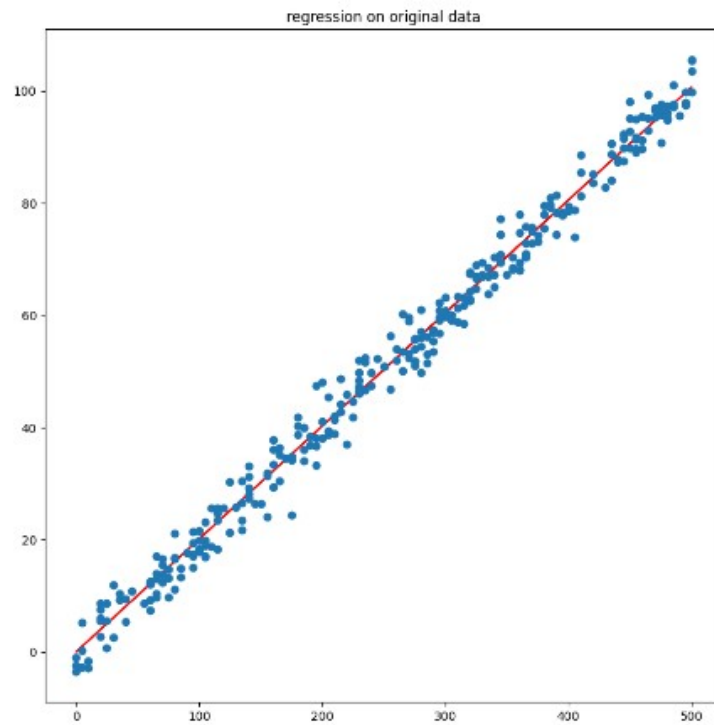
در این حالت برای همگرایی بهتر مقدار learning rate را باید کمتر قرار می دادم. در بخش قبل 0.0001 و در این بخش 0.00001 است. ولی در حالت کلی با تعداد تکرار کمتر تابع هزینه همگرا شد.



همان طور که مشاهده می شود فقط در ۴ تکرار همگرا شده است.

هزینه نهایی:
هزینه نهایی برابر با 4.607583971241822 شده است.

رسم نقاط و تابع رگرسیون:



بخش ۵-۱)

در فایل q1_c_2 قرار دارد.

در این بخش یک ستون جدید به ویژگی‌ها اضافه می‌کنیم و مقدار آن را برابر با X^2 قرار می‌دهیم. متغیر X که در ستون صفر مقدار یک و در ستون ۲ مقدار x^2 اضافه شده است.

```
### make needed data
```

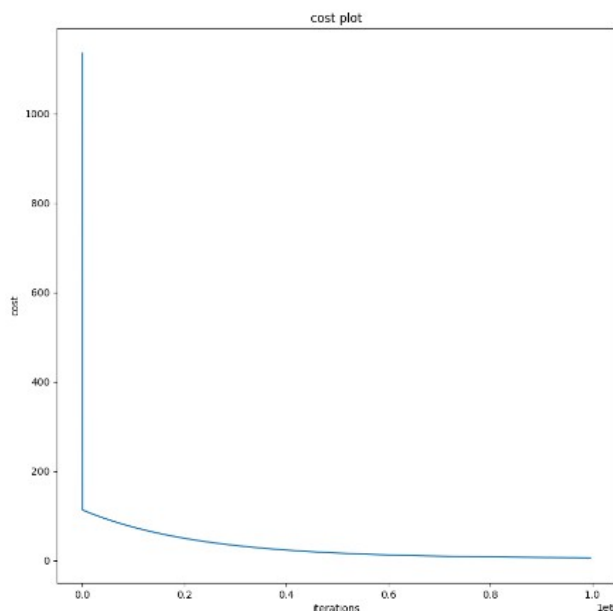
```
m = len(data['x'])
x = (np.array(data['x'])).reshape(m,1)
y = (np.array(data['y'])).reshape(m,1)
x_2 = (np.array(np.power(data['x'],2))).reshape(m,1)
X = np.insert(x, 0, np.ones([m]), axis=1)
X = np.append(X, x_2, axis=1)
```

	÷ 0	÷ 1	÷ 2
0	1	77	5929
1	1	21	441
2	1	22	484
3	1	20	400
4	1	36	1296
5	1	15	225
6	1	62	3844
7	1	95	9025
8	1	20	400
9	1	5	25
10	1	4	16
11	1	19	361
12	1	96	9216
13	1	62	3844
14	1	36	1296
15	1	15	225
16	1	65	4225
17	1	14	196
18	1	87	7569
19	1	69	4761

روش گرادیان کاهشی در این بخش مشابه قبل است. برای همگرا شدن مقدار الفا را خیلی باید کوچک‌تر در نظر بگیریم.

از طرفی برای اینکه هزینه نهایی کاهش یابد باید دقت را افزایش دهیم. فکر می‌کنم در این صورت overfitting رخ دهد. به هر حال در صورت افزایش دقت، تعداد تکرارها بسیار زیاد می‌شود و تابع هزینه به این شکل است:

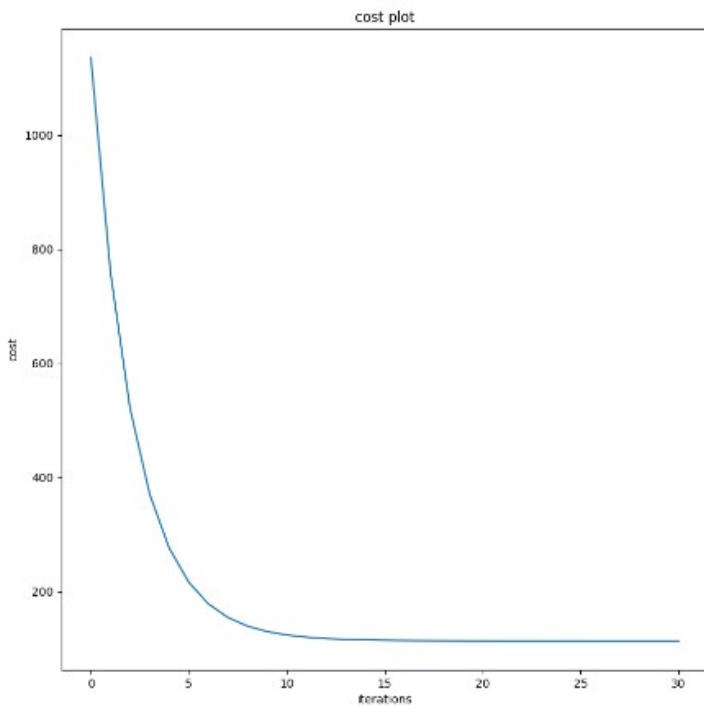
هزینه نهایی در این حالت بسیار کم و نزدیک به صفر است.



ولی به دلیل overfit نبودن مدل دقت را کاهش دادم و به این صورت شد:

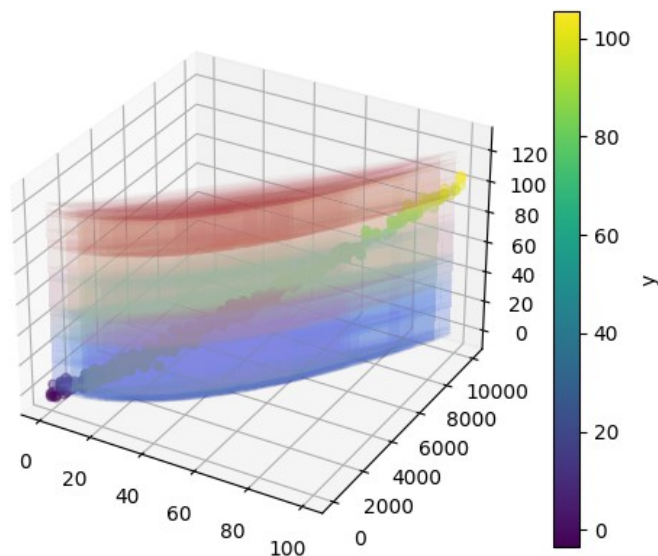
در ۳۰ تکرار به میزان خوبی هزینه را کاهش داده است.

و هزینه نهایی برابر شد با :
113.15064258358937



به دلیل اینکه سه مقدار برای ضرایب تابع پیشبینی داریم. مدل را به صورت سه بعدی باید پرینت کنیم:

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
pnt3d= ax.scatter(X[:,1],X[:,2],y, c=y)
ax.plot_surface(X[:,1],X[:,2],h, cmap='coolwarm',linewidth=0, alpha=0.01)
cbar=plt.colorbar(pnt3d)
cbar.set_label("y")
plt.show()
```



بخش ۶-۱)

در این بخش با استفاده از همان تابع خطا MSE رگرسیون خطی را با استفاده از روش Normal Equation به دست می آوریم:

در این روش داریم:

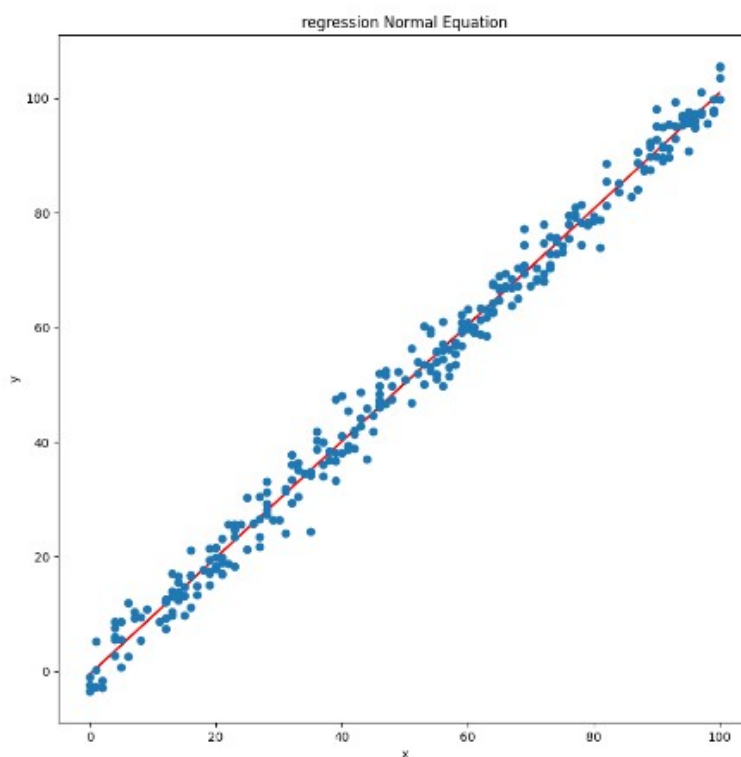
$$\theta_{(n+1)*1} = (X^T X)^{-1} X^T Y$$

```
### Normal Equation
m= len(data['x'])
x= (np.array(data['x'])).reshape(m,1)
y= (np.array(data['y'])).reshape(m,1)
X = np.insert(x, 0, np.ones([m]), axis=1)

XTX= np.linalg.inv(np.dot(X.T, X))
theta= np.linalg.multi_dot([XTX, X.T, y])
```

با استفاده از روابط بالا ضرایب را به دست می آوریم و مقدار هزینه نهایی را محاسبه می کنیم.
هزینه نهایی: 4.582143930307135

نمودار:



همان‌طور که مشخص است با این روش هزینه نهایی مشابه روش گرادیان کاهشی است و نشان دهنده صحت محاسبات است.

در روش Normal Equation چند نکته وجود دارد.

(۱) حتی اگر داده‌ها در یک اسکال نباشند نیازی به نرمال سازی نیست.

(۲) این روش ممکن است پاسخی نداشته باشد یا استفاده از آن بهینه نباشد.

- اگر در این روش تعداد ویژگی‌ها بسیار بیشتر از تعداد نمونه‌های آموزشی باشد سرعت همگرایی بسیار پایین است. و علت آن محاسبه معکوس ماتریس XTX است. زیرا محاسبه معکوس ماتریس در اوردر $O(n^3)$ می باشد.

- اگر ماتریس XTX معکوس پذیر نباشد یا singular باشد و دترمینان نزدیک به صفر داشته باشد باز هم این روش مناسب نیست. (البته دستور هایی وجود دارند که با وجود این هم معکوس را به ما می دهند مثلاً در متلب دستور pinv این کار را انجام می دهد.)

- در صورتی که ویژگی‌ها مستقل خطی نباشند باعث singular شدن ماتریس می شوند.

- اگر تعداد نمونه‌های آموزشی از ویژگی‌ها بسیار کم تر باشد باعث overfitting می شود و در این صورت باید از روش‌هایی مانند regularization یا حذف کردن تعدادی از ویژگی‌ها استفاده کنیم.

سؤال (۲)

بخش ۲-۱)

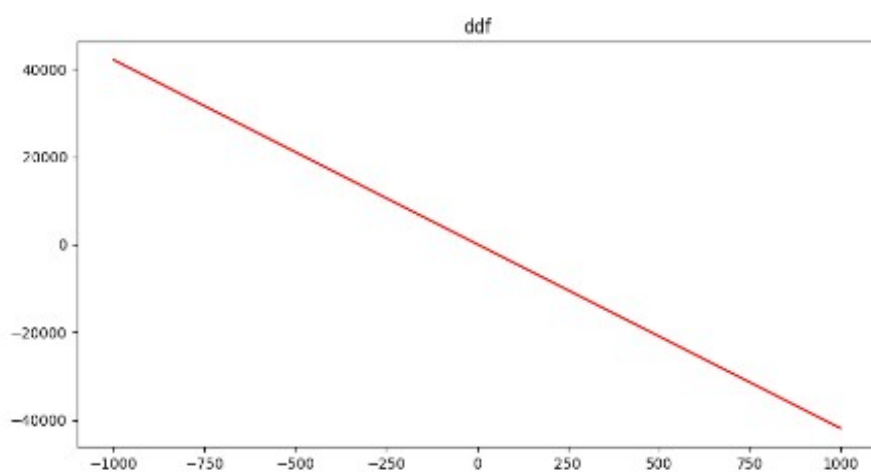
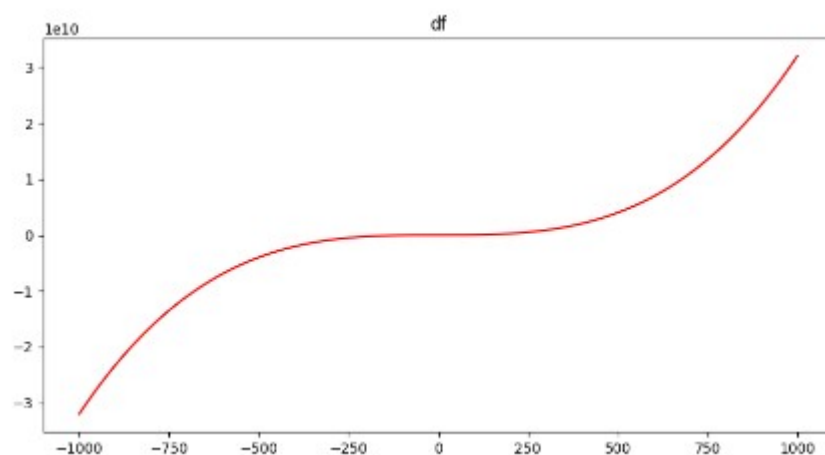
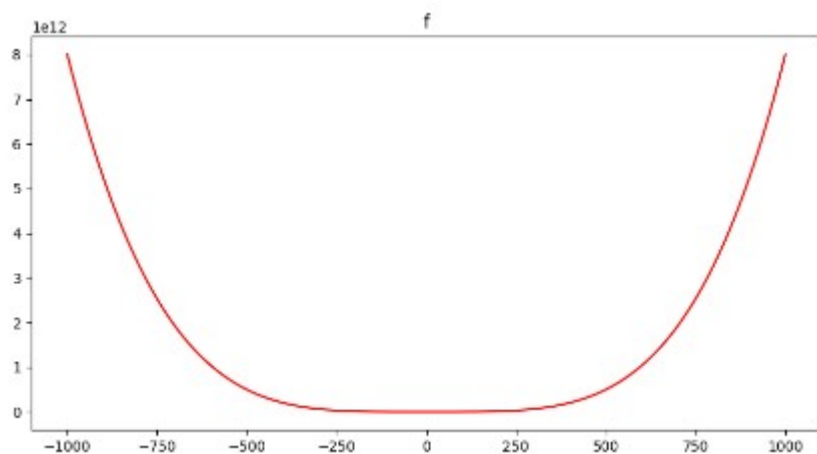
فایل q2_newton:

در این سؤال برای آشنا شدن با تابع و داده‌های به دست آمده ابتدا همه تابع‌ها را رسم می‌کنیم.

```
f = lambda x: x**2 - 7*x**3 + 8*x**4 - 12
```

```
Df = lambda x: 2*x - 21*x**2 + 32*x**3
```

```
DDf= lambda x: 2 - 42*x + 96
```



روش نیوتون یکی از روش‌های جایگزین برای گرادیان کاهشی است که سرعت همگرایی بیشتری دارد.

برای به دست آوردن ریشه تابع $(J(\theta))$ روش نیوتون به صورت زیر پیاده‌سازی می‌شود:

$$\frac{\partial J(\theta)}{\partial \theta} = 0$$

برای به دست آوردن max یا min یک تابع باید ریشه مشتق آن را به دست آوریم بنابر این داریم:

$$\max(f(\theta)) \equiv \theta = \theta - \frac{f'(\theta)}{f''(\theta)}$$

این محاسبات برای مقادیر عددی می‌باشد و در صورتی که ورودی برداری باشد باید از روش نیوتون رافسون استفاده کنیم که به صورت زیر می‌باشد:

$$\theta := \theta - H^{-1} \nabla_{\theta} (f(\theta))$$

H تابع Hessian می‌باشد که به صورت زیر تعریف می‌شود:

$$H_{ij} = \frac{\partial^2 f(\theta)}{\partial \theta_i \partial \theta_j}$$

$$\nabla_{\theta} f(\theta) = \left(\frac{\partial f}{\partial \theta_0}, \frac{\partial f}{\partial \theta_1}, \dots, \frac{\partial f}{\partial \theta_n} \right)$$

با توجه به توضیحات بالا برای به دست آوردن min در تابع f باید ریشه را در تابع f' به دست بیاوریم. بنابراین با توجه به نمودار مشخص است که جواب در حدود صفر باید باشد.

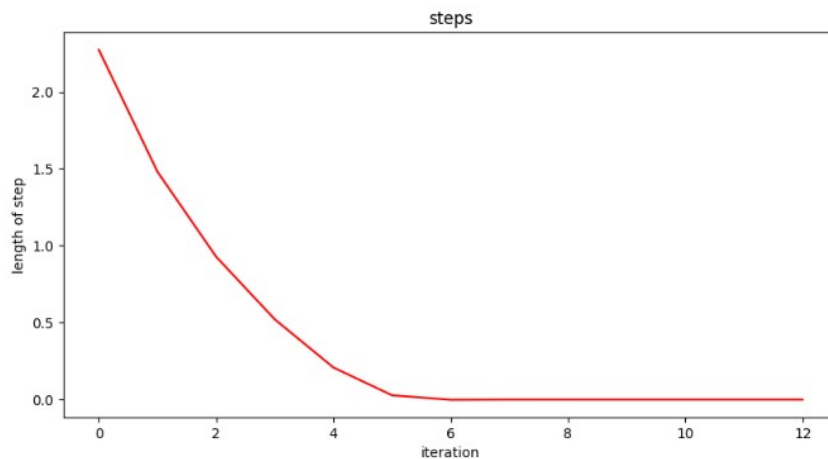
برای به دست آوردن ریشه f' در هر مرحله طول گام به اندازه $\frac{f'(\theta)}{f''(\theta)}$ است. یعنی در هر مرحله

شیب را روی f'' به دست می‌آوریم و به آن اندازه به سمت ریشه حرکت می‌کنیم. یعنی شیب بر روی نمودار f'' انقدر باید کم شود تا به صفر نزدیک شود. بنابر این شرط خاتمه الگوریتم را می‌توان نزدیک شدن طول گام به صفر در نظر گرفت.

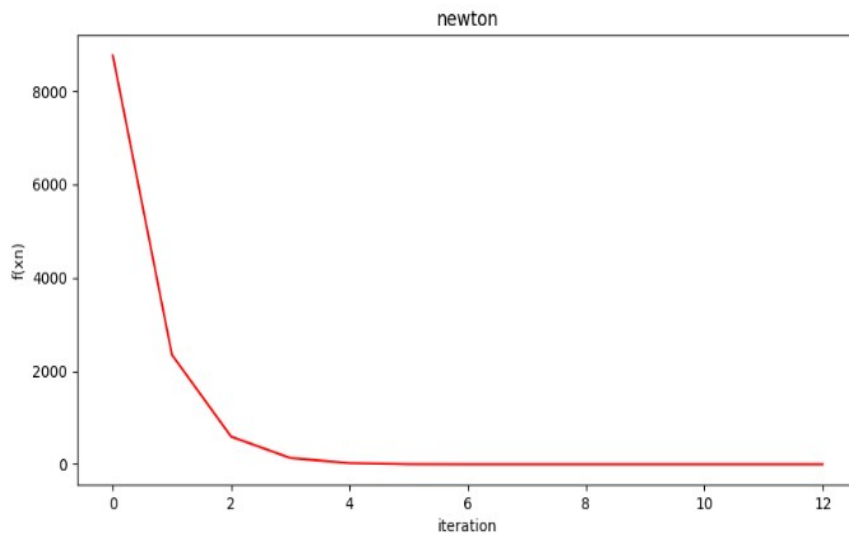
بنابراین روش نیوتون را به این صورت پیاده‌سازی می‌کنیم:

```
def newton(f,Df,epsilon):
    xn = 10
    steps=[]
    func=[]
    while(True):
        if(abs(f(xn)/Df(xn)) < epsilon):
            return xn,steps,func
        xn = xn - (f(xn)/Df(xn))
        steps.append(f(xn)/Df(xn))
        func.append(f(xn))
    return None,None,None
```

رسم طول گام در هر تکرار:
همان‌طور که گفته شد در هر مرحله گام حرکت یعنی شیب کاهش می‌یابد و انقدر کم می‌شود که به صفر نزدیک شود.
تعداد تکرارها با توجه به مقدار اولیه متفاوت خواهد بود ولی برای این مقدار اولیه ۱۲ تکرار داریم.



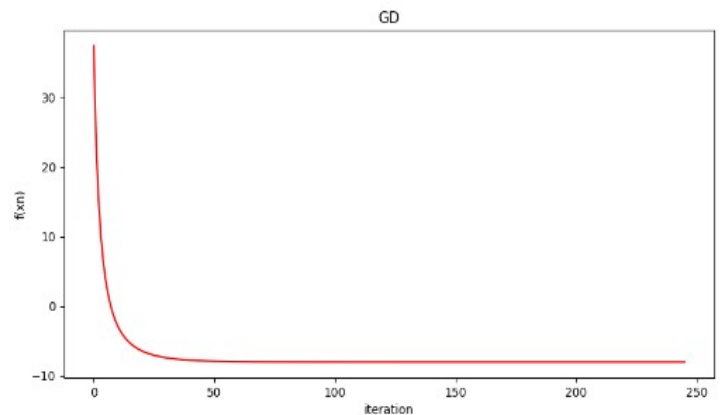
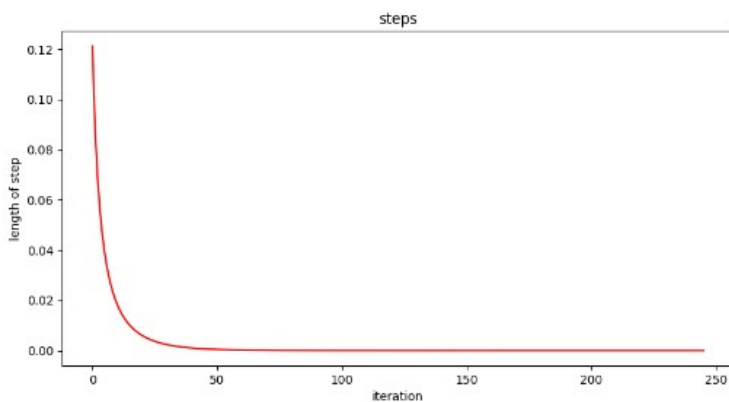
رسم مقدار تابع در هر تکرار:
مقدار تابع در هر تکرار کاهش می‌یابد تا به کمترین مقدار خود برسد.



فایل q2_gd:
با استفاده از گرادیان کاهشی:

```
def gradian_descent(f,Df,epsilon):  
    xn = 2  
    prev= 0  
    steps=[]  
    func=[]  
    tolerance = 100000  
    while (abs(tolerance) > epsilon):  
        xn = xn - 0.001 * Df(xn)  
        tolerance = abs(f(xn) - prev) / f(xn)  
        prev = f(xn)  
        steps.append(0.001 * abs(Df(xn)))  
        func.append(f(xn))  
    return xn,steps,func
```

با توجه به نمودار تعداد تکرار ها در این روش با همان دقت قبلی، بسیار بیشتر از روش نیوتون است. همچنان مانند قبل در هر مرحله گام کاهش می یابد. خروجی در هر دو حال مقدار یکسان دارند.



هنگامی که تعداد ویژگی‌ها کم است، روش نیوتون مناسب‌تر است زیرا در گرادیان کاهشی باید معکوس ماتریس محاسبه شود که از اوردر $O(n^3)$ می باشد.

روش Fisher scoring معادل روش نیوتون برای داده‌های طبقه بندی شده و logistic regression می باشد. روش Fisher scoring فرمی مشابه روش نیوتون دارد با این تفاوت که به جای اینکه به مشتق دوم نگاه کند، به Expected value مشتق دوم وابسته است:

$$\max(f(\theta)) \equiv \theta := \theta - \frac{f'(\theta)}{E[f''(\theta)]}$$

مزیت این روش این است که مقدار E حتماً مثبت است.
 هنگامی روش newton raphson و Fisher Scoring مشابه هم عمل می کنند که توزیع داده ها به صورت نمایی کرونیكال باشد.
 اثبات:
 اگر f از خانواده نمایی باشد دارای فرمت زیر است:

$$f(x) = \exp\left(\frac{\eta(\theta(x))x - b(\theta(x))}{a(\phi)} + c(x, \phi)\right)$$

حالت canonical زمانی اتفاق می افتد که $\eta(\theta) = \theta$ باشد.

بنابر این داریم:

$$f(x) = \exp\left(\frac{\theta(x)x - b(\theta(x))}{a(\phi)} + c(x, \phi)\right)$$

از رابطه بالا \log می گیریم:

$$\frac{\partial \log(f)}{\partial \theta} = \frac{x - b'(\theta(x))}{a(\phi)}$$

$$\frac{\partial^2 \log(f)}{\partial \theta^2} = \frac{-b''(\theta(x))}{a(\phi)}$$

سوال ۳)

بخش ۱-۳)

داده‌ها را ورودی می‌گیریم و داده آموزشی را از داده تست جدا می‌کنیم. ستون ۹ و ۱۰ که ستون هدف هستند را به عنوان y برای هر بخش در نظر می‌گیریم.

```
### read data
data = pd.read_csv('ENB2012_data.csv')
data_normal = (data.max()-data)/(data.max() - data.min());

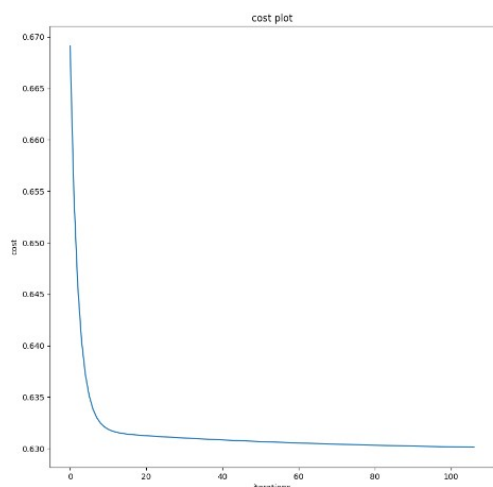
### parse input
data_train= np.array(data_normal.iloc[:600,:])
m=len(data_train)
x_train= data_train[:,0:8]
X_train= np.insert(x_train, 0, np.ones([m]), axis=1)

y_train= data_train[:,8:]

data_test= np.array(data_normal.iloc[600,:])
x_test= data_test[:, 0:8]
y_test= data_test[:, 8:]
```

از گرادینان کاهشی و تابع خطا MSE که در سؤال اول توضیح داده شد، استفاده می‌کنم.

```
### gradient descent
theta= np.zeros((9,1))
prev_cost= 0
tolerance= 100000
costs= []
while(abs(tolerance)>0.0001):
    h = np.dot(X_train, theta)
    theta = theta - 0.00001/m * np.dot(X_train.T,h-y_train)
    cost= cost_function(X_train,y_train,theta)
    costs.append(cost)
    tolerance= abs(cost-prev_cost)/cost
    prev_cost= cost;
```



می‌بینیم که تابع هزینه همگرا شده است.

هزینه نهایی برابر با:
همان‌طور که انتظار داشتیم هزینه برای داده‌های تست بیشتر شده است.

```
### total cost
total_cost_train= cost_function(X_train,y_train,theta)
total_cost_test= cost_function(X_test,y_test,theta)

total_cost_test = {float64} 0.2645280064148638
total_cost_train = {float64} 0.22570718259641742
```

تتا های به دست آمده برابر است با:

```
theta
array([[8.54463525e-06, 7.24088123e-06],
       [4.47795875e-06, 3.90526769e-06],
       [3.42834474e-06, 2.78247654e-06],
       [4.31687246e-06, 3.68491736e-06],
       [3.11236668e-06, 2.45300660e-06],
       [6.83652129e-06, 6.04191499e-06],
       [4.26571510e-06, 3.64612262e-06],
       [4.51218413e-06, 3.78601912e-06],
       [4.43827956e-06, 3.71472961e-06]])
```

بخش ۲-۳
داده‌های اعتبار سنجی بخشی از داده های آموزشی هستند که در ایجاد مدل شرکت داده نمی‌شوند و از آن‌ها برای تخمین اعتبار مدل در حین ایجاد مدل است.

بخش ۳-۳

بخش ۴-۳
یکی از روش‌های تشخیص ویژگی‌های مؤثر محاسبه corr یا کرلیشن بین آن ویژگی و ستون هدف می باشد. از این طریق می‌توان متوجه شد که کدام ویژگی با ستون هدف همبستگی بیشتری دارند و هر ویژگی که مقدار corr آن به یک نزدیک‌تر بود یعنی مؤثر تر بوده است. روش دیگر انتخاب داده‌های مؤثر بر اساس تایپ داده ورودی و داده هدف می باشد.

بخش ۵-۳

```
###
crr= data.corr(method='pearson')
```

با توجه به این دستور کورلیشن بین هر دو ستون گرفته می‌شود که در اینجا فقط با دو ستون آخر برای ما اهمیت دارد.

با توجه به جدول زیر متوجه می‌شویم که در ستون Y1 بیشترین همبستگی با ویژگی X5 می‌باشد و در ستون Y2 هم بیشترین همبستگی با ویژگی X5 می‌باشد.
ولی در کل ویژگی‌های X1, X2, X4, X5 همبستگی نسبتاً خوبی با ستون هدف دارند.

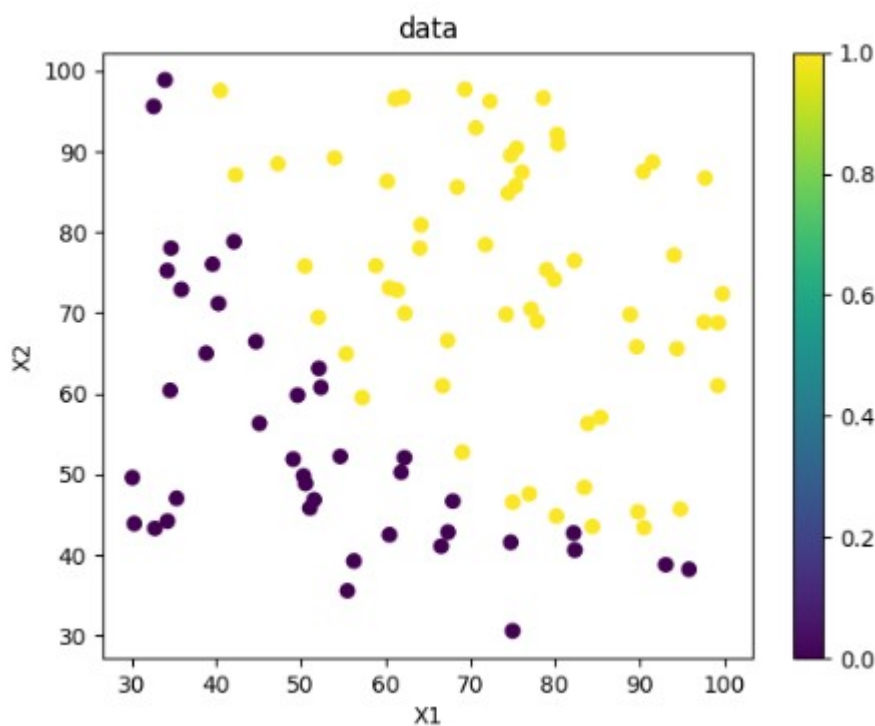
	÷ X2	÷ X3	÷ X4	÷ X5	÷ X6	÷ X7	÷ X8	÷ Y1	÷ Y2
X1	-0.99190	-0.20378	-0.86882	0.82775	0.00000	-0.00000	-0.00000	0.62227	0.63434
X2	1.00000	0.19550	0.88072	-0.85815	-0.00000	0.00000	0.00000	-0.65812	-0.67300
X3	0.19550	1.00000	-0.29232	0.28098	-0.00000	-0.00000	0.00000	0.45567	0.42712
X4	0.88072	-0.29232	1.00000	-0.97251	-0.00000	-0.00000	-0.00000	-0.86183	-0.86255
X5	-0.85815	0.28098	-0.97251	1.00000	0.00000	0.00000	-0.00000	0.88943	0.89579
X6	-0.00000	-0.00000	-0.00000	0.00000	1.00000	-0.00000	-0.00000	-0.00259	0.01429
X7	0.00000	-0.00000	-0.00000	0.00000	-0.00000	1.00000	0.21296	0.26984	0.20750
X8	0.00000	0.00000	-0.00000	-0.00000	-0.00000	0.21296	1.00000	0.08737	0.05053
Y1	-0.65812	0.45567	-0.86183	0.88943	-0.00259	0.26984	0.08737	1.00000	0.97586
Y2	-0.67300	0.42712	-0.86255	0.89579	0.01429	0.20750	0.05053	0.97586	1.00000

سؤال ۴)

بخش ۱-۴)

```
### import data
mat = scipy.io.loadmat('data_logistic.mat')
data = mat['logistic_data']

### plot data
plt.title("data")
plot=plt.scatter(data[:,0],data[:,1],c=data[:,2])
cbar=plt.colorbar(plot)
plt.xlabel('X1')
plt.ylabel('X2')
plt.show()
```



رنگ زرد به معنی داده ۱ یعنی وجود بیماری و رنگ آبی به معنی عدم وجود بیماری است.
با توجه به نمودار برای این نوع داده مرز تصمیم گیری خطی داریم.

بخش ۲-۴)
برای Y های گسسته از logistic regression استفاده می کنیم.

$$h_{\theta}(x) = g(\theta^T x) = \frac{1}{1 + e^{-(\theta^T x)}}$$

می دانیم مجموع احتمال بیمار بودن و سالم بودن، یک می شود بنابراین داریم:

$$P(y=1, x; \theta) + P(y=0, x; \theta) = 1$$

حال مقدار تابع پیش بینی با یک threshold مثلاً نیم مقایسه می شود:

$$\begin{aligned} h_{\theta}(x) \geq 0.5 (ths) &\rightarrow y=1 & \theta^T X \geq 0 &\rightarrow y=1 \\ h_{\theta}(x) \leq 0.5 (ths) &\rightarrow y=0 & \theta^T X \leq 0 &\rightarrow y=0 \end{aligned}$$

تابع هزینه برابر است با:

$$\text{cost}(h_{\theta}, y) = \frac{-1}{m} \sum_{i=1}^m (y^{(i)} \log(h_{\theta}(X^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(X^{(i)})))$$

و در نتیجه برای گرادینان کاهش می داریم:

$$\theta_j := \theta_j - \alpha \frac{\partial J(\theta_0, \theta_1)}{\partial \theta_j}$$

$$\theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^m (h_{\theta}(X^{(i)}) - y^{(i)}) X_j^{(i)}$$

که در آن h همان رابطه ای است که در بالا گفته شد. در واقع همان تابع پیش بینی برای این نوع مسایل است.

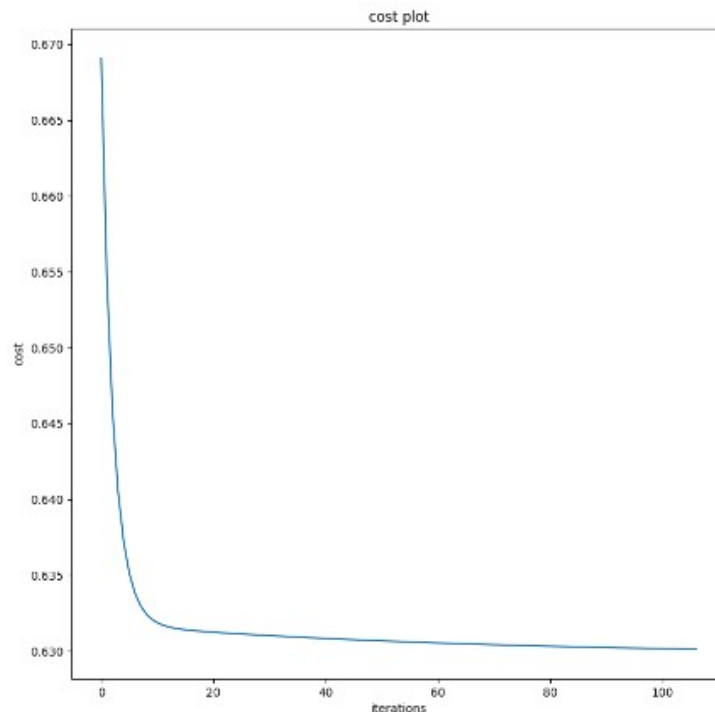
در تابع هزینه مقدار h را مشابه آنچه خواسته شده قرار می دهیم و کاست را محاسبه می کنیم.

```
##### cost
def cost_function(X, y, theta):
    m = len(y)
    h = 1 / (1 + np.exp(-1 * np.dot(X, theta)))
    cost = (-1/m) * np.sum((np.dot(y.T, np.log(h)) + np.dot((1-y).T, np.log(1-h))))
    return cost
```

با توجه به توضیحات گرادیان کاهش می کنیم:

```
### gradian descent
prev_cost= 0
tolerance= 100000
costs= []
while(abs(tolerance)>0.00001):
    h = 1 / (1 + np.exp(-1*np.dot(X, theta)))
    theta = theta - 0.0001/m * np.dot(X.T,h-y)
    cost= cost_function(X,y,theta)
    costs.append(cost)
    tolerance= abs(cost-prev_cost)/cost
    prev_cost= cost;
```

و مقدار هزینه را در هر مرحله رسم می کنیم:



همان طور که مشاهده می شود این مقدار کاهش میابد تا زمانی که شرط پایان حلقه رعایت شود.
میزان هزینه نهایی برابر است با:

0.6301253493641691

و مقادیر تتا برابر اند با:

```
array([[ -0.00067011],
       [ 0.00830081],
       [ 0.00254221]])
```

بخش ۳-۴

فایل q4_2:

با استفاده از L2norm تابع هزینه به صورت زیر در می آید:

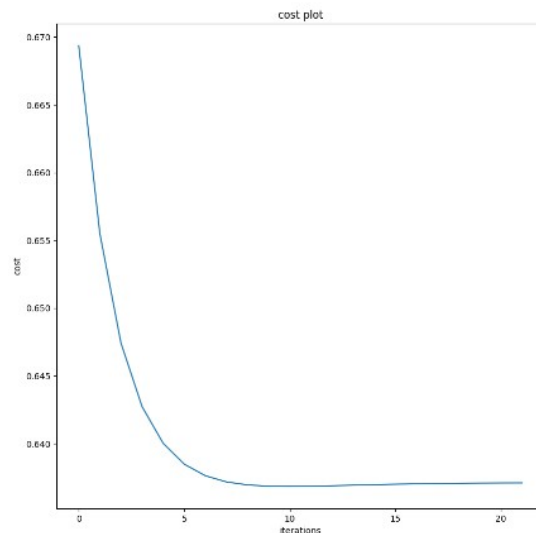
$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(X^{(i)}) - Y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2$$

```
##### cost
def cost_function(X, y, theta):
    m = len(y)
    ld= 100
    h = 1 / (1 + np.exp(-1 * np.dot(X, theta)))
    cost = (-1/m) * np.sum((np.dot(y.T, np.log(h)) + np.dot((1-y.T), np.log(1-h)))) + ld*np.sum(theta**2)
    return cost
```

با در نظر گرفتن تابع هزینه بالا، گرادیان کاهشی برای logistic regression به صورت زیر در می آید:

$$\theta_j := \theta_j \left(1 - \frac{\alpha \lambda}{m}\right) - \frac{\alpha}{m} \sum_{i=1}^m (h_{\theta}(X^{(i)}) - y^{(i)}) X_j^{(i)}$$

```
##### gradient descent
prev_cost= 0
tolerance= 100000
costs= []
ld=100
alpha= 0.0001
while(abs(tolerance)>0.00001):
    h = 1 / (1 + np.exp(-1*np.dot(X, theta)))
    theta = theta * (1-alpha*ld/m) - alpha/m * np.dot(X.T, h-y)
    cost= cost_function(X, y, theta)
    costs.append(cost)
    tolerance= abs(cost-prev_cost)/cost
    prev_cost= cost;
```



در این حالت تتا بهینه برابر است با:

```
array([[ -7.55361573e-05],  
       [ 6.16187421e-03],  
       [ 4.56324961e-03]])
```

و هزینه نهایی برابر است با:

0.6371038537892917

بخش ۴-۴
تحلیل:

با استفاده از regularization انتظار داریم که کاست بیشتر شود. به دلیل اینکه زمانی regularization انجام می‌دهیم که برای مدل ما over fitting رخ داده است بنابراین با این کار دقت را کاهش می‌دهیم که خطا را برای داده‌های تست جدید کمتر کنیم. و با توجه به نتایج بالا همین اتفاق هم افتاده است. تابع هزینه در حالت دوم بیشتر از حالت اول شده است و دقت برای این داده‌ها کاهش یافته است.

دقت را از فرمول زیر هم می‌توان بدست آورد:

```
##### accuracy calculation  
y_predicted = np.dot(X, theta)  
y_predicted = y_predicted > 0  
acc = np.sum(y_predicted == y) / len(y)
```

بخش ۵-۴

در مواردی که لاجستیک رگرشن با چند کلاسی (multi class) داریم از یکی از روش‌های زیر می‌توان استفاده کرد.

روش یک به چند (one vs all):

در هر مرحله به دنبال خطی هستیم که یک کلاس را از بقیه کلاس‌ها جدا کند.

$$y_{test} = \max_i h_i(\theta(x_{test})) \quad i=1 \dots k$$

$$h_i(\theta(x)) = P(y=i|x, \theta) \quad i=1 \dots k$$

روش یک به یک (one vs one):

در این روش به طور کلی $\binom{k}{2}$ تعداد مدل خواهیم داشت. که به ازای هر کلاس خطی داریم که دو به دو کلاس‌ها را از هم متمایز می‌کند.

سؤال ۵)

تعبیر احتمالاتی رگرسیون خطی:
اگر متغیرهای ما deterministic نباشند، یعنی دارای متغیر تصادفی یا نویز باشند، به این معنی که با ورودی یکسان خروجی یکسان نداشته باشیم، در این صورت داریم:

$$y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)}$$

با فرض کردن ϵ به عنوان یک ترم مستقل با توزیع گوسی، آنگاه داریم:

$$P(y^{(i)} | x^{(i)}, \theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)$$

هدف افزایش احتمال گفته شده و در نتیجه آن کاهش تابع هزینه رگرسیون خطی است.
تابع $l(\theta)$ را برابر با \log تابع احتمال گفته شده در نظر می‌گیریم.

$$P_{\theta}(y|x) = \prod P(y^{(i)} | x^{(i)}, \theta)$$

$$l(\theta) = \log(P(\theta))$$

حال با \max کردن این تابع مقدار بهینه θ را به دست می‌آوریم.

$$\max(l(\theta)) = \min \frac{1}{2} \sum (y^{(i)} - \theta^T x^{(i)})^2$$

تعبیر احتمالاتی رگرسیون لاجستیک:
در رگرسیون لاجستیک مجموع احتمالات برابر با یک می‌باشد.

$$h_{\theta} = P(y=1|x, \theta)$$
$$1-h_{\theta} = P(y=0|x, \theta)$$

تابع احتمالاتی لاجستیک را به صورت زیر هم می‌توان تعریف کرد:

$$P(y|x, \theta) = h_{\theta}^y (1-h_{\theta})^{(1-y)}$$

حال از این تابع \log می‌گیریم و سپس آن را \max می‌کنیم و از این طریق مقدار بهینه θ را به دست می‌آوریم.

$$L(\theta) = \log(P(\theta))$$

$$\max(l(\theta)) = \min \left(\sum y^{(i)} h_{\theta} + (1-y^{(i)})(1-h_{\theta}) \right)$$

مدل خطی تعمیم یافته رگرسیون پواسون:
در توزیع خانواده نمایی داریم:

$$P(y; \eta) = b(y) \exp(\eta^T T(y) - a(\eta))$$

و توزیع احتمالاتی پواسن برابر است با:

$$P(y; \eta) = \frac{\lambda^y e^{-\lambda}}{y!}$$

از تابع احتمالاتی گفته شده \log و \exp می گیریم:

$$\exp(\log(\frac{\lambda^y e^{-\lambda}}{y!})) = \exp(y \log(\lambda) - \lambda - \log(y!)) = \exp(y \log(\lambda)) * \exp(-\lambda) \frac{1}{y!}$$

$$\exp(y \log(\lambda) - \lambda) \frac{1}{y!}$$

$$\eta = \log(\lambda) \rightarrow \lambda = e^\eta$$

$$b(y) = \frac{1}{y!}$$

$$T(y) = y$$

$$a(\eta) = \lambda = e^\eta$$