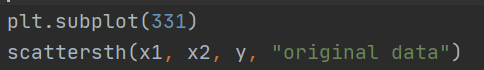
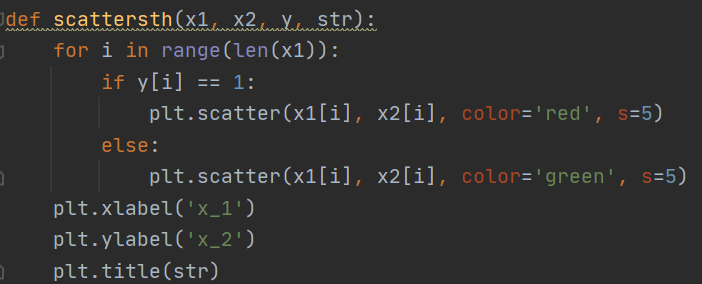
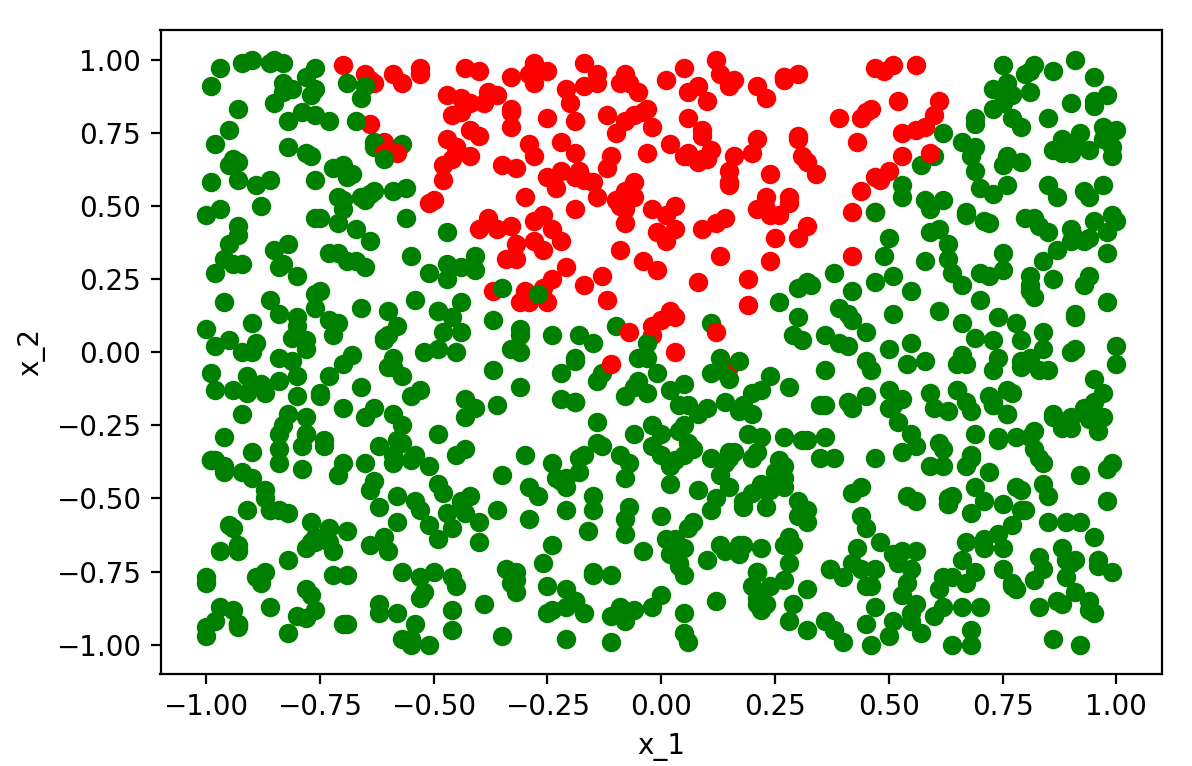
**(a）(i)I use a iteration when y’s value equals 1 I give the color “red” , else give the color “green”, and I set the x axis as ‘x\_1’ , set the y axis as ‘x\_2’ , because on the plot the x-axis is the value of parameter ‘x1 ’, the y-axis is the value of the second parameter ‘x1’ , and the parameter of ‘str’ is the title of the plot .Since I have to plot many times using the same pattern , so I define a plot function in the “funcfile.py” :** 

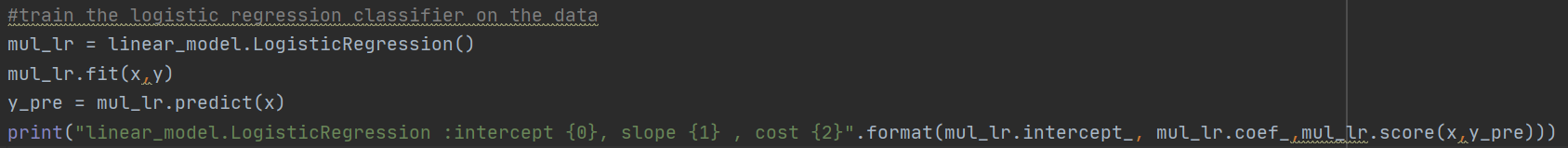
**And just call it to plot the original data , and the subplot means this is the first image of 3\*3=9 (all images), because I have many plots, this helps to see clearly:**

****

**Figure 1: the original data , as we can see from the plot**

**the red points mean the value of y is 1 , green ones mean -1 ,**

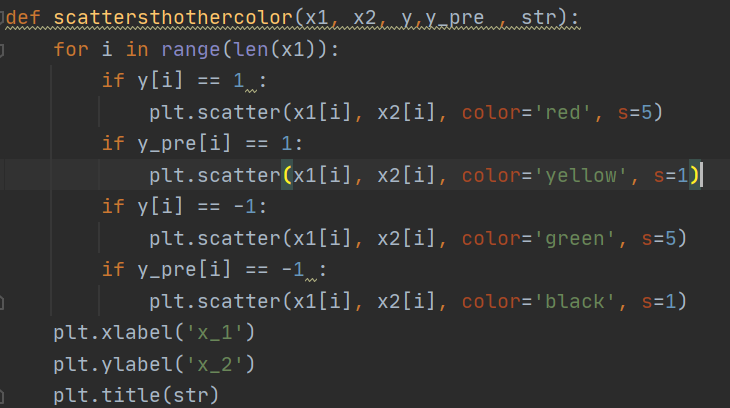
**we can see clearly there’s a boundary of the two colors’ points.**

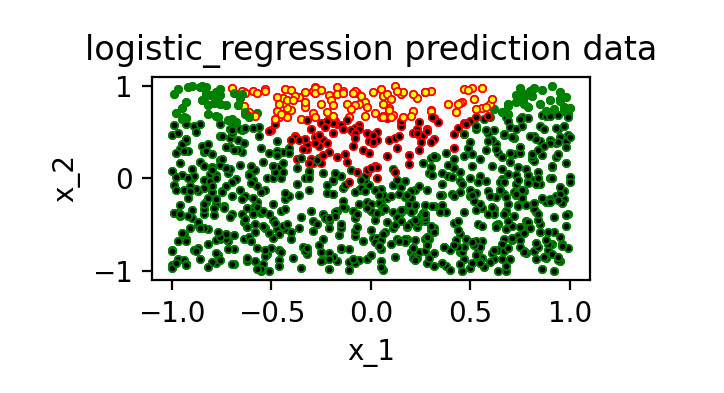
**(ii)** 

**I use sklearn to train a logistic regression classifier on the data and the result is intercept : [-2.29502037], slope : [[-0.22934877 3.51212233]]**

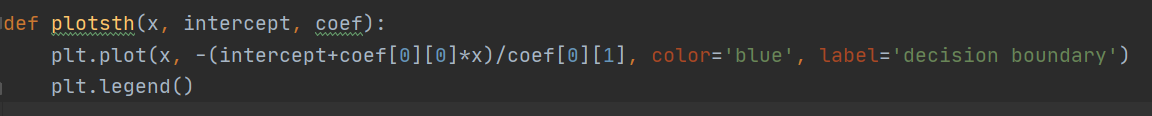
**And the score of this model is 0.8268268268268268, which means this model’s accuracy is 82.68% when predicts the data.**

**(iii)Because this requires different color with 2 datasets , so I define a function “scattersthothercolor” to plot the two datasets and could clearly see the different y value of the original ones and the prediction ones . When y=1 plot red point , y=-1 plot the green point , y\_pre=1 plot the yellow point , y\_pre=-1 plot the black point , in case the overlap so I set the size of point of the prediction data 1 while the original data is 5 , then we can see clearly the yellow points with red circle are the data when y’s value equals y\_pre , the whole green points are the data when y’s value is -1 while the y\_pre is 1 ; the black points with red circle is when y=1 and y\_pre=-1 , the black points with green circle is when y=-1 and y\_pre=-1.**





**Figure 2 the prediction data with the original data**

**We usually use to define decision boundary, when we classify it to y=1 , and the parameters b is the intercept and [w1,w2]=slop , so we can get the x2 equals -(w1x1+b)/w2 , so I define a function to draw the linear decision boundary in “funcfile.py” , the parameter “intercept” is the b , the parameter “coef” is the array of [w1 , w2] :** 

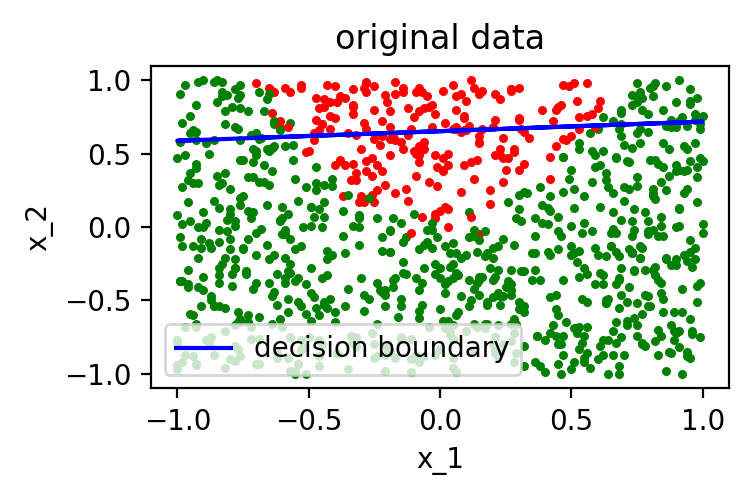
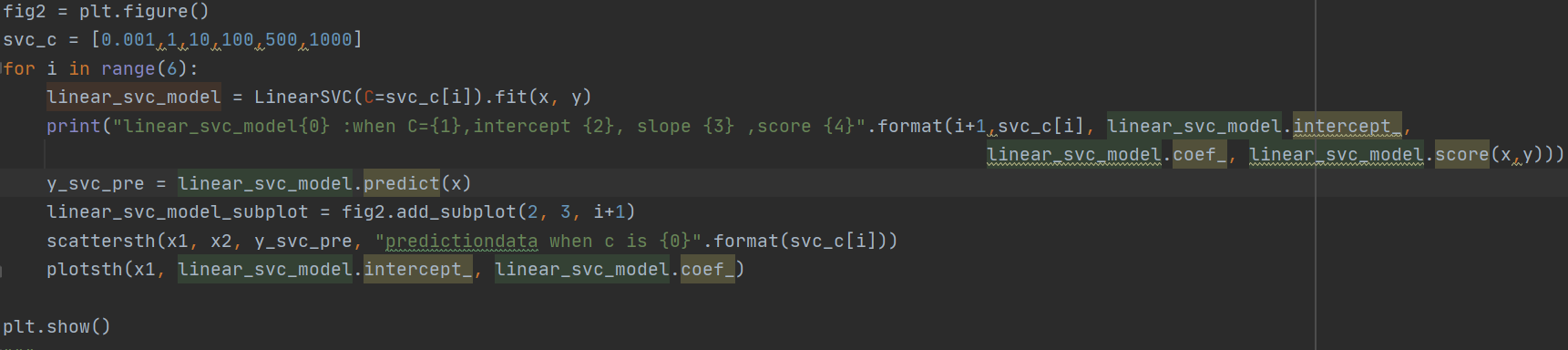
**So just recall it and the output image comes.** 

Figure 3: the decision boundary of the trained model in the original data

**(iv) This prediction can’t reflect the feature of all orignal data , I think it just return whichever class is the most likely, and the model can be improved to some extent , because the model is linear and through the plot of original data I think it’s more like a curve using quadratic function to plot the decision boundary.**

**(b）(i)I pick 6 numbers of C to train the model , and print the parameters, after that predict the data then plot it , using the “add\_subplot” function.**



**I found when C is too small or too large , the score of the model would decrese.**

**linear\_svc\_model1 :when C=0.001,intercept [-0.38304573], slope [[-0.02813036 0.31868212]] ,score 0.7877877877877878**

**linear\_svc\_model2 :when C=1,intercept [-0.78348024], slope [[-0.08562309 1.2261786 ]] ,score 0.8248248248248248**

**linear\_svc\_model3 :when C=10,intercept [-0.78858122], slope [[-0.08627484 1.23652484]] ,score 0.8268268268268268**

**linear\_svc\_model4 :when C=100,intercept [-0.80587576], slope [[-0.05154838 1.24861954]] ,score 0.8288288288288288**

**linear\_svc\_model5 :when C=500,intercept [-0.85731115], slope [[0.21140994 1.02179456]] ,score 0.7977977977977978**

**linear\_svc\_model6 :when C=1000,intercept [-1.17827911], slope [[0.03957859 1.56852242]] ,score 0.8118118118118118**

**(ii)**

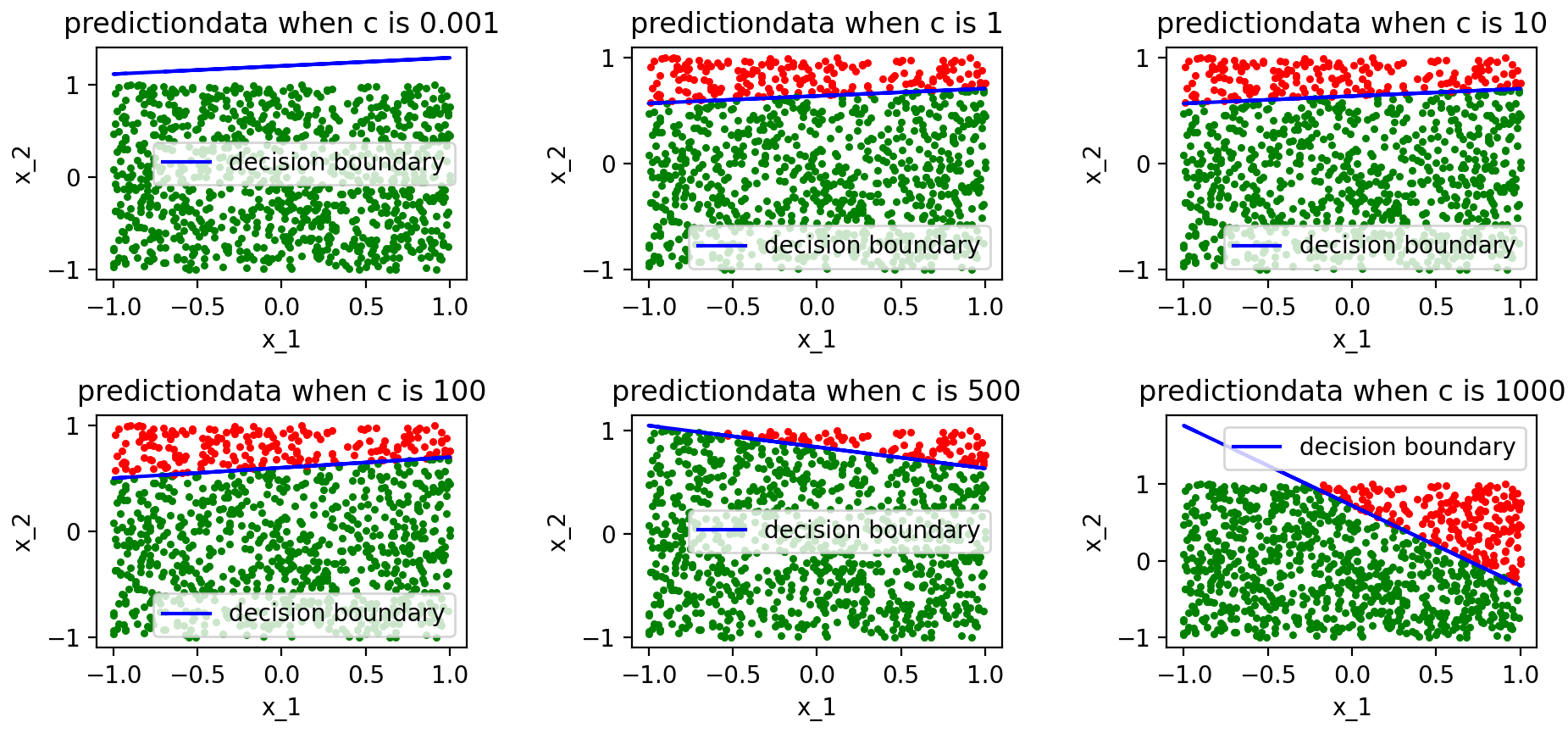
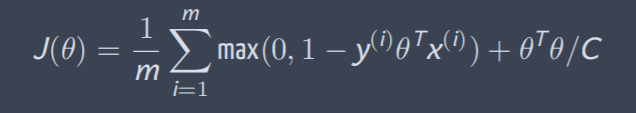


Figure 4 the prediction\_data with different value of C

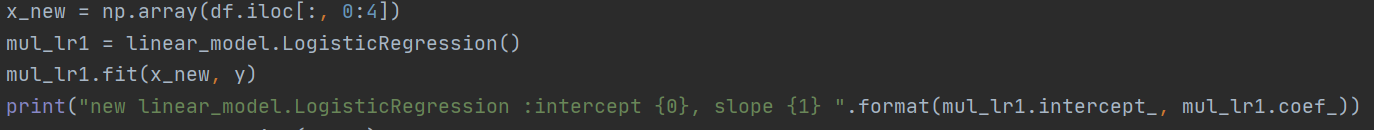
**I find the if C is too small it’s underfitting while too large is overfitting , and they are all linear prediction models. When the value of c limits to a range the prediction data doesn’t vary so much , once out of that bound it varies much.**

**(iii) As we can see from the SVM cost function :**

**The less value of C , the model would be underfitting, the higher the value of C , the less the model is regularized , and the less important penalty is ,that means you can tolerate higher cost function values. If C is so big that the SVM predictions would be changing at each run time , because it is over-fitting which means it doesn’t generalize well and doesn’t predict well for data outside the training set.**

**(c）**

**(i) To get the parameter:**

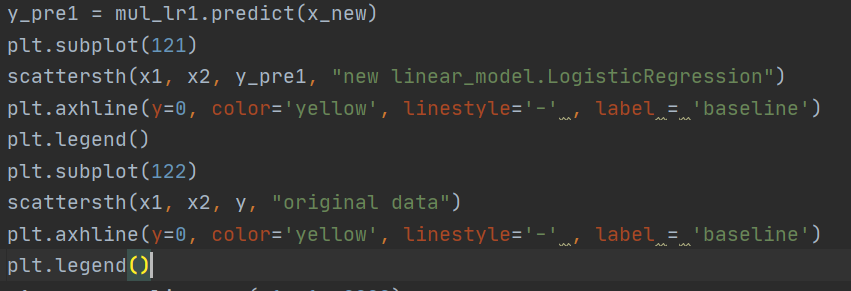


**(new linear\_model.LogisticRegression)**

**intercept [-0.80227213]**

**slope [[-0.22141638 5.35919863 -8.05566943 -0.26219426]]**

**(ii) I use subplot function and recall the “scattersth” function to plot the two sets of data.**



**When add two features the graph , as we can see from the baseline , the new model’s prediction doesn’t change much compared with the original data.**

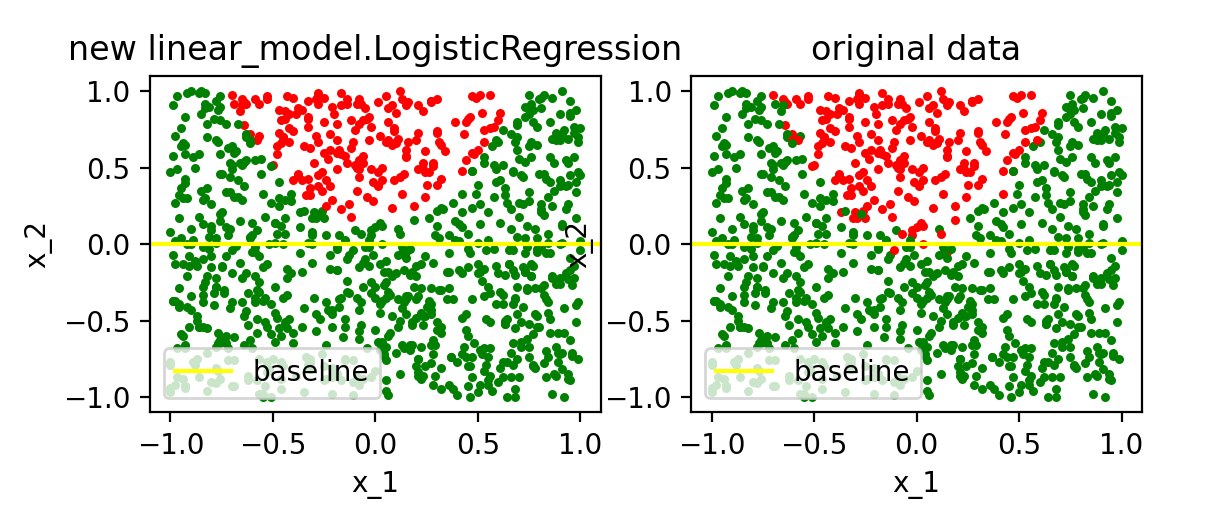


Figure 5 the new model and the original data with the y=0 baseline

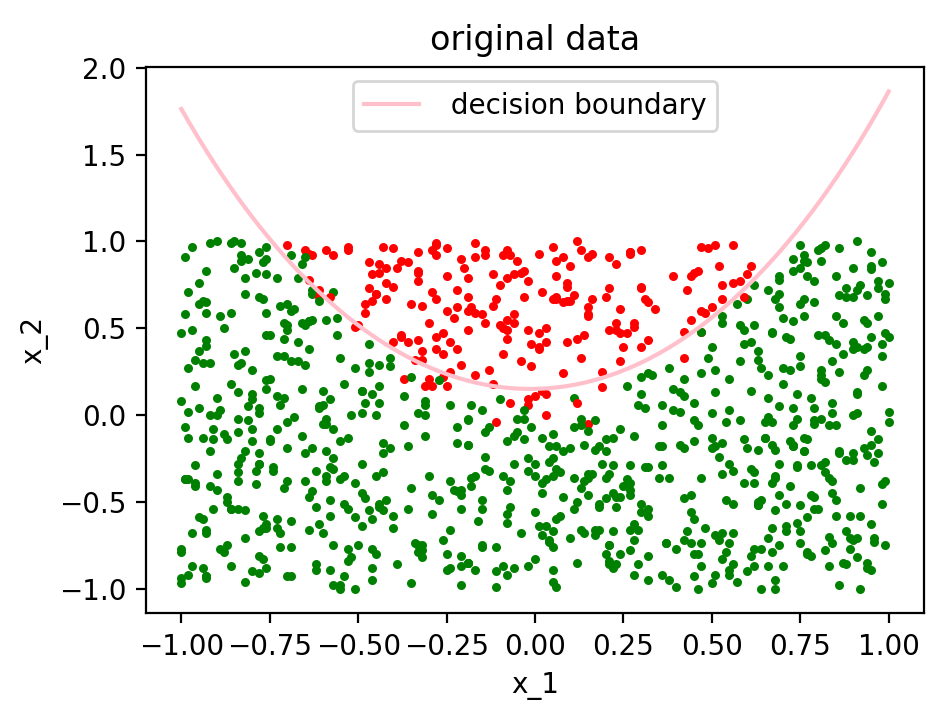
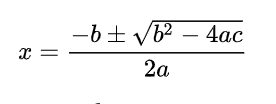
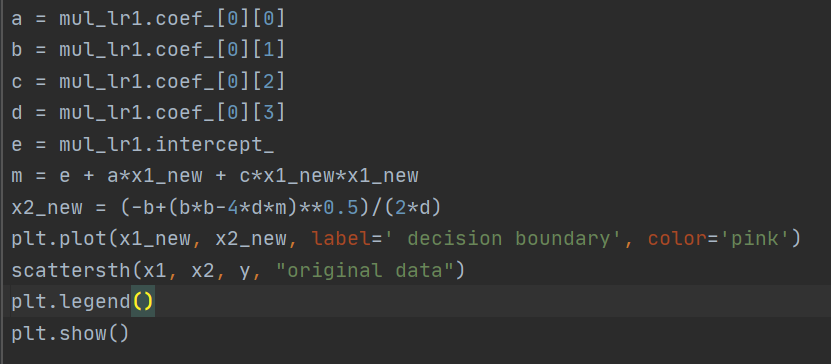
**(iii) **

Figure 6 the decision boundary of the original data

**I define each of the coefficient as a,b,c,d , and e is the intercept , then I apply the mathematic quadratic formula**  **,and in this case m is the c in the formula because we can regard the x1 as constant to get the x2\_new , and then call the plot function.**



**Appendix**

**week2.py**

import numpy as np  
import pandas as pd  
from funcfile import \*  
from sklearn import linear\_model  
from sklearn.svm import LinearSVC  
from sklearn.pipeline import Pipeline  
from sklearn.preprocessing import PolynomialFeatures  
  
#read data  
df = pd.read\_csv("week2.csv",comment="#")  
print(df.head())  
x = np.array(df.iloc[:, 0:2])#read the first two colunms as array  
y = np.array(df.iloc[:, 4])#read the y value  
  
#train the logistic regression classifier on the data  
mul\_lr = linear\_model.LogisticRegression()  
mul\_lr.fit(x,y)  
y\_pre = mul\_lr.predict(x)  
print("linear\_model.LogisticRegression :intercept {0}, slope {1} , score {2}".format(mul\_lr.intercept\_, mul\_lr.coef\_,mul\_lr.score(x,y)))  
  
x1 = x[:, 0]  
x2 = x[:, 1]  
  
  
plt.subplot(331)  
scattersth(x1, x2, y, "original data")  
plotsth(x1, mul\_lr.intercept\_, mul\_lr.coef\_)  
  
plt.subplot(332)  
plt.plot()  
scattersthothercolor(x1, x2, y,y\_pre, "logistic\_regression prediction data")  
plt.show()  
#pick 6 numbers of C to train the model , and print the parameters,   
#after that predict the data then plot it , using the “add\_subplot” function  
fig2 = plt.figure()  
svc\_c = [0.001,1,10,100,500,1000]  
for i in range(6):  
 linear\_svc\_model = LinearSVC(C=svc\_c[i]).fit(x, y)  
 print("linear\_svc\_model{0} :when C={1},intercept {2}, slope {3} ,score {4}".format(i+1,svc\_c[i], linear\_svc\_model.intercept\_,  
 linear\_svc\_model.coef\_, linear\_svc\_model.score(x,y)))  
 y\_svc\_pre = linear\_svc\_model.predict(x)  
 linear\_svc\_model\_subplot = fig2.add\_subplot(2, 3, i+1)  
 scattersth(x1, x2, y\_svc\_pre, "predictiondata when c is {0}".format(svc\_c[i]))  
 plotsth(x1, linear\_svc\_model.intercept\_, linear\_svc\_model.coef\_)  
  
plt.show()  
  
  
# (c)  
x\_new = np.array(df.iloc[:, 0:4])  
mul\_lr1 = linear\_model.LogisticRegression()  
mul\_lr1.fit(x\_new, y)  
print("new linear\_model.LogisticRegression :intercept {0}, slope {1} ".format(mul\_lr1.intercept\_, mul\_lr1.coef\_))  
y\_pre1 = mul\_lr1.predict(x\_new)  
plt.subplot(121)  
scattersth(x1, x2, y\_pre1, "new linear\_model.LogisticRegression")  
plt.axhline(y=0, color='yellow', linestyle='-' , label = 'baseline')  
plt.legend()  
plt.subplot(122)  
scattersth(x1, x2, y, "original data")  
plt.axhline(y=0, color='yellow', linestyle='-' , label = 'baseline')  
plt.legend()  
"""define each of the coefficient as a,b,c,d , and e is the intercept , then I apply the mathematic quadratic formula """  
x1\_new = np.linspace(-1, 1, 9999)#to get the axis of x  
a = mul\_lr1.coef\_[0][0]  
b = mul\_lr1.coef\_[0][1]  
c = mul\_lr1.coef\_[0][2]  
d = mul\_lr1.coef\_[0][3]  
e = mul\_lr1.intercept\_  
m = e + a\*x1\_new + c\*x1\_new\*x1\_new  
x2\_new = (-b+(b\*b-4\*d\*m)\*\*0.5)/(2\*d)  
plt.plot(x1\_new, x2\_new, label=' decision boundary', color='pink')  
scattersth(x1, x2, y, "original data")  
plt.legend()  
plt.show()

**“funcfile.py”**

import matplotlib.pyplot as plt  
import sympy as sp  
  
#plot the data  
"""when y’s value equals 1 color is “red” ,else “green”, and set the x axis as ‘x\_1’ ,  
set the y axis as ‘x\_2’ , because on the plot the x-axis is the value of parameter ‘x1 ’,  
the y-axis is the value of the second parameter ‘x1’ , and the parameter of ‘str’ is   
the title of the plot"""  
def scattersth(x1, x2, y, str):  
 for i in range(len(x1)):  
 if y[i] == 1:  
 plt.scatter(x1[i], x2[i], color='red', s=5)  
 else:  
 plt.scatter(x1[i], x2[i], color='green', s=5)  
 plt.xlabel('x\_1')  
 plt.ylabel('x\_2')  
 plt.title(str)  
  
""" plot the two datasets and could clearly see the different y value of the original ones and the prediction ones"""  
def scattersthothercolor(x1, x2, y,y\_pre , str):  
 for i in range(len(x1)):  
 if y[i] == 1 :  
 plt.scatter(x1[i], x2[i], color='red', s=5)  
 if y\_pre[i] == 1:  
 plt.scatter(x1[i], x2[i], color='yellow', s=1)  
 if y[i] == -1:  
 plt.scatter(x1[i], x2[i], color='green', s=5)  
 if y\_pre[i] == -1 :  
 plt.scatter(x1[i], x2[i], color='black', s=1)  
 plt.xlabel('x\_1')  
 plt.ylabel('x\_2')  
 plt.title(str)  
  
"""plot the line with intercept and coef"""  
def plotsth(x, intercept, coef):  
 plt.plot(x, -(intercept+coef[0][0]\*x)/coef[0][1], color='blue', label='decision boundary')  
 plt.legend()