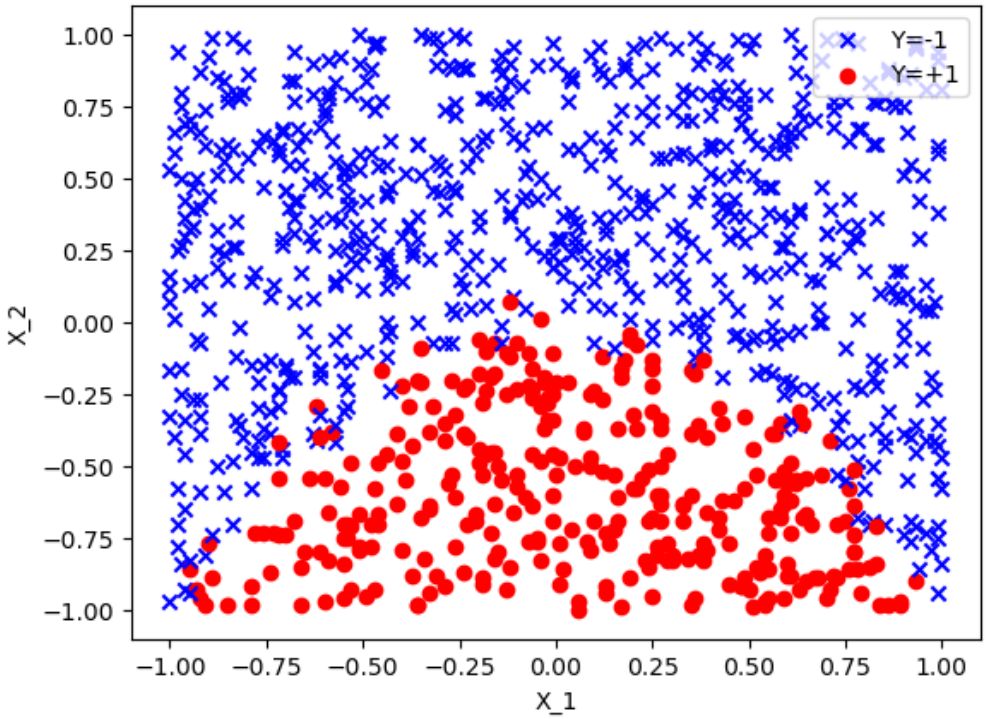
A(i):

The following plot is created using Matplotlib library. The data on the X axis which is labelled as X_1 represents the first feature(X1) in the csv file and the data on the Y axis which is labelled as X_2 represents the second feature(X2) in the csv file. From the figure, we can see that all the points are in the range between -1 and 1. The blue x (x) represents the y value to be 1(Y=1) and the red circle(o) represents the y value to be -1 (Y=-1) and we can see the same on the legends on the top right corner of the plot.

Output:



A(ii):

The logistic regression model we obtained after fitting it with the training data X, gave the Y intercept value as -1.75476793. Also, the coefficients, -0.02844251 and -4.7886813 which are weights of X1 and X2 respectively. As the weight of X2 is negative and large will cause the feature Y to decrease rapidly for an increase in X2. The accuracy we obtained which is 86.08% shows the percentage of accurate results when tested with training data. Accuracy is obtained by calculating the percentage of correct predictions out of all the predictions.

Output:

Intercept= [-1.75476793] Coefficients(X1,X2) [[-0.02844251 -4.7886813]]
Accuracy= 0.8608608608608609

A(iii):

The following plot shows the results we got from the predicting the training data. The equation for the logistic regression is z=w1x1+w2x2+b for a two-dimensional problem. The decision boundary can be x2=mx1+c. For the x2 intercept, x1=0 and for the points on the decision boundary, z=0.

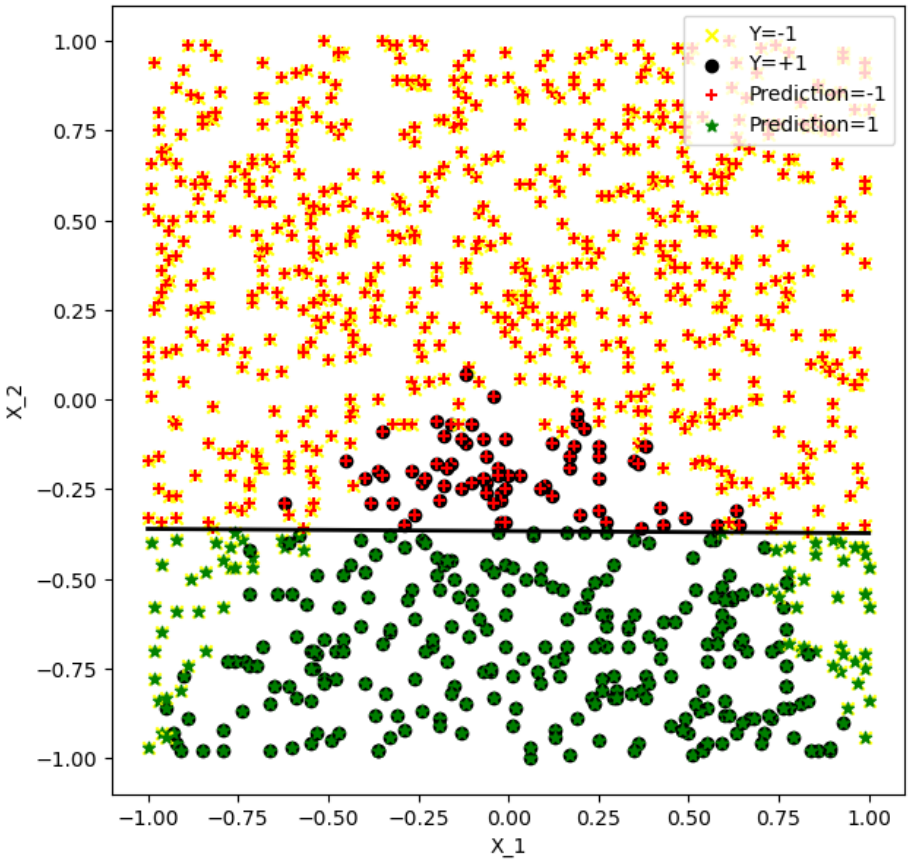
So, the equation becomes: 0=0+w2c+b ⇒ c=-b/w2

For the slope, m, consider having two points on the decision boundary (a1,a2) and (a3,a4). Then the slope of it becomes m=(a4-a2)/(a3-a1).

Since both of the points are on the same line,

w1a1+w2a2+b = w1a3+w2a4+b
⇒ -w2(a4-a2)=w1(a3-a1)
⇒ m= -w1/w2 .
Then we plot the decision boundary by using values of m, c in the line x2=mx1+c.

Output:



A(iv):

From the plot above, we can see that there are two classes in the dataset and they are not linearly separable (cannot be separated by a straight line) but the logistic regression is separating them linearly. We obtained an accuracy of 86.08% and so the error% will be 100%-86.08%=13.92%.

B(i):

Below is the output of parameters for the Linear SVC models for 5 values of C(0.0001,0.001.0.1,1,100).

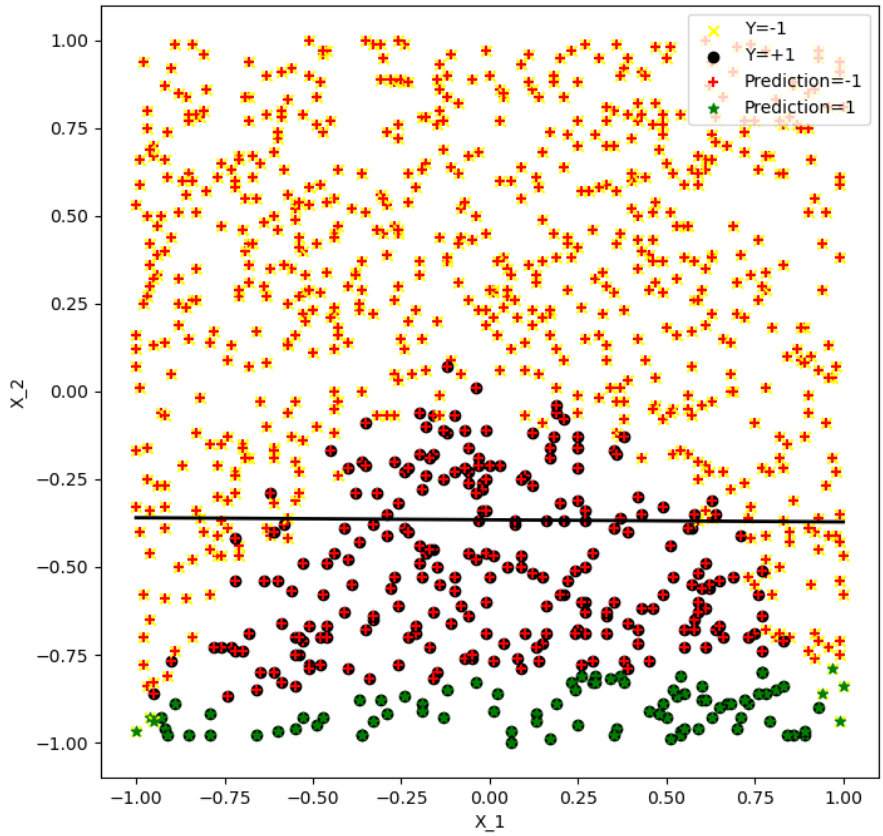
Output:

C=0.0001 --- Intercept= [-0.06008564] Coefficeints(X,Y)= [[0.00481313 -0.07307907]]
C=0.001 --- Intercept= [-0.23847461] Coefficeints(X,Y)= [[0.02226261 -0.46358698]]
C=0.1 --- Intercept= [-0.56485568] Coefficeints(X,Y)= [[-0.00839884 -1.6116824]]
C=1 --- Intercept= [-0.61158311] Coefficeints(X,Y)= [[-0.01287362 -1.72834663]]
C=100 --- Intercept= [-0.61809681] Coefficeints(X,Y)= [[-0.01343021 -1.74436941]]

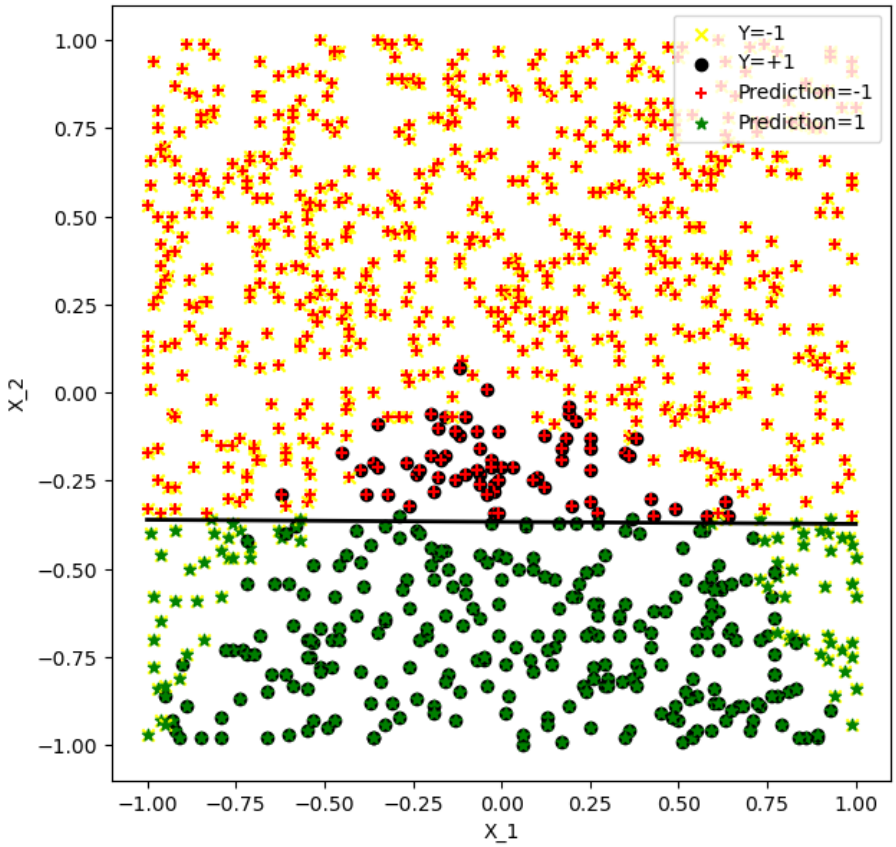
Accuracy when C=0.0001 = 0.7557557557557557
Accuracy when C=0.001 = 0.8458458458458459
Accuracy when C=0.1 = 0.8568568568568569
Accuracy when C=1 = 0.8568568568568569
Accuracy when C=100 = 0.8568568568568569

B(ii):

The below plot shows the prediction of the training data when C=0.0001.



The below plot shows the prediction of the training data when C=0.1.



The graph is plotted by using the steps used before in a(iii).

B(iii):

C is a regularization parameter for the hyperplane(decision boundary's width in simple terms). It is inversely proportional to the width of the hyperplane. So, as the C increases, the data misclassificatio n decreases, as there will be a compact hyperplane. For a lower value of C, the model tend to generalize the prediction but it is not the case in our model, since we imported svc from linear model in sklearn and the data is not linearly separable.

B(iv):

From B(i) we can see that, the accuracy of the model increases as the value of the C increases until 0.1 and then it is constant which is because the size of the dataset is small. As we increase the value of C, since the hyperplane is compact and the data misclassification in that plane is reduced, the accuracy will be close to that of Logistic Regression.

Logistic Regression Accuracy: 86.08%
Linear SVC accuracy when C value is high : 85.68%

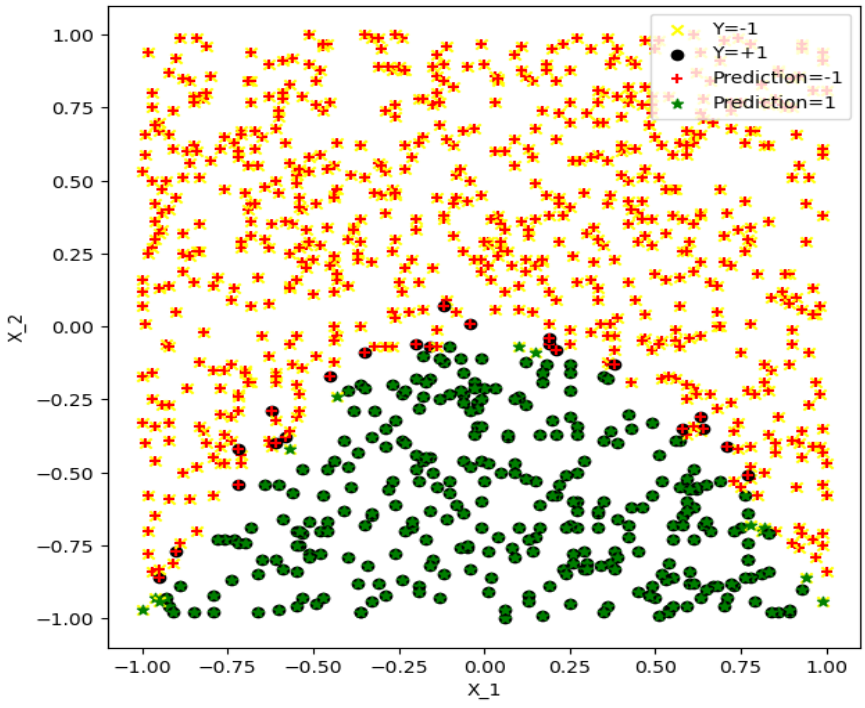
C(i):

Two features are created by squaring the existing features and they are passed to the Logistic Regres sion Model. Weight of the feature X2 is negative and large. So the output Y depends on X2 more than any other feature but they are inversely proportional. The Intercept Values and Coefficients are show n below.

Output:

Intercept= [-0.38367671] Coefficients(X,Y)= [[0.09756772 -6.80240725 -6.74528179 0.78087507]]

C(ii):
The plot below shows the prediction of the training data using just the two original features(X1 and X 2). From the plot we understand that the decision boundary is non-linear. We notice that the predict ions are better than that of our previous SVM and Logistic Regression Models.
Output:



C(iii):

In our data, the majority class is -1(Y=-1). So we compare our model accuracy against the accuracy of a baseline predictor which always predicts -1.

Output:

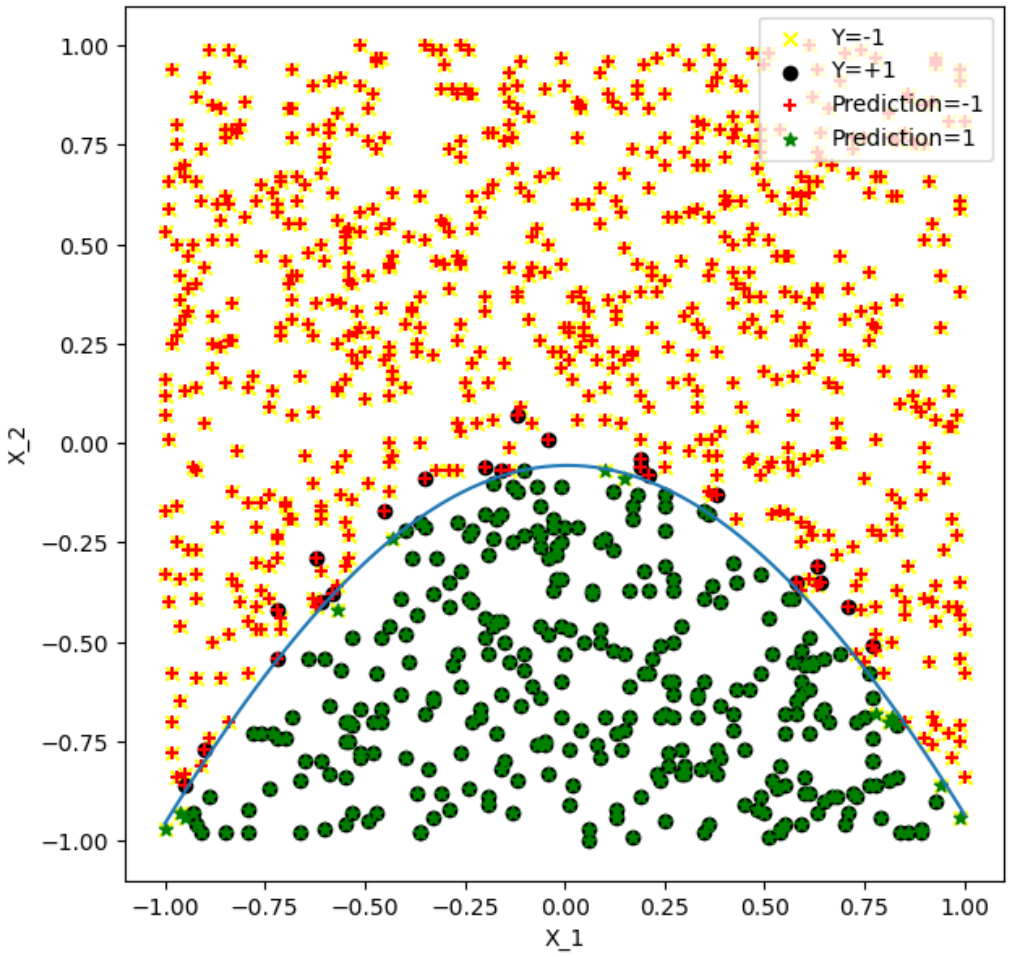
Baseline Score 0.6806806806806807
New Score_LG 0.965965965965966

C(iv):

The equation for the logistic regression is z=w1x1+w2x2+w3x1^2+w4x2^2+w5(intercept) for regression with 4 features. So, the decision boundary will be w1x1+w2x2+w3x1^2+w4x2^2+intercept=0.

We can convert this equation to ⇒ w4x2^2+w2x2+(w3x1^2+w1x1+intercept)
Consider w3x1^2+w1x1+intercept=c
⇒ w4x2^2+w2x2+c=0
This is a quadratic equation. So, we find out the roots of the equation to find the values of x2 by using the formula -b+/-sqrt(b^2-4*a*c)/2*a (Considering the values of X1 ranging from -1 to 1). Where the determinant value is not equal to zero, we find the values of x2 where it lies between (-1,1).

Plot the values of X1, X2 on the scatter plot obtained above.



Appendix:

Given Code:

```
## id:14--14--14
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
df = pd.read_csv("week2.csv")
print(df.head())
X1=df.iloc[:,0]
X2=df.iloc[:,1]
X=np.column_stack((X1,X2))
Y=df.iloc[:,2]
```

A(i):

```
#A1
points_1=plt.scatter(X1[Y<0],X2[Y<0],marker="x",color="blue",label="Y=-1")
points_2=plt.scatter(X1[Y>0],X2[Y>0],marker="o",color="red",label="Y=+1")
plt.xlabel("X_1")
plt.ylabel("X_2")
plt.legend(handles=[points_1,points_2],loc='upper right')
plt.rcParams["figure.figsize"] = (10,10)
plt.show()
```

A(ii):

```
#A2
from sklearn.linear_model import LogisticRegression as LG
Logistic_Model=LG().fit(X,Y)
print("Intercept=",Logistic_Model.intercept_,"Coefficients(X1,X2)",Logistic
_Model.coef_) #Intercept value and Weights value
y_prediction=Logistic_Model.predict(X) #Predicting the Model on the
training data
accuracy = np.sum(np.equal(Y, y_prediction)) / len(Y) #Calculating the
accuracy
print("Accuracy=",accuracy)
```

A(iii):

```
#A3
y_prediction=Logistic_Model.predict(X)    #Predicting the Model on the
training data
points_1=plt.scatter(X1[Y<0],X2[Y<0],marker="x",color="yellow",label="Y=-
1")
points_2=plt.scatter(X1[Y>0],X2[Y>0],marker="o",color="black",label="Y=+1")
points_3=plt.scatter(X1[y_prediction<0],X2[y_prediction<0],
marker="+",color="red",label="Prediction=-1")
points_4=plt.scatter(X1[y_prediction>0],X2[y_prediction>0],marker="*",color
="green",label="Prediction=1")
plt.xlabel("X_1")
plt.ylabel("X_2")
plt.legend(handles=[points_1,points_2,points_3,points_4],loc='upper right')
```

```
#z=w1x1+w2x2+b
b = Logistic_Model.intercept_[0]
w1, w2 = Logistic_Model.coef_.T
#x2=mx1+c
c = -b/w2
m = -w1/w2
x1d = np.array([min(X1[:,]),max(X1[:,])]) #Minimum and Maximum value on X
Axis
x2d = m*x1d + c
plt.plot(x1d, x2d, c='black', lw=2)
plt.rcParams["figure.figsize"] = (7,7)
plt.show()
```

A(iv):

```
#A4
accuracy = np.sum(np.equal(Y, y_prediction)) / len(Y) #Calculating the
Accuracy
print("Accuracy=",accuracy)
```

B(i):

```
#B1
from sklearn.svm import LinearSVC as lsvc
lsvc_1 = lsvc(C=0.0001,max_iter=100000).fit(X, Y)  #Fitting the Linear SVC
Model with C values[0.0001,0.001,0.1,1,100]
lsvc_2 = lsvc(C=0.001,max_iter=100000).fit(X, Y)
lsvc_3 = lsvc(C=0.1,max_iter=100000).fit(X, Y)
lsvc_4 = lsvc(C=1,max_iter=100000).fit(X, Y)
lsvc_5 = lsvc(C=100,max_iter=100000).fit(X, Y)
print("C=0.0001 --- Intercept=",lsvc_1.intercept_,
"Coefficeints(X,Y)=",lsvc_1.coef_) #Intercept value and Weights value
print("C=0.001 --- Intercept=",lsvc_2.intercept_,
"Coefficeints(X,Y)=",lsvc_2.coef_)
print("C=0.1 --- Intercept=",lsvc_3.intercept_,
"Coefficeints(X,Y)=",lsvc_3.coef_)
print("C=1 --- Intercept=",lsvc_4.intercept_,
"Coefficeints(X,Y)=",lsvc_4.coef_)
print("C=100 --- Intercept=",lsvc_5.intercept_,
"Coefficeints(X,Y)=",lsvc_5.coef_)

y_prediction=lsvc_1.predict(X) #Calculating the accuracy of model when C
values[0.0001,0.001,0.1,1,100]
print("Accuracy when C=0.0001 =",np.sum(np.equal(Y, y_prediction)) / len(Y))
y_prediction=lsvc_2.predict(X)
print("Accuracy when C=0.001 =",np.sum(np.equal(Y, y_prediction)) / len(Y))
y_prediction=lsvc_3.predict(X)
print("Accuracy when C=0.1 =",np.sum(np.equal(Y, y_prediction)) / len(Y))
y_prediction=lsvc_4.predict(X)
print("Accuracy when C=1 =",np.sum(np.equal(Y, y_prediction)) / len(Y))
y_prediction=lsvc_5.predict(X)
print("Accuracy when C=100 =",np.sum(np.equal(Y, y_prediction)) / len(Y))
```

B(ii):

```
#B2
y_prediction=lsvc_3.predict(X) #Predicting value of Y with Training data
when C=0.1
points_1=plt.scatter(X1[Y<0],X2[Y<0],marker="x",color="yellow",label="Y=-
1")
points_2=plt.scatter(X1[Y>0],X2[Y>0],marker="o",color="black",label="Y=+1")
points_3=plt.scatter(X1[y_prediction<0],X2[y_prediction<0],
marker="+",color="red",label="Prediction=-1")
points_4=plt.scatter(X1[y_prediction>0],X2[y_prediction>0],marker="*",color
="green",label="Prediction=1")
plt.xlabel("X_1")
plt.ylabel("X_2")
plt.legend(handles=[points_1,points_2,points_3,points_4],loc='upper right')
#z=w1x1+w2x2+b
b = Logistic_Model.intercept_[0]
w1, w2 = Logistic_Model.coef_.T
#x2=mx1+c
c = -b/w2
m = -w1/w2
x1d = np.array([min(X1[:,]),max(X1[:,])]) #Minimum and Maximum value on X
Axis
x2d = m*x1d + c
plt.plot(x1d, x2d, c='black', lw=2)
plt.rcParams["figure.figsize"] = (7,7)
plt.show()
```

B(iv):

```
#B4
y_prediction=lsvc_3.predict(X)
print("Accuracy when C=0.1 =",np.sum(np.equal(Y, y_prediction)) / len(Y))
```

C(i):

```
#C1
X3=X1*X1.T
X4=X2*X2.T
X_Final=np.column_stack((X, X3, X4))
model_new = LG().fit(X_Final, Y)
print('Intercept=',model_new.intercept_,'Coefficients(X,Y)=',model_new.coef
_) #Intercept value and Weights value
y_prediction=model_new.predict(X_Final) #Predicting value of Y with
Training data
print("Accuracy=",np.sum(np.equal(Y, y_prediction)) / len(Y)) #Calculating
the accuracy
```

C(ii)::

```
#C2
y_prediction=model_new.predict(X_Final) #Predicting value of Y with
Training data
points_1=plt.scatter(X1[Y<0],X2[Y<0],marker="x",color="yellow",label="Y=-
1")
points_2=plt.scatter(X1[Y>0],X2[Y>0],marker="o",color="black",label="Y=+1")
```

```
points_3=plt.scatter(X1[y_prediction<0],X2[y_prediction<0],
marker="+",color="red",label="Prediction=-1")
points_4=plt.scatter(X1[y_prediction>0],X2[y_prediction>0],marker="*",color
="green",label="Prediction=1")
plt.xlabel("X_1")
plt.ylabel("X_2")
plt.legend(handles=[points_1,points_2,points_3,points_4],loc='upper right')
plt.rcParams["figure.figsize"] = (7,7)
plt.show()
```

C(iii):

```
#C3
print("Baseline_Score=",((Y==-1).sum()/len(Y)))
y_prediction=model_new.predict(X_Final) #Predicting value of Y with
Training data
print("Accuracy=",np.sum(np.equal(Y, y_prediction)) / len(Y))
```

C(iv):

```
import math
w1,w2,w3,w4 = model_new.coef_.T  #Getting the weights of the model
intrcpt_val=model_new.intercept_ #Getting the intercept value of the model
def Val_Find(x1): #Function for calculating the values of x2 with respect
to values of x1 based on the weights and intercept of the model
    c=(w3*x1*x1)+(w1*x1)+intrcpt_val
    b=w2
    a=w4
    Determinant=b*b-4*a*c
    if Determinant < 0: #If Determinant is less than zero roots are
imaginary
        return None
    else:
        root_1=(-b+math.sqrt(Determinant))/(2*a)
        root_2=(-b-math.sqrt(Determinant))/(2*a)
        if root_1>=-1 and root_1 <= 1:
            return root_1
        else:
            return root_2
X_1 = np.linspace(min(X1[:,]),max(X1[:,]), 100) #Creating an array with 100
values ranging from min and max of training data
X_2 =[]
for x1 in X_1:
    X_2.append(Val_Find(x1))
points_1=plt.scatter(X1[Y<0],X2[Y<0],marker="x",color="yellow",label="Y=-
1")
points_2=plt.scatter(X1[Y>0],X2[Y>0],marker="o",color="black",label="Y=+1")
points_3=plt.scatter(X1[y_prediction<0],X2[y_prediction<0],
marker="+",color="red",label="Prediction=-1")
points_4=plt.scatter(X1[y_prediction>0],X2[y_prediction>0],marker="*",color
="green",label="Prediction=1")
plt.xlabel('X_1')
plt.ylabel('X_2')
plt.legend(handles=[points_1,points_2,points_3,points_4],loc='upper right')
plt.rcParams["figure.figsize"] = (7,7)
plt.plot(X_1,X_2)
```