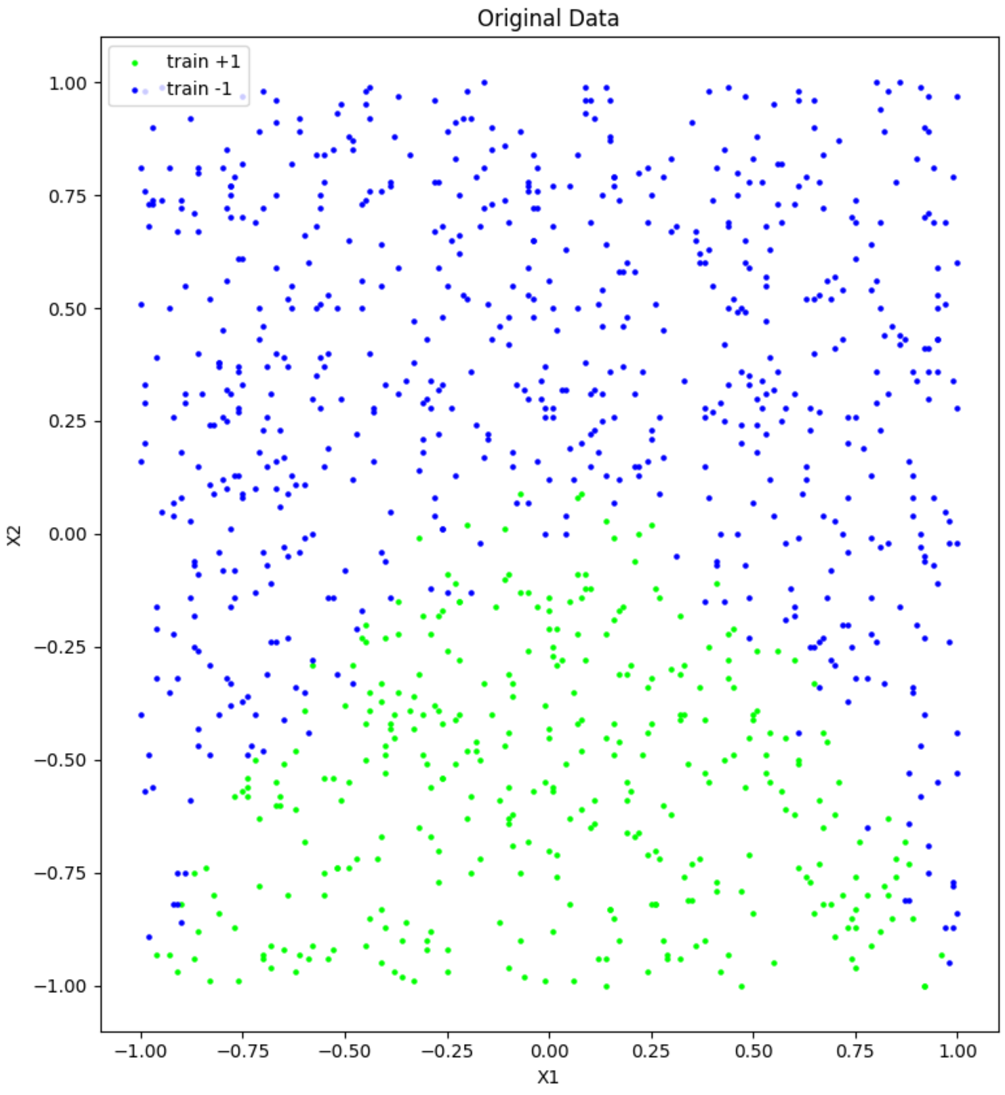
First line of dataset: `# id:2--2--2,0,1`

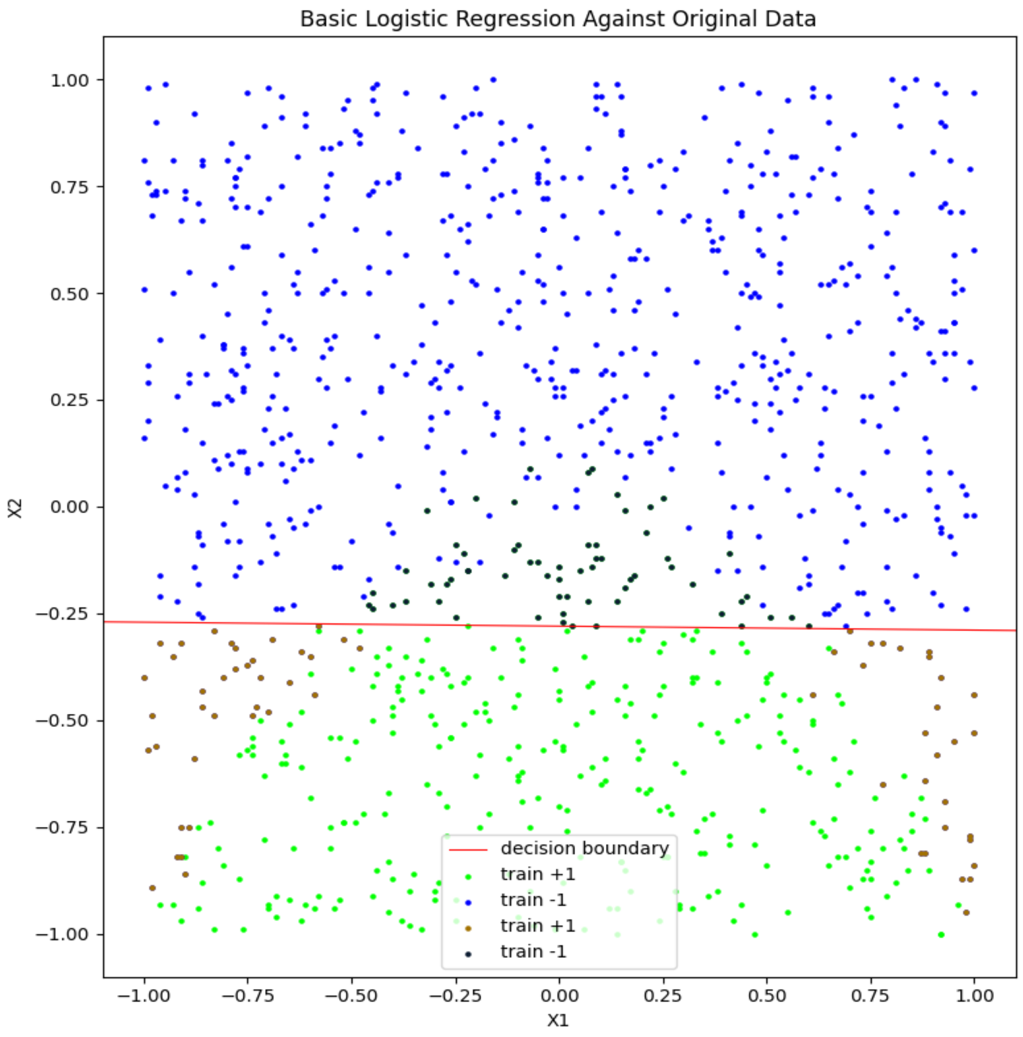
a. (i)



This plot shows the variable x_1 and x_2 from the original dataset plotted against each other, with y determining the colour of the points in the plot, by doing this we see a few key things, one, the data is not linearly separable, two, some of the points are blurring together, so a model that perfectly fits our data while not over-fitting is impossible.

(ii) Using k-fold cross-validation to find the model with the smallest mean square error on our test data, the parameters I got were an intercept of -1.506949, a slope for x_1 of -0.049770 and a slope for x_2 of -5.386699, giving a logistic regression model of y=1 when $-1.506949 -0.049770x_1 -5.386699x_2 > 0$ and $y = -1$ when $-1.506949 -0.049770x_1 -5.386699x_2 < 0$. This tells us that x_2 has a much larger effect on the outcome of y in comparison to x_1.

(iii)



Obtaining the decision boundary is relatively easy, we get it by simply first, making our logistic regression formula from earlier equal to 0, giving us $\Theta_0 + \Theta_1x_1 + \Theta_2x_2 = 0$, then we rearrange this to make x_2 the result of the equation, giving us $(\Theta_0 + \Theta_1x_1) / (-\Theta_2) = x_2$.

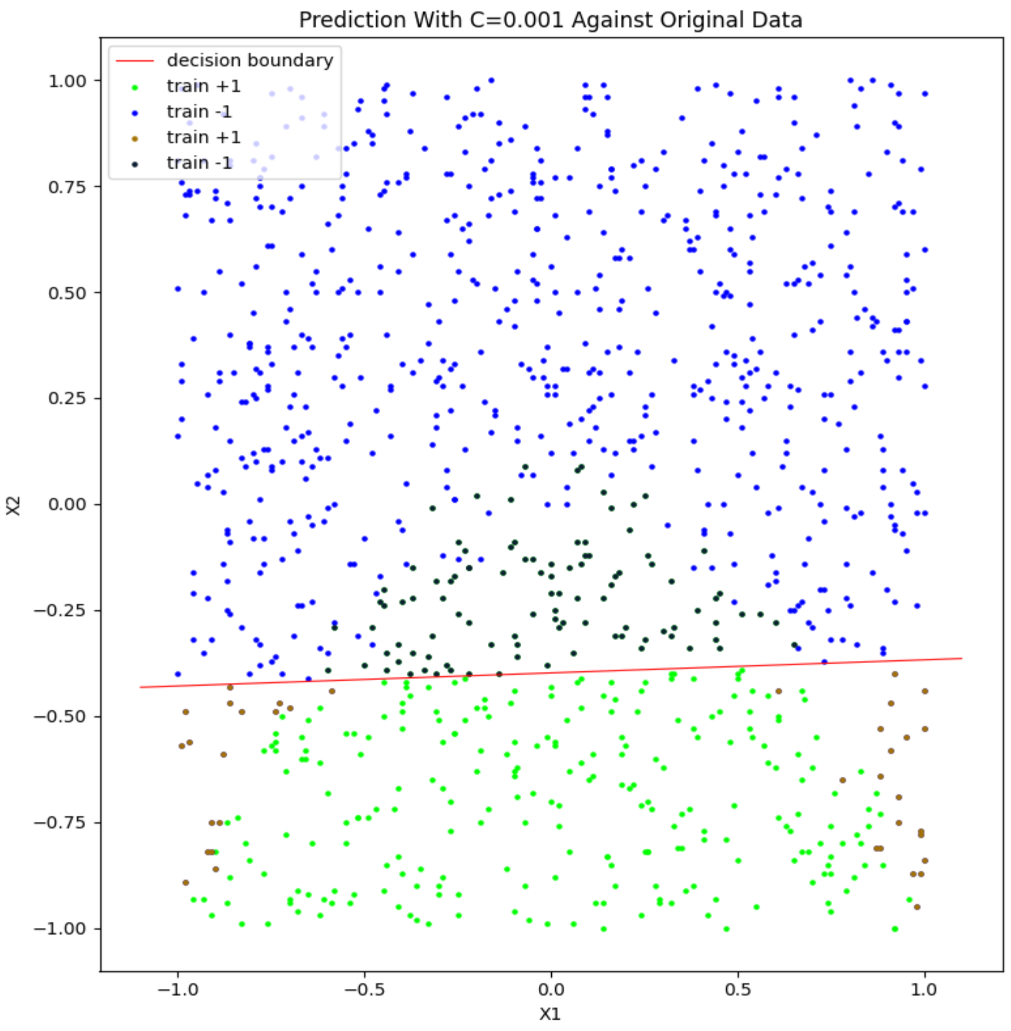
(iv) The predictions and the training data compare quite poorly, as the model that we got was a linear model, and the trend in our data showed that the model should have some quadratic nature to it, due to the parabola that is formed in the plot. When we compare the y's from the original dataset and the predicted dataset, the error that we get from this is 0.127127, meaning that even though it very poorly models the data, it has still identified a trend in the data and is able to predict the vast majority of the data.

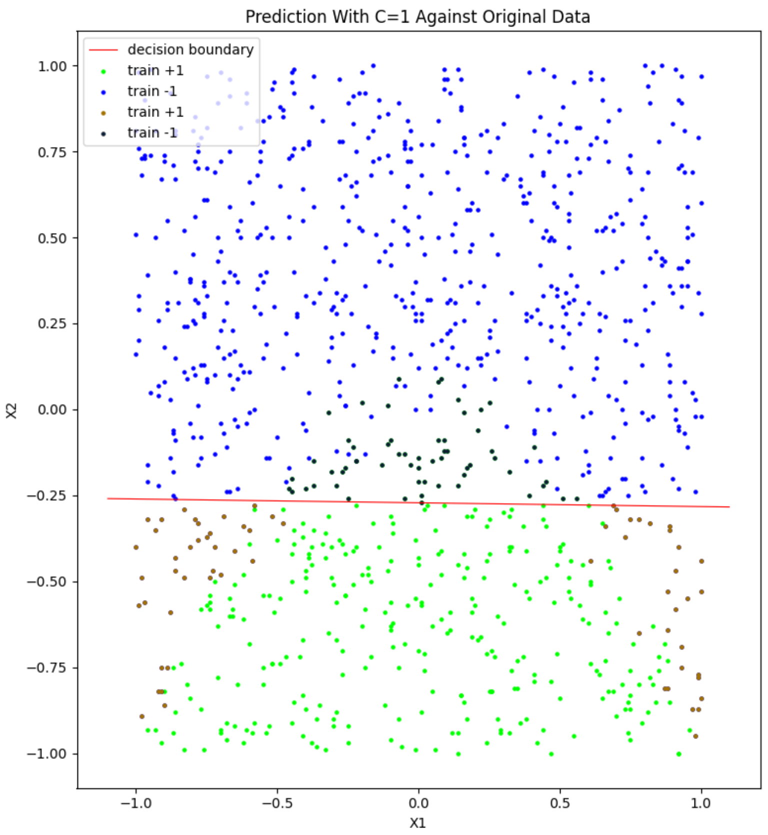
b. (i) For C = 0.001, I got an intercept of -0.200280, an x_1 slope of 0.007761 and an x_2 slope of -0.487292, giving us a model of y = 1 when $-0.200280 + 0.007761\ x_1 - 0.487292x_2 > 0$ and y = -1 when $-0.200280 + 0.007761\ x_1 - 0.487292x_2 < 0$.

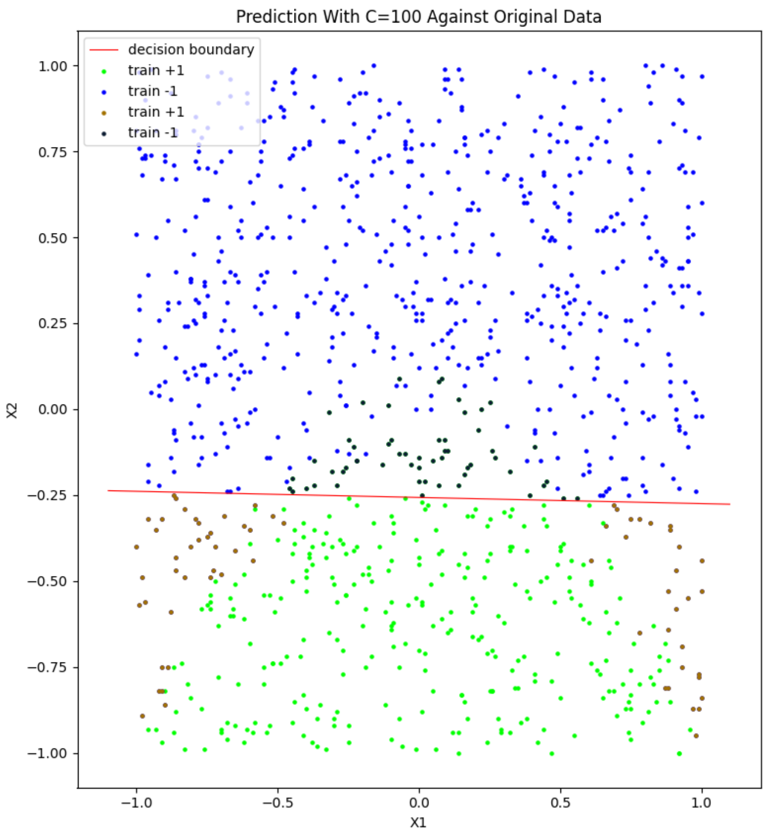
For C = 1, I got an intercept of -0.505597, an x_1 slope of -0.005463 and an x_2 slope of -1.821217, giving us a model of y = 1 when $-0.505597 - 0.005463x_1 - 1.821217x_2 > 0$ and y = -1 when $-0.505597 - 0.005463x_1 - 1.821217x_2 < 0$.

For C = 100, for the default iterations, the model failed to converge, sklearn still outputted what it had at the time however, so I got an intercept of -0.488823, an x_1 slope of -0.000093 and an x_2 slope of -1.856587, giving us a model of y = 1 when $-0.488823 - 0.000093x_1 - 1.856587x_2 > 0$ and y = -1 when $- 0.488823 - 0.000093x_1 - 1.856587x_2 < 0$.

(ii)





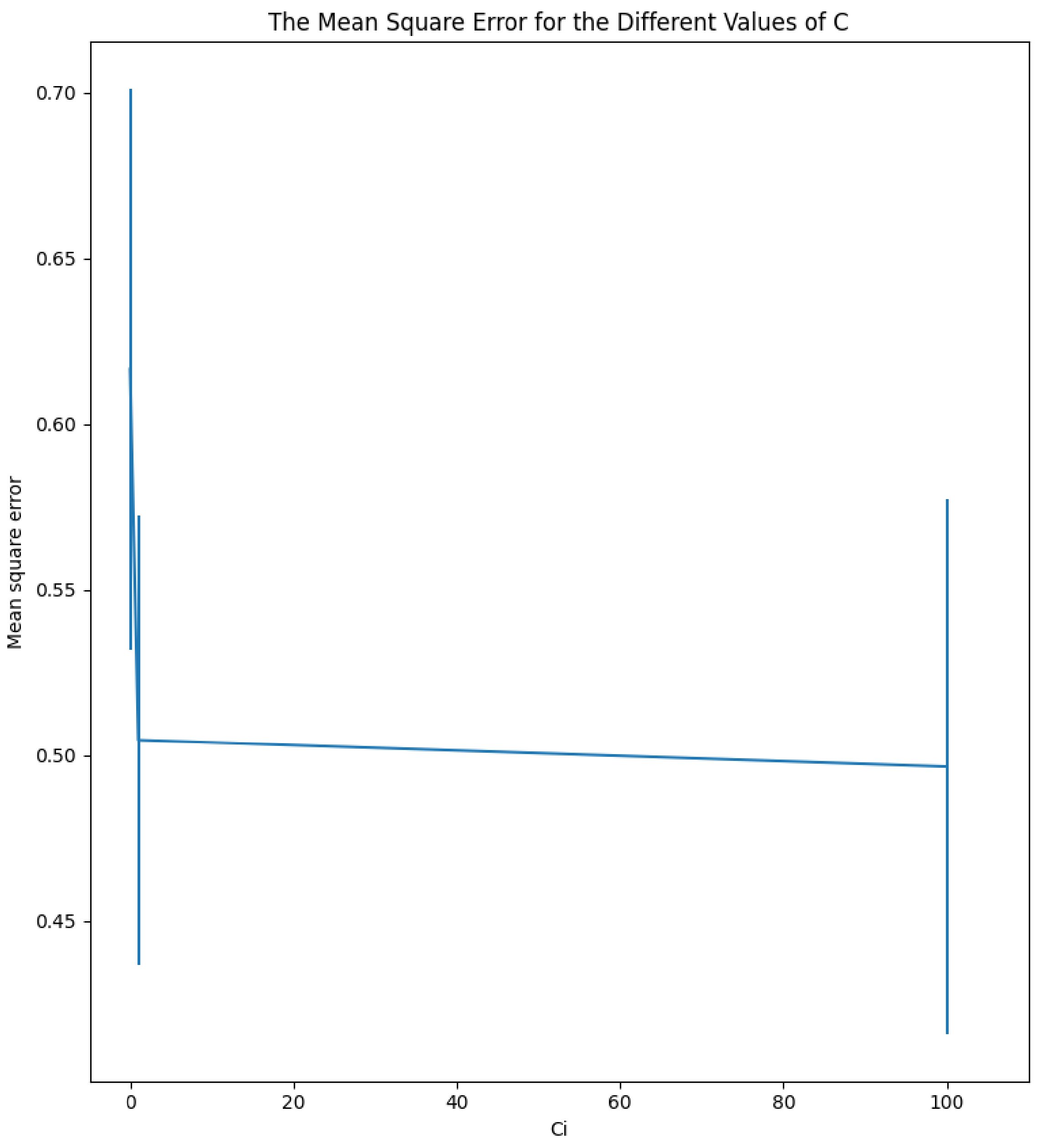


For C=0.001, the intercept is -0.1711872527099532, the slope of X1 is 0.013253244822098569 and the slope of X2 is -0.4300833374294709, this gave me a model of y = 1 when $-0.1711872527099532 + 0.013253244822098569x_1 - 0.4300833374294709x_2 > 0$ and y = 1 when $-0.1711872527099532 + 0.013253244822098569x_1 - 0.4300833374294709x_2 < 0$.

For C=1, the intercept is -0.47005934663674065, the slope of X1 is -0.019101240418555817 and the slope of X2 is -1.7321406014592164, this gave me a model of y = 1 when $-0.47005934663674065 - 0.019101240418555817x_1 - 1.7321406014592164 x_2 > 0$ and y = -1 when $-0.47005934663674065 - 0.019101240418555817x_1 - 1.7321406014592164 x_2 < 0$.

For C=100, the intercept is -0.4591322406278939, the slope of X1 is -0.030234596395931092 and the slope of X2 is -1.7840949484409825, this gave me a model of y = 1 when $-0.4591322406278939 - 0.030234596395931092x_1 - 1.7840949484409825 x_2 > 0$ and y = -1 when $-0.4591322406278939 - 0.030234596395931092x_1 - 1.7840949484409825 x_2 < 0$.

(iii)

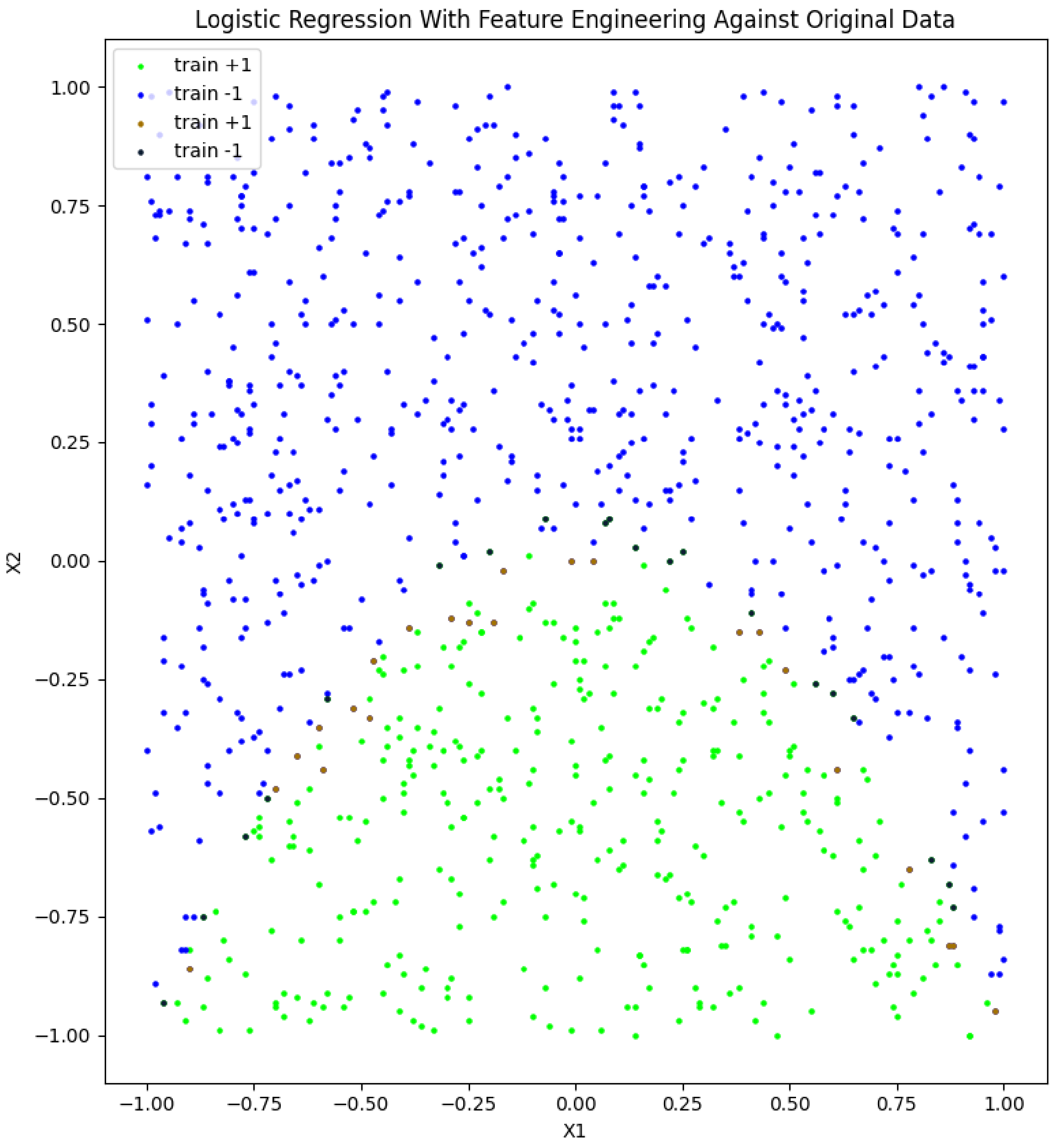


Decreasing the value of C will decrease the values of the parameters because from the regularisation used in SVM of $\Theta^T\Theta/C$, if C is large, this penalty will have a tiny effect on the data, but if it is very small, it will increase the effect of the penalty on the data, as the penalty is $\Theta^T\Theta/C$, the larger the Θ, the larger the penalty also. From the graph above, we see that the effect of the varying sizes of C on the predictions that the model makes, is that it changes the mean square error, this is because, as we decrease C and regularise the model, we make it better at generalizing, but this could also mean that we are under-fitting the data, however, when we we make it bigger, we tend to over-fit, we're looking for the sweet spot where it generalizes well but also clearly follows the general trend in the data and doesn't under-fit.

(iv) We can see from the model that Θ_2 is substantially smaller in the SVM model than in the logistic regression model, however from the predictions we got, apart from C=0.001, which is trying to reduce the amount of false negatives as much as possible, the models perform about the same, with all the other models not being very good at modelling the data either.

c. (i)For the feature engineered logistic regression model, I got an intercept of 0.713595, a slope for x_1 0.427286, slope x_2 -19.871368, slope x_3 -19.772823, slope x_4 -0.897317, square error 0.040000.

(ii)



This plot shows the original data against the predictions, but only shows the predictions which were different from the original data.

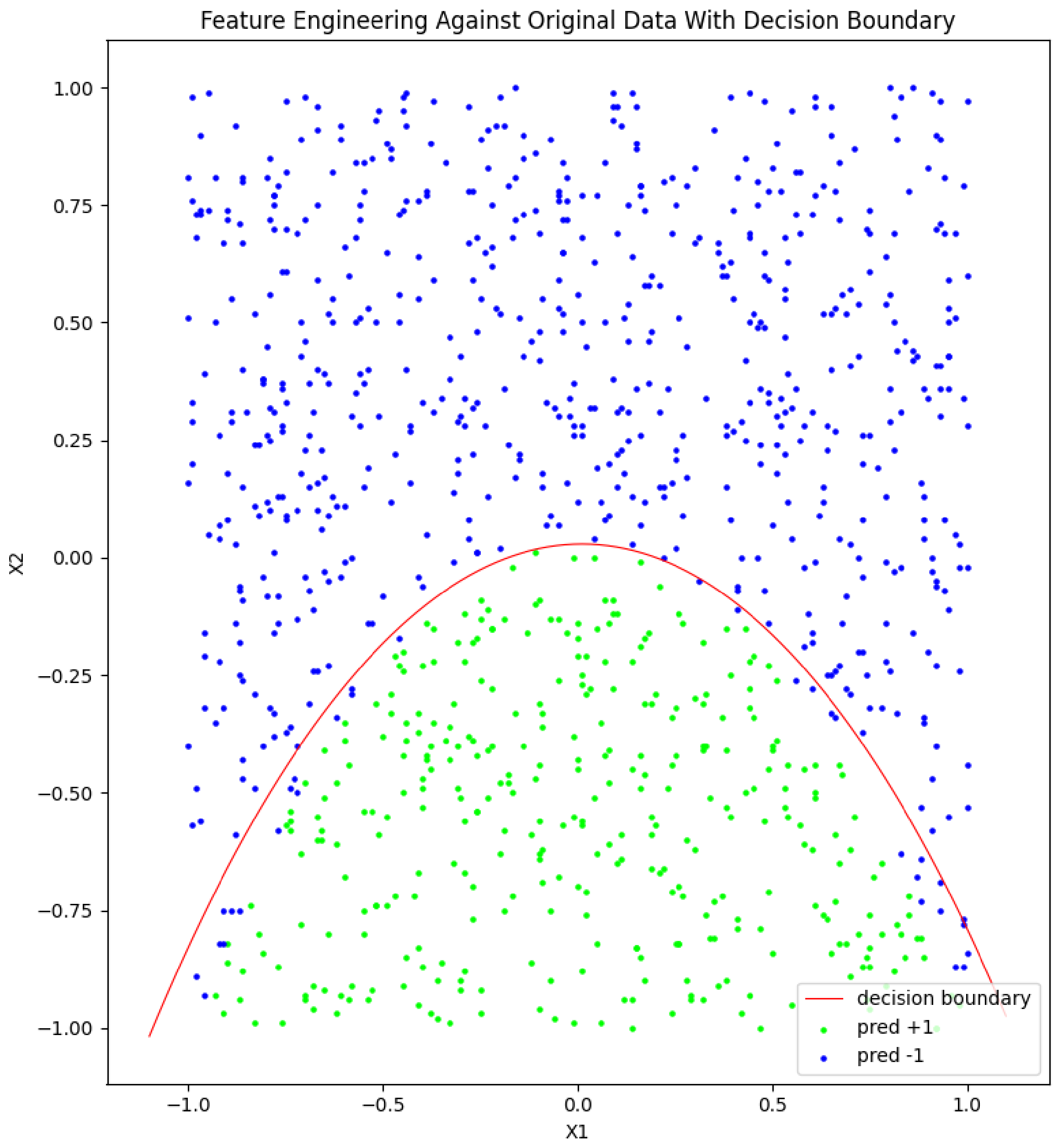
As we can see, the predictions attained from this model are significantly more accurate and representative of the data that we collected and trained against than the models that we got in part a) and b), this is very clearly due to the fact that we were attempting to fit an obviously none-linear dataset with a linear model, by implementing feature engineering, we have managed to model quadratic functionality using our logistic model. It should also be noted that we still have many points which are incorrect, this could be explained by many things, one of which being that, by leaving in x_1 and x_4, which have a very small effect on the model, we have actually left in features that do not actually effect the outcome of the data in reality, and so us leaving them in may mean that our predictions will always be slightly off because we are using features that don't contribute to the answer.

(iii) The baseline predictor I chose to compare the model too is a very simple model that simply chooses, for every single point, the outcome which is most common. In this case, it found that -1 was the most common outcome, so it chose -1 as the answer to all the points.

	Accuracy	True Positive Rate	False Positive Rate	Precision
Base Line	0.6546546546546547	0.0	0	Not Defined
Logistic Model	0.9569569569569569	0.8831521739130435	0	1

This shows us that our model is clearly better that a baseline predictor, as it best the predictor in every category.

(iv)



The curve does not perfectly separate the data because x_1^2 has a huge influence on the data, and is not plotted here, this is easy to see as the decision boundary splits the data better when closer to 0, and larger the closer to 1 or -1, because at 0, x_1^2 is at its smallest, and gets larger when approaching -1 and 1.

I got the curve by playing around with the model and with the determinant formula:

We start off with: $\Theta_0 + \Theta_1 x_1 + \Theta_2 x_2 + \Theta_3 x_3 + \Theta_4 x_4$

For the sake of clarity, we're going to call x_2 y and x_4 y^2

Doing this we get $\Theta_4y^2 + \Theta_2y + \Theta_3x_3 + \Theta_1x_1 + \Theta_0$

Which we can think of like this: $\Theta_4y^2 + \Theta_2y + (\Theta_3x_3 + \Theta_1x_1 + \Theta_0)$

We can then use the determinant formula to solve for y:

$y = \Theta_2$ +or- $(\Theta_2^2 – 4(\Theta_4)(\Theta_3x_3 + \Theta_1x_1 + \Theta_0))^{0.5} / 2(\Theta_4)$

Appendix

# Note: in order to make the provided code work, I needed to add ,0,1 to the first line of

```
# the data given so that it would correctly identify it as a csv file

# csv first line # id:2--2--2,0,1

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
from matplotlib import cm
from matplotlib.colors import ListedColormap

# takes in X1, X2 which have all values, regardless if y is 1 or -1
# returns two sets of 3 variables, holding the sets where y is 1 or y is -1
def seperator(df):
X1pos, X2pos, Ypos = [], [], []
X1neg, X2neg, Yneg = [], [], []

for _, row in df.iterrows():
if row['Y'] == 1:
X1pos.append(row['X1'])
X2pos.append(row['X2'])
Ypos.append(row['Y'])
else:
X1neg.append(row['X1'])
X2neg.append(row['X2'])
Yneg.append(row['Y'])

pos = {'X1': X1pos, 'X2': X2pos, 'Y': Ypos}
neg = {'X1': X1neg, 'X2': X2neg, 'Y': Yneg}
posData = pd.DataFrame(pos)
negData = pd.DataFrame(neg)
return posData, negData
```

```
df = pd.read_csv("./data.csv")
X1 = df.iloc[:,0]
X2 = df.iloc[:,1]
X = np.column_stack((X1,X2))
y = df.iloc[:, 2]

pos, neg = seperator(df)

new_set1 = cm.get_cmap('Set1', 2)

hsv_modified = cm.get_cmap('hsv')
newcmp = ListedColormap(hsv_modified(np.linspace(0.3,0.7,256)))

inferno_modified = cm.get_cmap('inferno')
predCmp = ListedColormap(inferno_modified(np.linspace(0.5,0.8,256)))

def plotOriginalData():
plt.scatter(data=pos, x='X1', y='X2', s=5, c='#00ff00', label='train +1')
plt.scatter(data=neg, x='X1', y='X2', s=5, c='#0000ff', label='train -1')
plt.xlabel('X1')
plt.ylabel('X2')

def plotAgainstOriginal(prediction, decisionBoundary=None):
plotOriginalData()
pred = pd.DataFrame({'X1': X1, 'X2': X2, 'Y': prediction})
newPos = {'X1': [], 'X2': []}
newNeg = {'X1': [], 'X2': []}
for index, row in pred.iterrows():
if y[index] != row['Y'] and y[index] == -1:
newPos['X1'].append(row['X1'])
newPos['X2'].append(row['X2'])
elif y[index] != row['Y'] and y[index] == 1:
newNeg['X1'].append(row['X1'])
newNeg['X2'].append(row['X2'])

plt.scatter(data=newPos, x='X1', y='X2', s=5, c='#a37200', label='train +1')
plt.scatter(data=newNeg, x='X1', y='X2', s=5, c='#0e1e33', label='train -1')

if decisionBoundary:
x = np.linspace(-1.1, 1.1, 100)
plt.plot(x, decisionBoundary(x), c='#ff0000', linewidth=.8, label='decision
boundary')
plt.legend()

def plotFinal(prediction, decisionBoundary):
pred = pd.DataFrame({'X1': X1, 'X2': X2, 'Y': prediction})
```

```
predPos, predNeg = seperator(pred)
plt.scatter(data=predPos, x='X1', y='X2', s=5, c='#00ff00', label='pred +1')
plt.scatter(data=predNeg, x='X1', y='X2', s=5, c='#0000ff', label='pred -1')
x = np.linspace(-1.1, 1.1, 100)
plt.plot(x, decisionBoundary(x), c='#ff0000', linewidth=.8, label='decision
boundary')
plt.xlabel('X1')
plt.ylabel('X2')
plt.legend()

# (a)
def questionA():
print("(a)")

# (i)
plotOriginalData()
plt.legend(['train +1', 'train -1'])
plt.title('Original Data')
plt.show()

# (ii)

##### modeling
bestModel = [LogisticRegression(), 10000]
kf = KFold(n_splits=5)
for train, test in kf.split(X):
model = LogisticRegression(penalty='none', solver='lbfgs').fit(X[train],
y[train])
ypred = model.predict(X[test])
squareError = mean_squared_error(y[test], ypred)
print("intercept %f, slope X1 %f, slope X2 %f, square error %f"%
(model.intercept_, model.coef_[0][0], model.coef_[0][1], squareError))
if bestModel[1] > squareError:
bestModel[0], bestModel[1] = model, squareError
model = bestModel[0]

# (iii)

ypred = model.predict(X)
m1, m2 = model.coef_[0]
c = model.intercept_
decisionBoundary = lambda x: ((m1 * x) + c) / -(m2)

plotAgainstOriginal(ypred, decisionBoundary)

plt.margins(x=0)
plt.title('Basic Logistic Regression Against Original Data')
plt.show()

# (iv)

# (b)
def questionB():
print("\n(b)\n")

# (i)
mean_error=[]; std_error=[]
meanErrorOfPredictions = []

smallCBest = [LinearSVC(), 10000]
kf = KFold(n_splits=5)
for train, test in kf.split(X):
model = LinearSVC(C=0.001).fit(X[train], y[train])
ypred = model.predict(X[test])
squareError = mean_squared_error(y[test], ypred)
if smallCBest[1] > squareError:
smallCBest[0], smallCBest[1] = model, squareError
meanErrorOfPredictions.append(squareError)
mean_error.append(np.array(meanErrorOfPredictions).mean());
std_error.append(np.array(meanErrorOfPredictions).std());
meanErrorOfPredictions = []

mediumCBest = [LinearSVC(), 10000]
kf = KFold(n_splits=5)
for train, test in kf.split(X):
model = LinearSVC(C=1).fit(X[train], y[train])
ypred = model.predict(X[test])
squareError = mean_squared_error(y[test], ypred)
if mediumCBest[1] > squareError:
mediumCBest[0], mediumCBest[1] = model, squareError
meanErrorOfPredictions.append(squareError)
mean_error.append(np.array(meanErrorOfPredictions).mean());
std_error.append(np.array(meanErrorOfPredictions).std());
meanErrorOfPredictions = []

largeCBest = [LinearSVC(), 10000]
kf = KFold(n_splits=5)
for train, test in kf.split(X):
model = LinearSVC(C=100).fit(X[train], y[train])
ypred = model.predict(X[test])
squareError = mean_squared_error(y[test], ypred)
if largeCBest[1] > squareError:
largeCBest[0], largeCBest[1] = model, squareError
meanErrorOfPredictions.append(squareError)
```

```
mean_error.append(np.array(meanErrorOfPredictions).mean())
std_error.append(np.array(meanErrorOfPredictions).std())

# (ii)

smallCModel = smallCBest[0]
mediumCModel = mediumCBest[0]
largeCModel = largeCBest[0]
ypredSmall = smallCModel.predict(X)
ypredMedium = mediumCModel.predict(X)
ypredLarge = largeCModel.predict(X)

print(f'\nFor C=0.001, the intercept is {smallCModel.intercept_[0]}, the slope
of X1 is {smallCModel.coef_[0][0]} and the slope of X2 is {smallCModel.coef_[0]
[1]}')
print(f'For C=1, the intercept is {mediumCModel.intercept_[0]}, the slope of X1
is {mediumCModel.coef_[0][0]} and the slope of X2 is {mediumCModel.coef_[0]
[1]}')
print(f'For C=100, the intercept is {largeCModel.intercept_[0]}, the slope of
X1 is {largeCModel.coef_[0][0]} and the slope of X2 is {largeCModel.coef_[0]
[1]}')

decisionBoundary = lambda x: ((smallCModel.coef_[0][0] * x) +
smallCModel.intercept_) / -(smallCModel.coef_[0][1])
plotAgainstOriginal(ypredSmall, decisionBoundary)
plt.title('Prediction With C=0.001 Against Original Data')
plt.show()

decisionBoundary = lambda x: ((mediumCModel.coef_[0][0] * x) +
mediumCModel.intercept_) / -(mediumCModel.coef_[0][1])
plotAgainstOriginal(ypredMedium, decisionBoundary)
plt.title('Prediction With C=1 Against Original Data')
plt.show()

decisionBoundary = lambda x: ((largeCModel.coef_[0][0] * x) +
largeCModel.intercept_) / -(largeCModel.coef_[0][1])
plotAgainstOriginal(ypredLarge, decisionBoundary)
plt.title('Prediction With C=100 Against Original Data')
plt.show()

# (iii)

Ci_range = [0.001, 1, 100]
plt.errorbar(Ci_range, mean_error, yerr=std_error)
plt.xlabel('Ci'); plt.ylabel('Mean square error')
plt.xlim((-5, 110))
plt.title('The Mean Square Error for the Different Values of C')
plt.show()

# (iv)

# (c)
def questionC():
print("\n(c)\n")

# (i)
XSquared = X * X
Xnew = np.c_[X, XSquared]

bestModel = [LogisticRegression(), 10000]
kf = KFold(n_splits=5)
for train, test in kf.split(Xnew):
model = LogisticRegression(penalty='none', solver='lbfgs').fit(Xnew[train],
y[train])
ypred = model.predict(Xnew[test])
squareError = mean_squared_error(y[test], ypred)
print("intercept %f, slope X1 %f, slope X2 %f, slope X3 %f, slope X4 %f, square
error %f"%(model.intercept_, model.coef_[0][0], model.coef_[0][1],
model.coef_[0][2], model.coef_[0][3], squareError))
if bestModel[1] > squareError:
bestModel[0], bestModel[1] = model, squareError
model = bestModel[0]

# (ii)
ypred = model.predict(Xnew)

plotAgainstOriginal(ypred)
plt.title('Logistic Regression With Feature Engineering Against Original Data')
plt.show()

# (iii) baseline predictor, one that predicts most common class simply
predicts 1 or -1 everytime depending on which is more common
mostCommonClass = 0
sum = 0
for i in y:
sum = sum + i
if sum < 0:
mostCommonClass = -1
elif sum > 0:
mostCommonClass = 1
else:
print("More complex baseline required")

# Get fp, fn, tp and tn's for both predictions
```

```
fpPred, fnPred, tpPred, tnPred = 0, 0, 0, 0
fpBase, fnBase, tpBase, tnBase = 0, 0, 0, 0
for i in range(len(y)):
if (y[i] == ypred[i]) & (y[i] == -1):
tnPred = tnPred + 1
elif (y[i] == ypred[i]) & (y[i] == 1):
tpPred = tpPred + 1
elif (y[i] != ypred[i]) & y[i] == 1:
fnPred = fnPred + 1
else:
fpPred = fpPred + 1

if (y[i] == mostCommonClass):
tnBase = tnBase + 1
elif (y[i] != mostCommonClass):
fnBase = fnBase + 1

print("%d %d %d %d" % (tpPred, tnPred, fpPred, fnPred))
print("%d %d %d %d" % (tpBase, tnBase, fpBase, fnBase))
acc = (tnPred + tpPred) / (tnPred + tpPred + fnPred + fpPred)
accBase = (tnBase + tpBase) / (tnBase + tpBase + fnBase + fpBase)
print(acc, accBase)

tpRate = (tpPred) / (tpPred + fnPred)
tpRateBase = (tpBase) / (tpBase + fnBase)
print(tpRate, tpRateBase)

fpRate = (fpPred) / (tnPred + fpPred)
fpRateBase = (fpBase) / (tnPred + fpPred)
print(fpRate, fpRateBase)

precision = (tpPred) / (tpPred + fpPred)
print(precision)

accDiff = ((tnPred + tpPred) / (tnPred + tpPred + fnPred + fpPred)) - ((tnBase
+ tpBase) / (tnBase + tpBase + fnBase + fpBase))
tpRateDiff = ((tpPred) / (tpPred + fnPred)) - ((tpBase) / (tpBase + fnBase))
fpRateDiff = ((fpPred) / (tnPred + fpPred)) - ((fpBase) / (tnPred + fpPred))
precisionDiff = ((tpPred) / (tpPred + fpPred))

print("%f %f %f %f" % (accDiff, tpRateDiff, fpRateDiff, precisionDiff))

# (iv)
m0, m1, m2, m3 = model.coef_[0]
c = model.intercept_

# determinant, where c = O3x^2 + O1x + O0
```

```
boundary_f = lambda x: (-m1 - pow(m1*m1 - 4*m3*(m2*x*x + m0*x + c), 0.5)) /
2*m3
plotFinal(ypred, boundary_f)
plt.title('Feature Engineering Against Original Data With Decision Boundary')
plt.show()

questionA()
questionB()
questionC()
```