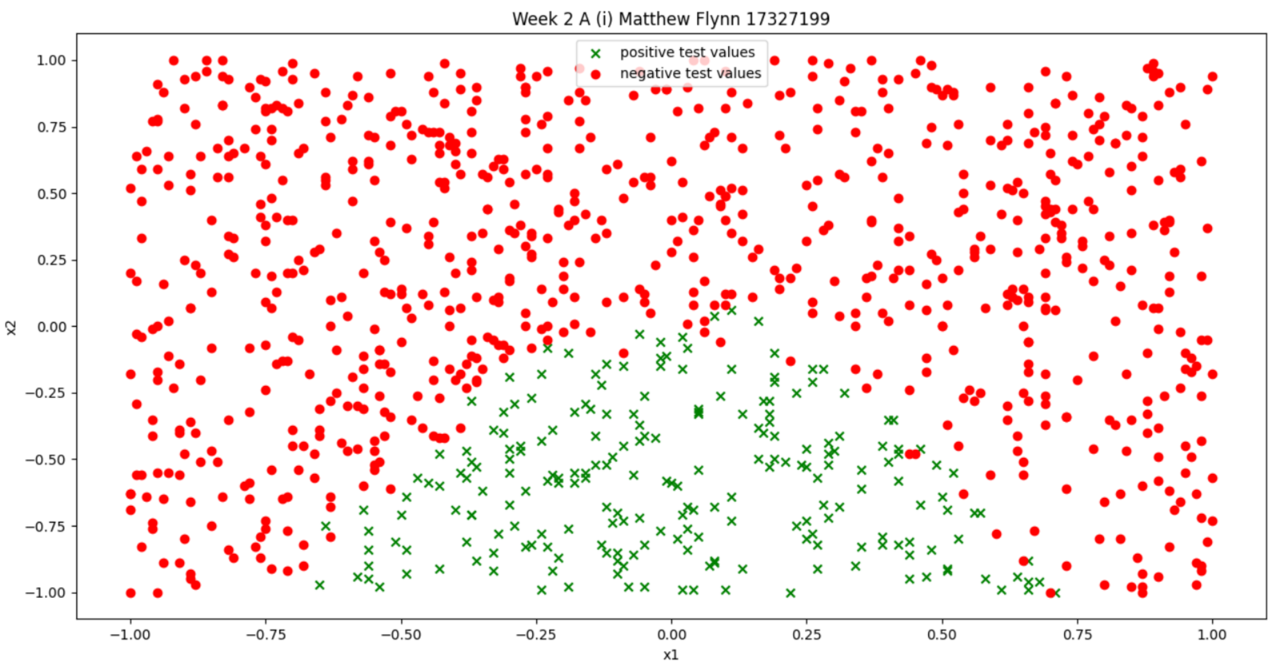
Week 2 assignment: # id:12--24--12

A (i) Data visualised

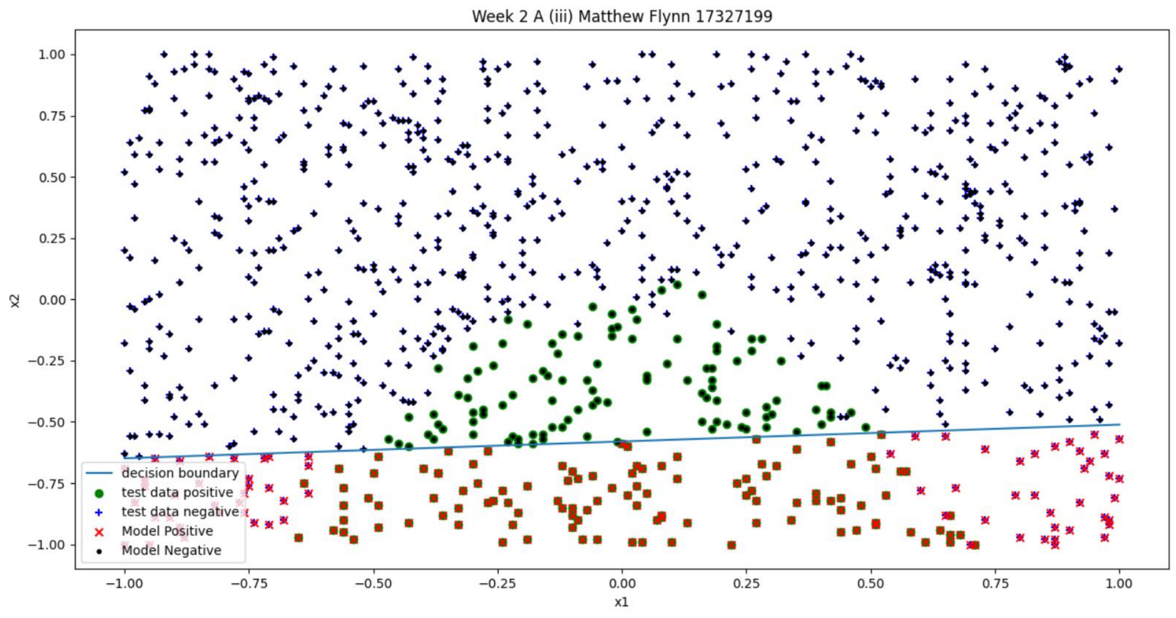


(ii) Trained logistic regression.

When the above data is used as training data to develop a model the following values were obtained. Intercept = [-2.1904738]. the 2 coefficients are [0.2598257, -3.78358625]. These are the parameter values.

(iii) Predict target values and show decision boundary.

To get the decision boundary the given parameters were used to create the line. X = [-1->1] and y =mx+c, where c = intercept/coeffiecent2 (-3.78358625), m = coef1/coef2. This will give us the equation of the decision boundary. The line and the test vs predicted values are shown below.

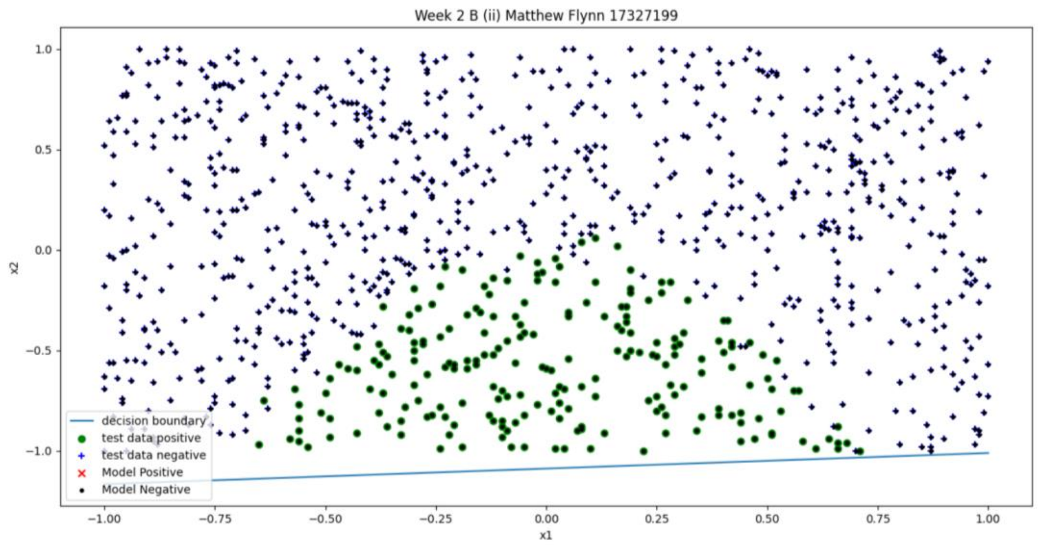


(iv)

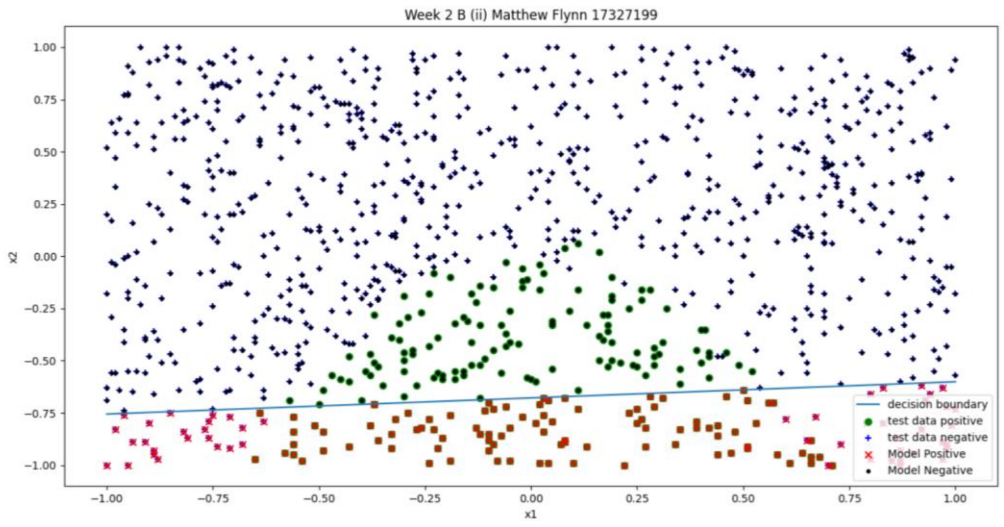
The model is attempting to create a linear decision boundary which keep the positives under the line and the negatives above the line. It is not the best fit for the test data as the data is closer to a quadratic. However, the model does work well within it's boundary. There is a only a small amount of false positives and positive negative. The model balances the prediction within it's limits but it is the wrong shape for the data.

B (i)
I used the following C values [0.001, 0.01,0.1,1,10] when I attempted to use 100 there was an error that the model wouldn't converge. This makes sense as C approaches infinite the cost becomes irrelevant to the cost function. For the values taken I got the following results.
C = 0.001 intercept = [-0.36498101] Coefficient = [[0.02601053 -0.33533873]]
C = 0.01 intercept = [-0.54931359] Coefficient = [[0.06258687 -0.8100746]]
C = 0.1 intercept = [-0.69048746] Coefficient = [[0.0953905 -1.18118502]]
C = 1.0 intercept = [-0.72642362] Coefficient = [[0.10140422 -1.26480628]]
C = 10.0 intercept = [-0.73053027] Coefficient = [[0.10203343 -1.27422807]]

(ii)
I visualised the data and decision boundary. It is interesting to not how off the boundary is when c is low, i.e the cost is high. As the cost becomes smaller the decision boundary becomes closer to that calculated in A(iii)
C = 0.001

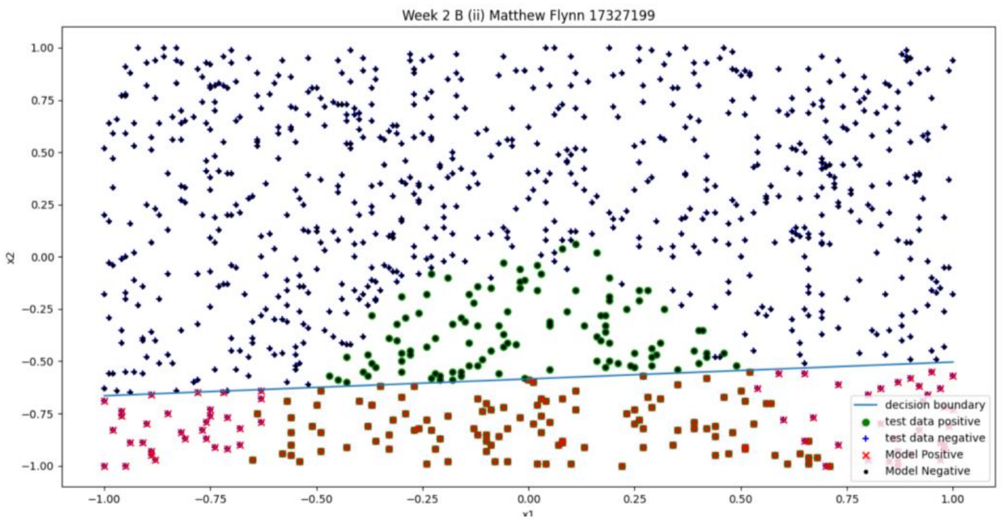


C = 0.01



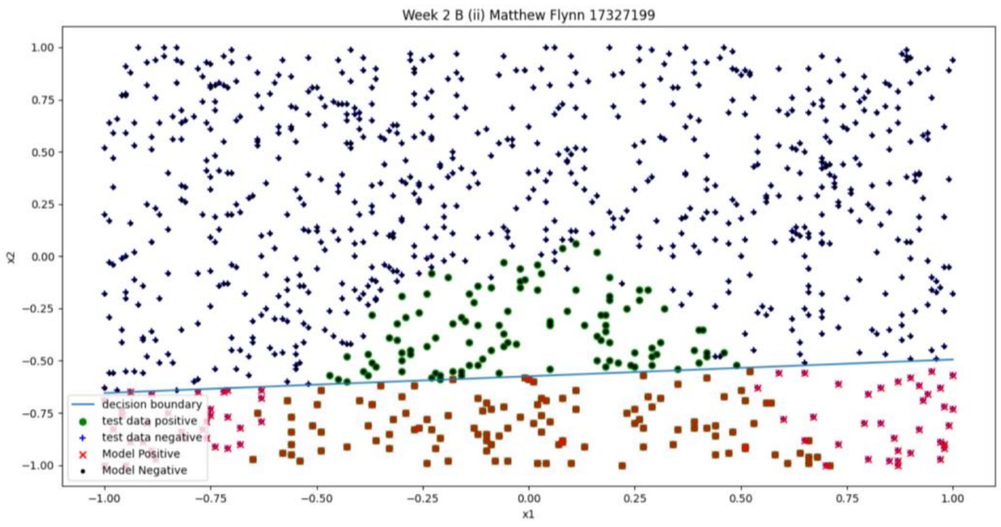
Week 2 B (ii) Matthew Flynn 17327199
x1
x2
decision boundary
test data positive
test data negative
Model Positive
Model Negative

C = 0.1



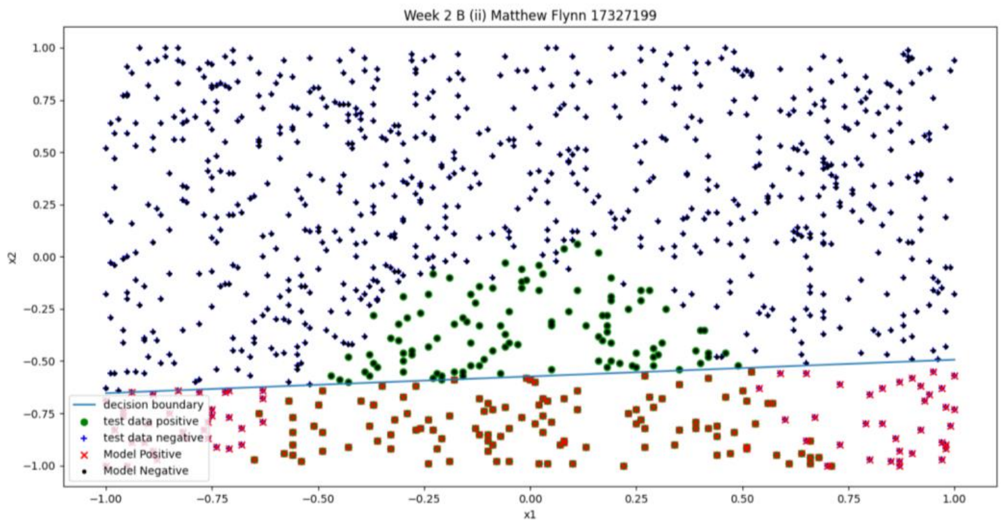
Week 2 B (ii) Matthew Flynn 17327199
x1
x2
decision boundary
test data positive
test data negative
Model Positive
Model Negative

C = 1



Week 2 B (ii) Matthew Flynn 17327199
x1
x2
decision boundary
test data positive
test data negative
Model Positive
Model Negative

C = 10

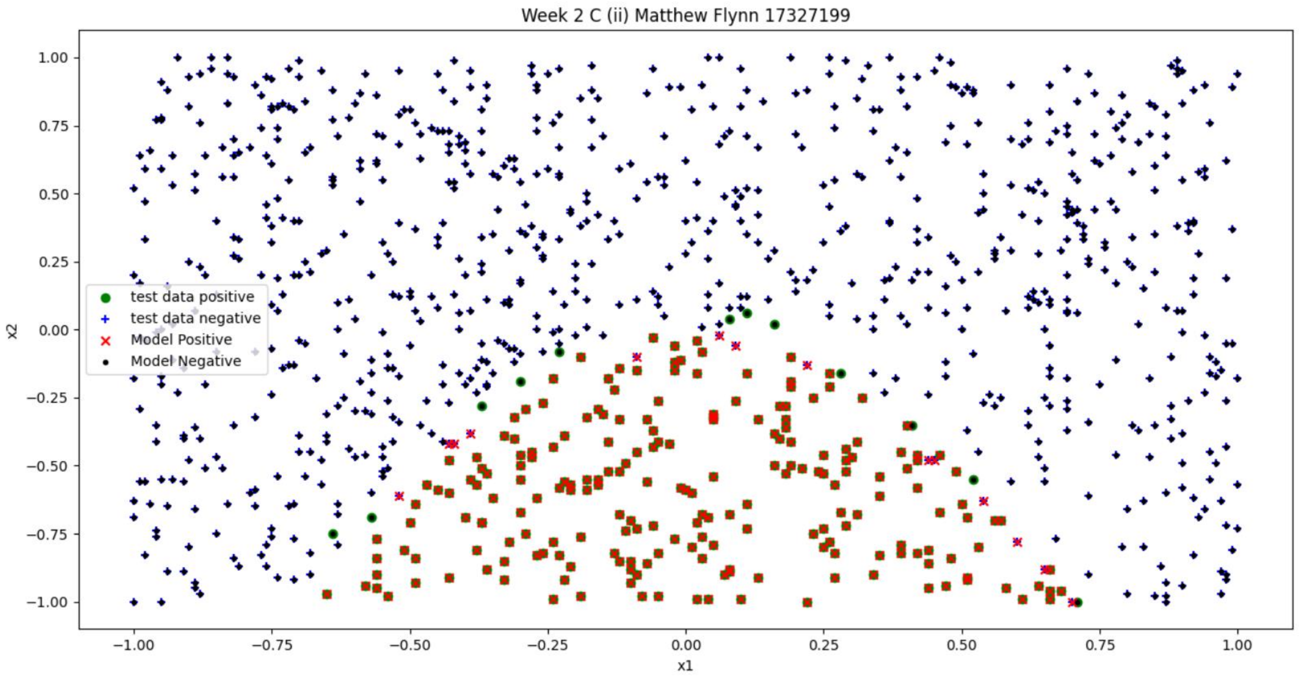


(iii) C is way of scaling a punishment on our cost function. As C gets lower the penalty placed on the cost function increases. This in practice causes large values of theta to be punished heavily. These heavy punishments make our parameters smaller and thus results in the decision boundary we see in 0.001. As C gets bigger the penalty approaches 0 until there is effectively no penalty, parameters increase to a point. Your model parameter's can therefore be altered greatly by a changing C as the cost function is altered by the penalty giving you different parameters producting the graphs seen above. Cost = $J(\emptyset) = \frac{1}{m}\sum_{i=1}^{m} \max\left(0,1 - y^i \emptyset^t x^i\right) + \frac{\emptyset^t \emptyset}{C}$.

C (i)

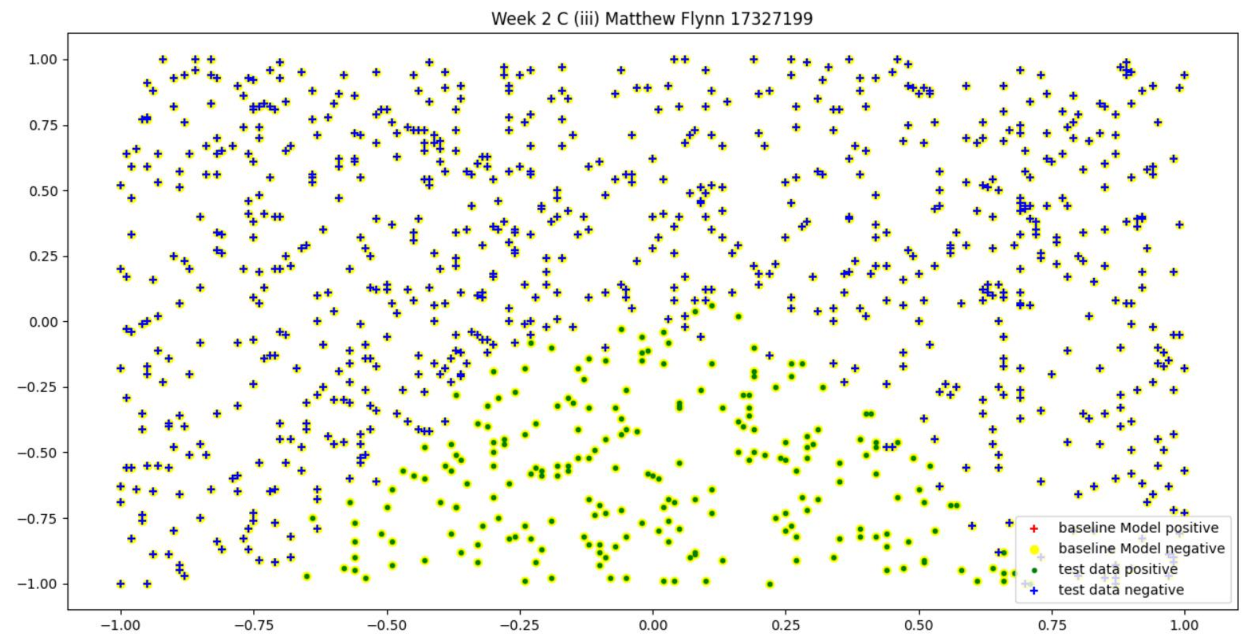
When adding the additional features $x3 = x1^2 and\ x4 = x2^2$ we now have 4 coefficients in addition to the intercepts. The calculated parameters are: intercept [0.10875784] coeff [[0.91926211 -22.87213311 -52.74133369 2.90279164]].

C(ii)



C (iii)

When the model is compared against a model which just predicts against a baseline model the base model simply assigns every value to be negative. The predicted model fairs much better and gives far better predictions.



As you can see when we compare the baseline with the test data and then look at how the trained model compares with the test data we can see that when using a baseline which simply predicts whichever class is more likely. My baseline sees how many of each class there are then assigns the class with the most members to all input data. This obviously shows that the logistical modelling can be very effective.

Appendix:

```
import numpy as np
import pandas as pd
from array import array
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC


# plot the data for part a (i)
def seperate_pos_neg(x1, x2, y):
    num_of_positives = 0
    index = 0
    pos_tmp_array = []
    pos_tmp_x_vals = []
    neg_tmp_x_vals = []
    neg_tmp_array = []
    index = 0
    while index < len(y):
        if y[index] == 1:
            pos_tmp_array.append(x2[index])
            pos_tmp_x_vals.append(x1[index])
        else:
            neg_tmp_array.append(x2[index])
            neg_tmp_x_vals.append(x1[index])
        index = index+1
    pos_vals = np.array(pos_tmp_array)
    neg_vals = np.array(neg_tmp_array)
    pos_x_vals = np.array(pos_tmp_x_vals)
    neg_x_vals = np.array(neg_tmp_x_vals)
    return pos_vals, neg_vals, pos_x_vals, neg_x_vals


def plot_descision_boundary(model):
    intercept = model.intercept_
    coef1, coef2 = model.coef_.T
    c = -intercept/coef2
    m = -coef1/coef2
    xd = np.array([min(x1), max(x1)])
    yd = m*xd + c
    plt.plot(xd, yd)


def plot_train-
ing_data_against_model(pos_y_vals, neg_y_vals, pos_x_vals, neg_x_vals, x, mode
l, title, legend):
    y_predict = model.predict(x)
    pos_y_vals_pred, neg_y_vals_pred, pos_x_vals_pred, neg_x_vals_pred = seper
ate_pos_neg(
        x1, x2, y_predict)
    pos_y_vals, neg_y_vals, pos_x_vals, neg_x_vals = seperate_pos_neg(
        x1, x2, y)
    plt.scatter(pos_x_vals, pos_y_vals, color="green", marker="o")
    plt.scatter(neg_x_vals, neg_y_vals, color="blue", marker="+")
    plt.scatter(pos_x_vals_pred, pos_y_vals_pred, color="red", marker="x")
    plt.scatter(neg_x_vals_pred, neg_y_vals_pred, color="black", marker=".")
    plt.title(title)
    plt.xlabel("x1")
    plt.ylabel("x2")
    plt.legend(legend)


def new_features(x1, x2):
    x3 = x1**2
    x4 = x2**2
    return x3, x4


def baselineModel(pos_x_vals, neg_x_vals, input_data_1, input_data_2):
    if len(pos_x_vals) > len(neg_x_vals):
        return input_data_1, input_data_2, [], []
    else:
        return [], [], input_data_1, input_data_2


# read in the data
df = pd.read_csv("week2.csv")
x1 = np.array(df.iloc[:, 0])
x2 = np.array(df.iloc[:, 1])
x = np.column_stack((x1, x2))
y = df.iloc[:, 2]

# a (i) visualise the data
pos_y_vals, neg_y_vals, pos_x_vals, neg_x_vals = seperate_pos_neg(x1, x2, y)
plt.scatter(pos_x_vals, pos_y_vals, color="green", marker="x")
plt.scatter(neg_x_vals, neg_y_vals, color="red", marker="o")
plt.title("Week 2 A (i) Matthew Flynn 17327199")
plt.xlabel("x1")
plt.ylabel("x2")
plt.legend(['positive test values', 'negative test values'])
plt.show()

# a (ii)
model = LogisticRegression(penalty="none", solver="lbfgs")
model.fit(x, y)
print("intercept", model.intercept_, "  coeff ", model.coef_)

# a (iii)
# get intercept and slope for the decision boundary
plot_descision_boundary(model)
plot_training_data_against_model(
    pos_y_vals, neg_y_vals, pos_x_vals, neg_x_vals, x, model, "Week 2 A (iii)
Matthew Flynn 17327199", ['decision boundary', 'test data posi-
tive', 'test data negative',

                          'Model Positive', 'Model Negative'])
plt.show()


# b (i)
# train SVM classifiers over a wide range of Cs
classifiers_List = np.array([0.001, 0.01, 0.1, 1, 10])
trained_Classifiers = []
for i in classifiers_List:
    model = LinearSVC(C=i).fit(x, y)
    print("C = ", i, "intercept = ", model.intercept_,
          "Coefficient = ", model.coef_)
    trained_Classifiers.append(model)

# b (ii)
# trained classifiers maintains the data.
for i in trained_Classifiers:
    y_predict = i.predict(x)
    pos_y_vals_pred, neg_y_vals_pred, pos_x_vals_pred, neg_x_vals_pred = seper
ate_pos_neg(
        x1, x2, y_predict)
    plot_descision_boundary(i)
    plot_train-
ing_data_against_model(pos_y_vals, neg_y_vals, pos_x_vals, neg_x_vals, x, i, "
Week 2 B (ii) Matthew Flynn 17327199", ['decision boundary', 'test data posi-
tive', 'test data negative',

                                                      'Model Posi-
tive', 'Model Negative'])
    plt.show()


# C (i)
# new features are x1 and x2 squared
x3, x4 = new_features(x1, x2)
new_features = np.column_stack((x1, x2, x3, x4))
model = LogisticRegression(penalty="none", solver="lbfgs")
model.fit(new_features, y)
print("intercept", model.intercept_, "  coeff ", model.coef_)

# C (ii)
plot_training_data_against_model(
    pos_y_vals, neg_y_vals, pos_x_vals, neg_x_vals, new_fea-
tures, model, "Week 2 C (ii) Matthew Flynn 17327199", ['test data posi-
tive', 'test data negative',

                                    'Model Positive', 'Model Negative'])
plt.show()

#C (iii)
base_x_pos, base_y_pos, base_x_neg, base_y_neg = baselineModel(
    pos_x_vals, neg_x_vals, x1, x2)
# model that predicts the same result for everything.
plt.scatter(base_x_pos, base_y_pos, color="red", marker='+')
plt.scatter(base_x_neg, base_y_neg, color="yellow", marker='o')
plt.title("Week 2 C (iii) Matthew Flynn 17327199")
pos_y_vals, neg_y_vals, pos_x_vals, neg_x_vals = seperate_pos_neg(
    x1, x2, y)
plt.scatter(pos_x_vals, pos_y_vals, color="green", marker=".")
plt.scatter(neg_x_vals, neg_y_vals, color="blue", marker="+")
plt.legend(['baseline Model positive', 'baseline Model negative',
            'test data positive', 'test data negative'])
plt.show()
```