Name: Karan Dua
Student Id: 21331391

Course Code: CS7CS4-202223 MACHINE LEARNING

Question (a): Data Loading

To read the data, I have used pandas library in python. The dataset had a comment, so I have removed it using the inbuilt feature of this function**. My data id is:15-15--15**:

```
df = pd.read_csv("data_week2.csv", names=["x1", "x2", "y"], header=None, comment='#')
display(df.head())
x1 = df.iloc[:,0]
x2 = df.iloc[:,1]
x = np.column_stack((x1, x2))
y = df.iloc[:,2]
```

Output: *The output of above code shows head of the data i.e. Features x1 and x2 and label y.*

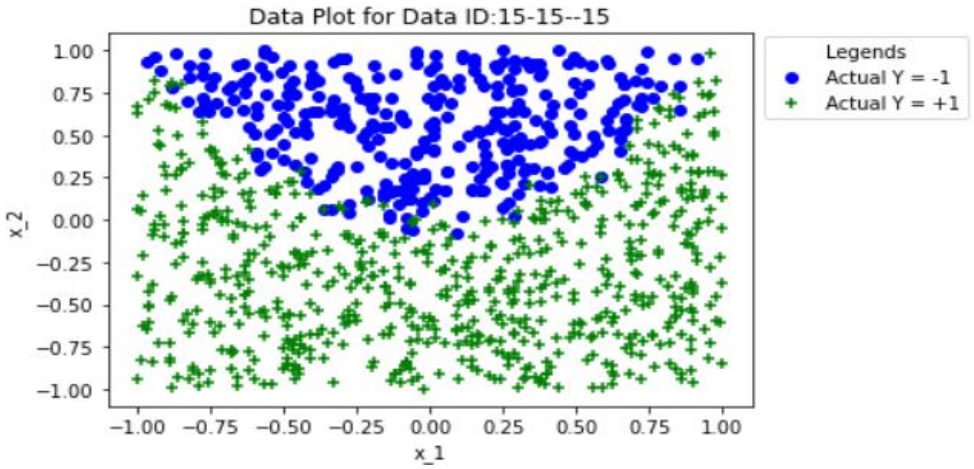
	x1	x2	y
0	-0.82	-0.67	1
1	-0.95	-0.09	1
2	0.06	0.28	-1
3	-0.06	-0.95	1
4	0.22	0.58	-1

Question (a)(i): Visualise the data

I have used matplot lib library to plot the scatterplot of data. I have used *'Blue o'* marker for x1 and x2 where y = -1 and *'Green +'* marker for x1 and x2 where y = +1.

```
plot.scatter(x1[y == -1], x2[y == -1], color='blue', marker="o")
plot.scatter(x1[y == 1], x2[y == 1], color='green', marker="+")
plot.title('Data Plot for Data ID:15-15--15')
plot.xlabel("x_1")
plot.ylabel("x_2")
plot.legend(['Actual Y = -1', 'Actual Y = +1'], loc = 'lower right', bbox_to_anchor=(1.35, 0.75), title="Legends")
plot.show()
```

Output:



By looking at the scatter plot, I can identify that there is bias in the data provided. There are significantly less points for label y = -1 when compared to y = +1 and the data is not linearly separable which means a straight line cannot separate both label classes. Both of these can have implications on model's performance, which can be checked later. To confirm the bias in data, I have used the following code:

```
y_postive = np.count_nonzero(y == 1)
y_negative = np.count_nonzero(y == -1)
actual_df = pd.DataFrame({"Actual Positive":[y_postive], "Actual Negative":[y_negative]})
actual_df
```

Output:

	Actual Positive	Actual Negative
0	675	324

From this, we can confirm that we have almost double points for y = +1 than y = -1 in given dataset. This clearly shows that we have bias in the data.

Question (a)(ii): Train Logistic Regression Classifier

I have used sklearn library to train the Logistic regression classifier model from the data using the following piece of code:

```
LRModel = LogisticRegression(solver='lbfgs', penalty="none")
LRModel.fit(x, y)
```

Parameter value for the trained model can be extracted using the following piece of code:

```
m1 = LRModel.coef_[0,0]
m2 = LRModel.coef_[0,1]
c = LRModel.intercept_[0]
model_params_df = pd.DataFrame({"Coefficient 1":[m1], "Coefficient 2":[m2], "Intercept":[c]})
display(model_params_df)
```

Output:

	Coefficient 1	Coefficient 2	Intercept
0	0.288599	-5.574721	1.782507

From the coefficient values, I can summarise that x2 had significantly higher influence on the prediction as it has much higher coefficient weight (5.57) as compared to coefficient weight of x1(0.288). Also, since the sign is negative it will cause prediction to decrease for the following reason. If we draw a simple equation from these parameters we get:

y = m1*x1 + m2*x2 + c, where m1 = Coefficient 1, m2 = Coefficient 2, and c = Intercept

So, y > 0 when, given x1 and x2, 0.288*x1 - 5.57*x2 + 1.78 is greater than 0

y < 0 when, given x1 and x2, 0.288*x1 - 5.57*x2 + 1.78 is less than 0

From this I can conclude that since m2 has significantly higher weight than m1, therefore x2 will have significant impact on predicted label y. Furthermore, I can conclude that x2 can cause prediction to decrease due negative weight of m2 model parameter.

Question (a)(iii): Logistic Regression Classifier Prediction and Decision Boundary

I have used predict function from the trained classifier to predict the values for variable y from given input variables x1 and x2 using the following code:

```
prdctns = LRModel.predict(x)
```

Decision Boundary:

Decision boundary is a line where probability of an event occurring and not occuring is exactly same i.e. 0.5. In our case, this line signifies the points where probability of y = 1 and y = -1 is 0.5

To get decision boundary, I have used the following probability equation for Logistic Regression which says:

Probability of event = $1 / (1 + e^{(m1*x1 + m2*x2 + c)})$a

Therefore, for decision boundary, Probability of event = $1 / (1 + e^{(m1*x1 + m2*x2 + c)}) = ½$, Solving this equation, we get:

$e^{(m1*x1 + m2*x2 + c)} = 1$, therefore m1*x1 + m2*x2 + c = 0

This means that for each value of x1, I have to find value of x2 for which m1*x1 + m2*x2 + c = 0. Now, to get decision boundary we can get the following equation:

x2 = (-1/m2) * ((m1 * x1) + c)

I have done same calculation using below code:

```
x2_dcsn_bndry = []
for i in range(len(x1)) :
    x2_calc = (-1/m2) * ((m1 * x1[i]) + c)
    x2_dcsn_bndry.append(x2_calc)
```

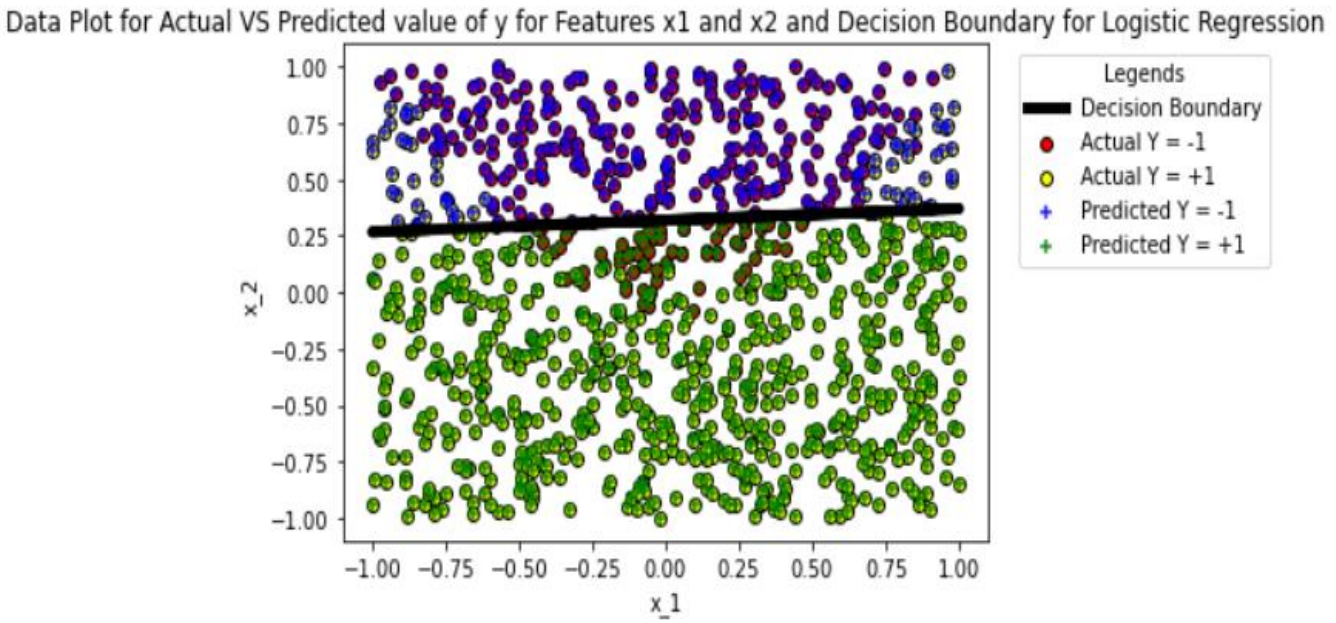
Here, for given x1 points, x2_dcsn_bndry signifies points lying on the decision boundary.

Scatterplot of training data, predictions and decision boundary:

This plot can be generated using the following code:

```
plot.scatter(x1[y == -1], x2[y == -1], color='red', marker="o", edgecolors="black")
plot.scatter(x1[y == 1], x2[y == 1], color='yellow', marker="o", edgecolors="black")
plot.scatter(x1[prdctns == -1], x2[prdctns == -1], color='blue', marker="+", alpha=0.85)
plot.scatter(x1[prdctns == 1], x2[prdctns == 1], color='green', marker="+", alpha=0.85)
plot.plot(x1, x2_dcsn_bndry, linewidth = 5, color = "black")
plot.title('Data Plot for Actual VS Predicted value of y for Features x1 and x2 and Decision Boundary for Logistic Regression')
```

```
plot.xlabel("x_1")
plot.ylabel("x_2")
plot.legend(['Decision Boundary', 'Actual Y = -1', 'Actual Y = +1', 'Predicted Y = -1', 'Predicted Y = +1'], title="Legends", loc = 'lower right', bbox_to_anchor=(1.45, 0.53))
plot.show()
```



I can identify that for y=1, correctly classified points are the one where training data marker 'Yellow o' is overlapped by Predicted marker is 'Green +' and for y=-1, correctly classified points are the one where training data marker 'Red o' is overlapped by Predicted marker is 'Blue +'
Also, I can identify that there are certain misclassifications where training data marker 'Yellow o' is overlapped by Predicted marker is 'Blue +' and training data marker 'Red o' is overlapped by Predicted marker is 'Green +'.

Question (a)(iv): Comment

I have used the following code to check the performance of our model:

```
true_positive = 0
true_negative = 0
false_positive = 0
false_negative = 0
for i in range(len(y)) :
    if y[i] == 1 and prdctns[i] == 1 :
        true_positive += 1
    elif y[i] == -1 and prdctns[i] == -1 :
        true_negative += 1
    elif y[i] == 1 and prdctns[i] == -1 :
        false_negative += 1
    elif y[i] == -1 and prdctns[i] == 1 :
        false_positive += 1

Accuracy = ((true_positive + true_negative) / (true_positive + true_negative + false_positive + false_negative)) * 100

y_pos_1_correctly_predicted = (true_positive/y_postive)*100
y_neg_1_correctly_predicted = (true_negative/y_negative)*100

actual_predicted_df = pd.DataFrame({"True Positive":[true_positive], "True Negative":[true_negative], "False Positive":[false_positive], "False Negative":[false_negative], "Accuracy":[Accuracy], "Label Y = 1 Accuracy":[y_pos_1_correctly_predicted], "Label Y = -1 Accuracy":[y_neg_1_correctly_predicted]})
display(actual_predicted_df)
```

Output:

	True Positive	True Negative	False Positive	False Negative	Accuracy	Label Y = 1 Accuracy	Label Y = -1 Accuracy
0	613	249	75	62	86.286286	90.814815	76.851852

Explanation of above table parameters:
True Positive are those labels where model has predicted a label to be 1 and it was actually 1
True Negative are those labels where model has predicted a label to be -1 and it was actually -1
False Positive are those labels where model has predicted a label to be 1 and it was actually -1
False Negative are those labels where model has predicted a label to be -1 and it was actually 1
Accuracy is the measured by calculating the percentage of correct labels predicted by the model
Label Y=1 Accuracy is the measured by calculating the percentage of correct y=1 labels predicted by the model
Label Y=-1 Accuracy is the measured by calculating the percentage of correct y=-1 labels predicted by the model
From the first scatterplot and decision boundary I have identified that data is not linearly separable i.e. both classes can be separated with a single line and from the scatterplot above, I can identify that this has impacted our model's performance. The plot clearly shows that there are points where misclassification has occurred due to this. The decision boundary also confirms this. It shows that some y=-1 labels are below the decision boundary which means they were misclassified by our model.
I have also identified that there was bias in data. From above table we can see that model correctly predicted 90.81% y=1 labels and labels y=-1 were predicted correctly only 76.85% labels which shows the impact of bias in the data on model's performance. Furthermore, there is a very high number of False Positive in the result which confirms that model tried to overfit y=1 label.

Considering above concerns our model has correctly predicted 86.38% of labels which is good score for the data.

Question (b)(i) and (ii): Train LinearSVC, Model Parameter values and Plots

To train LinearSVC on different values of penalty parameter C I have used the below code:

```
Penalty_Score = [0.001, 0.01, 1, 100]
model_dictionary = {}
for penalty in Penalty_Score :
    Linear_SVC_Model =  LinearSVC(C = penalty, max_iter=1200000)
    Linear_SVC_Model.fit(x, y)
    model_dictionary[penalty] = Linear_SVC_Model
```

To extract parameter values and plot these values, I used the below code:

```
for key in model_dictionary :
    Linear_SVC_Model = model_dictionary[key]
    m1 = Linear_SVC_Model.coef_[0,0]
    m2 = Linear_SVC_Model.coef_[0,1]
    c = Linear_SVC_Model.intercept_[0]

    model_params_df = pd.DataFrame({"Penalty":[key], "Coefficient 1":[m1], "Coefficient 2":[m2], "Intercept":[c]})
    display(model_params_df)

    x2_dcsn_bndry = []
    for i in range(len(x1)) :
        x2_calc = (-1/m2) * ((m1 * x1[i]) + c)
        x2_dcsn_bndry.append(x2_calc)
    prdctns = Linear_SVC_Model.predict(x)

    true_positive = 0
    true_negative = 0
    false_positive = 0
    false_negative = 0
    for i in range(len(y)) :
        if y[i] == 1 and prdctns[i] == 1 :
            true_positive += 1
        elif y[i] == -1 and prdctns[i] == -1 :
            true_negative += 1
        elif y[i] == 1 and prdctns[i] == -1 :
            false_negative += 1
        elif y[i] == -1 and prdctns[i] == 1 :
            false_positive += 1

    Accuracy = ((true_positive + true_negative) / (true_positive + true_negative + false_positive + false_negative)) * 100
```

```
y_pos_1_correctly_predicted = (true_positive/y_postive)*100

y_neg_1_correctly_predicted = (true_negative/y_negative)*100

actual_predicted_df = pd.DataFrame({"Penalty":[key], "True Positive":[true_positive], "True Negative":[true_negative], "False Positive":[false_positive], "False Negative":[false_negative], "Accuracy":[Accuracy], "Label Y = 1 Accuracy":[y_pos_1_correctly_predicted], "Label Y = -1 Accuracy":[y_neg_1_correctly_predicted]})
display(actual_predicted_df)


plot.figure()
plot.scatter(x1[y == -1], x2[y == -1], color='red', marker="o", edgecolors="black")
plot.scatter(x1[y == 1], x2[y == 1], color='yellow', marker="o", edgecolors="black")
plot.scatter(x1[prdctns == -1], x2[prdctns == -1], color='blue', marker="+", alpha=0.85)
plot.scatter(x1[prdctns == 1], x2[prdctns == 1], color='green', marker="+", alpha=0.85)
plot.plot(x1, dcsn_bndry, linewidth = 5, color = "black")
plot.title('Data Plot for Actual VS Predicted value of y for Features x1 and x2 and Decision Boundary for SVC with Penalty Score: %.3f' %key)
plot.xlabel("x_1")
plot.ylabel("x_2")
plot.legend(['Decision Boundary', 'Actual Y = -1', 'Actual Y = +1', 'Predicted Y = -1', 'Predicted Y = +1'], title="Legends", loc = 'lower right', bbox_to_anchor=(1.45, 0.53))
plot.show()
```

Model Parameters:

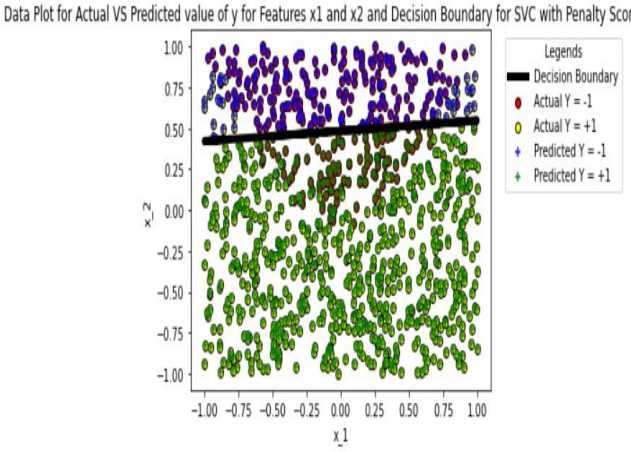
	Penalty	Coefficient 1	Coefficient 2	Intercept
0	0.001	0.029581	-0.468882	0.226969

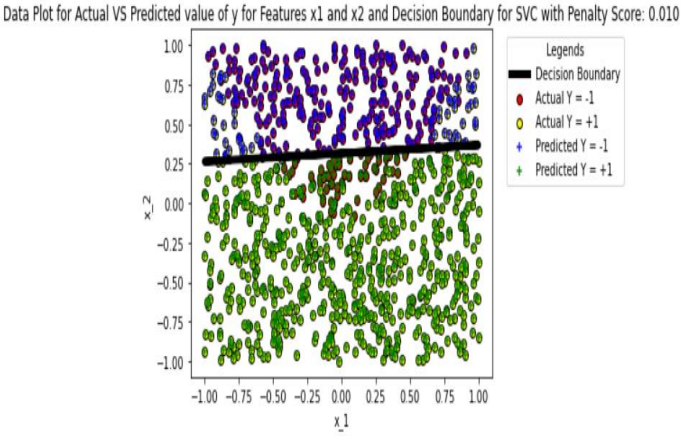
	Penalty	Coefficient 1	Coefficient 2	Intercept
0	0.01	0.060882	-1.167368	0.368237

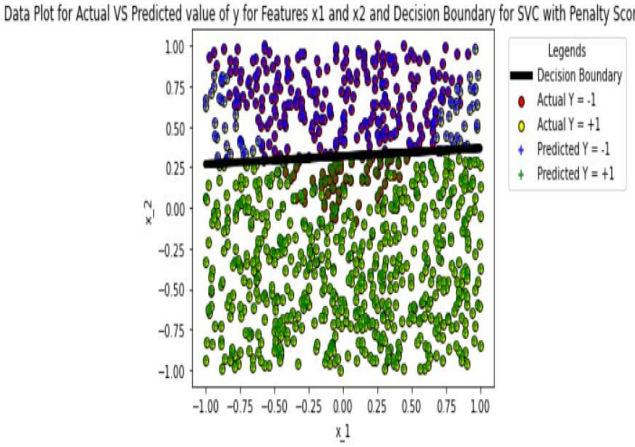
	Penalty	Coefficient 1	Coefficient 2	Intercept
0	1	0.090034	-1.827035	0.582153

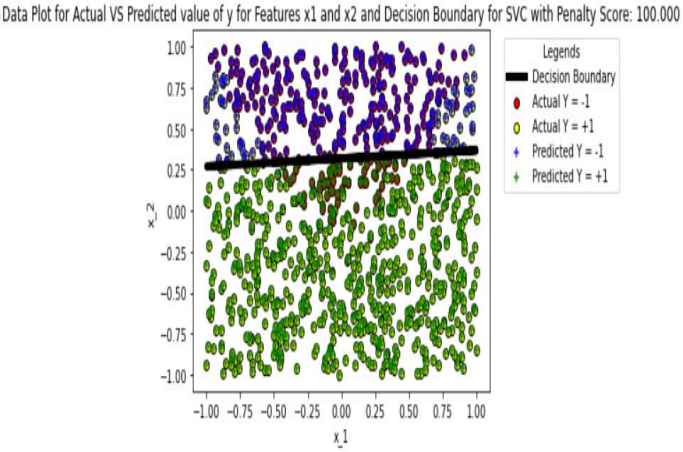
	Penalty	Coefficient 1	Coefficient 2	Intercept
0	100	0.090795	-1.84657	0.588781

Scatter Plots:









Explanation for above coefficients and scatterplots is given in the answer of below question.

Question (b)(iii) and (iv): Impact of Penalty on model and comparison with Logistic Regression

I can clearly see the impact of penalty parameter on the model coefficients.
For penalty = 0.001, we get equation 0.029*x1 - 0.468*x2 + 0.226
For penalty = 0.010, we get equation 0.060*x1 - 1.167*x2 + 0.368
For penalty = 1, we get equation 0.090 *x1 - 1.827*x2 + 0.582
For penalty = 100, we get equation 0.090*x1 - 1.846*x2 + 0.588
Now, as we increase the penalty value from 0.001 to 0.010 to 1, the weight of model parameter increased significantly which shows the decrease in slope of the decision boundary in the scatter plot. But as we increase penalty value from 1 to 100, the model parameters were almost similar indicating that the model has converged.

Also, for low value of penalty score, I can identify from scatter plot that slope of decision boundary is steeper showing that it might not separate classes of y well enough. This indicates that model might underfit data for negative values and overfit data for positive values of y. This overfitting can result in higher chances of misclassification of predicted label which means that model has higher chance to predict label of y as 1 when it might actually be -1 and vice-versa. However, for higher value of penalty, we see a less steep slope of decision boundary which shows that as we increase penalty value model tend to decrease the overfitting of data. We can confirm these results from below tables for model performance:

	Penalty	True Positive	True Negative	False Positive	False Negative	Accuracy	Label Y = 1 Accuracy	Label Y = -1 Accuracy
0	0.001	640	203	121	35	84.384384	94.814815	62.654321

For Penalty=0.001, we see that model has correctly predicted 94.81% y=1 label and y=-1 only 62.65%. Furthermore, the number of False Positive result in the model are extremely high. This clearly shows the underfitting of data for y=-1 and overfitting for label y=1.

	Penalty	True Positive	True Negative	False Positive	False Negative	Accuracy	Label Y = 1 Accuracy	Label Y = -1 Accuracy
0	0.01	613	252	72	62	86.586587	90.814815	77.777778

For Penalty=0.010, we see decrease in the underfitting of data for y=-1 as correct predictions significantly increased from 62.65% to 77.78%. Furthermore, the number of False Positive results has also decreased significantly. The overall correct predictions have also improved from 84.38% to 86.58%. This confirms that Overfitting of y=1 label has also reduced by the model.

	Penalty	True Positive	True Negative	False Positive	False Negative	Accuracy	Label Y = 1 Accuracy	Label Y = -1 Accuracy
0	1	613	249	75	62	86.286286	90.814815	76.851852

For Penalty=1, we see decrease in the underfitting of data for y=-1 as correct predictions significantly increased from 62.65% to 76.85%. The overall correct predictions have also improved from 84.38% to 86.38%

	Penalty	True Positive	True Negative	False Positive	False Negative	Accuracy	Label Y = 1 Accuracy	Label Y = -1 Accuracy
0	100	613	249	75	62	86.286286	90.814815	76.851852

For penalty=100, we no change in the performance parameters indicating that model has already converged.

When compared with Logistic regression, the performance of LinearSVC will depend largely on the penalty parameter chosen. For the given dataset, if penalty parameter chosen is very low like 0.001, I have seen that the model overfits the dominant class and we get subpar performance from LinearSVC model as it predicts only 84% labels correctly but if I chose the correct value of penalty parameter like 0.010 or 1, I am able to get a similar performance from LinearSVC model as Logistic Regression model. Therefore, I can conclude that penalty parameter plays a key role to achieve the best performance from a LinearSVC model.

Question (c)(i): Add square of features, Train Logistic Regression model and Parameter values

I have used below code to calculate square of features x1 and x2 and add them to data:

```
df = pd.read_csv("data_week2.csv", names=["x1", "x2", "y"], header=None, comment='#')
display(df.head())
x1 = df.iloc[:,0]
x2 = df.iloc[:,1]
x1_2 = x1 ** 2
x2_2 = x2 ** 2
x = np.column_stack((x1, x2, x1_2, x2_2))
y = df.iloc[:,2]
df_new = pd.DataFrame({"x1":x1, "x2":x2, "x1_2":x1_2, "x2_2":x2_2, "y":y})
display(df_new.head())
```

Ouput:

Name: Karan Dua
Student Id: 21331391

Course Code: CS7CS4-202223 MACHINE LEARNING

	x1	x2	y
0	-0.82	-0.67	1
1	-0.95	-0.09	1
2	0.06	0.28	-1
3	-0.06	-0.95	1
4	0.22	0.58	-1

Original Data

	x1	x2	x1_2	x2_2	y
0	-0.82	-0.67	0.6724	0.4489	1
1	-0.95	-0.09	0.9025	0.0081	1
2	0.06	0.28	0.0036	0.0784	-1
3	-0.06	-0.95	0.0036	0.9025	1
4	0.22	0.58	0.0484	0.3364	-1

New Data with squared

To train the model and extract model coefficients, I have used below code:

```
LRModel = LogisticRegression(solver='lbfgs', penalty="none")
LRModel.fit(x, y)
m1 = LRModel.coef_[0,0]
m2 = LRModel.coef_[0,1]
m3 = LRModel.coef_[0,2]
m4 = LRModel.coef_[0,3]
c = LRModel.intercept_[0]

model_params_df = pd.DataFrame({"Coefficient 1":[m1], "Coefficient 2":[m2], "Coefficient 3":[m3], "Coefficient 4":[m4], "Intercept":[c]})
display(model_params_df)
```

Output:

	Coefficient 1	Coefficient 2	Coefficient 3	Coefficient 4	Intercept
0	0.600782	-23.691791	25.257421	-0.330992	-0.24384

So, the model equation can be written as:

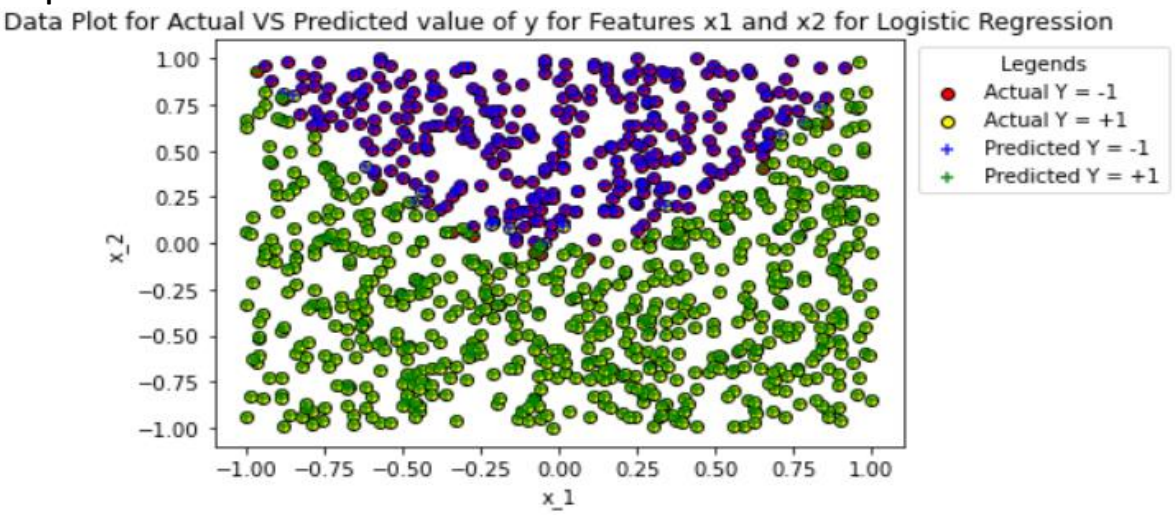
$0.600*x1 - 23.691*x2 + 25.257*x1^2 - 0.330*x2^2 - 0.243 = y$

Question (c)(ii): LR model predictions, plot of data points and compare performance with (a) and (b)

I have used below code to predict label of y and to plot the data points.

```
prdctns = LRModel.predict(x)
plot.scatter(x1[y == -1], x2[y == -1], color='red', marker="o", edgecolors="black")
plot.scatter(x1[y == 1], x2[y == 1], color='yellow', marker="o", edgecolors="black")
plot.scatter(x1[prdctns == -1], x2[prdctns == -1], color='blue', marker="+", alpha=0.85)
plot.scatter(x1[prdctns == 1], x2[prdctns == 1], color='green', marker="+", alpha=0.85)
plot.title('Data Plot for Actual VS Predicted value of y for Features x1 and x2 for Logistic Regression')
plot.xlabel("x_1")
plot.ylabel("x_2")
plot.legend(['Actual Y = -1', 'Actual Y = +1', 'Predicted Y = -1', 'Predicted Y = +1'], title="Legends", loc = 'lower right', bbox_to_anchor=(1.4, 0.6))
plot.show()
```

Output:



From the scatterplot I can observe that the misclassification of error has reduced significantly. This means that our model is able to fit the non-linearity in data. This also means that decision boundary is also non-linear. This might be due to addition of new features in the data. These features have provided model some extra information to predict the label. Below table gives more details about model's performance.

y_postive = np.count_nonzero(y == 1)

```
y_negative = np.count_nonzero(y == -1)
true_positive = 0
true_negative = 0
false_positive = 0
false_negative = 0
for i in range(len(y)) :
    if y[i] == 1 and prdctns[i] == 1 :
        true_positive += 1
    elif y[i] == -1 and prdctns[i] == -1 :
        true_negative += 1
    elif y[i] == 1 and prdctns[i] == -1 :
        false_negative += 1
    elif y[i] == -1 and prdctns[i] == 1 :
        false_positive += 1

Accuracy = ((true_positive + true_negative) / (true_positive + true_negative + false_positive + false_negative)) * 100
y_pos_1_correctly_predicted = (true_positive/y_postive)*100
y_neg_1_correctly_predicted = (true_negative/y_negative)*100
actual_predicted_df = pd.DataFrame({"True Positive":[true_positive], "True Negative":[true_negative], "False Positive":[false_positive], "False Negative":[false_negative], "Accuracy":[Accuracy], "Label Y = 1 Accuracy":[y_pos_1_correctly_predicted], "Label Y = -1 Accuracy":[y_neg_1_correctly_predicted]})
display(actual_predicted_df)
```

Output:

	True Positive	True Negative	False Positive	False Negative	Accuracy	Label Y = 1 Accuracy	Label Y = -1 Accuracy
0	661	310	14	14	97.197197	97.925926	95.679012

I can conclude from the above table that model has significantly reduced the underfitting of data for y=-1 as the number of False Positive and False Negative has reduced to a very low number and it was able to correctly predict 95.67% labels. Overall, the model's performance is better than in (a) and (b) as it was able to predict 97.19% labels correctly compared to 86% correct predictions by earlier models. Thus, we can conclude that model has significantly reduced the overfitting problem for y=1 labels due to addition of new features.

Question (c)(iii): Comparison with baseline model

Baseline model is a model that always predicts the most common label. To find most common label in our data, I have used below code:

```
y_postive = np.count_nonzero(y == 1)
y_negative = np.count_nonzero(y == -1)
actual_predicted_df = pd.DataFrame({"Actual Positive":[y_postive], "Actual Negative":[y_negative]})
display(actual_predicted_df)
```

Output:

	Actual Positive	Actual Negative
0	675	324

Based on the above table, I can conclude that y=1 is the most common label, therefore our model should always predict 1. To achieve this, I used below code

```
baseline_model_prediction = np.ones(len(y))
```

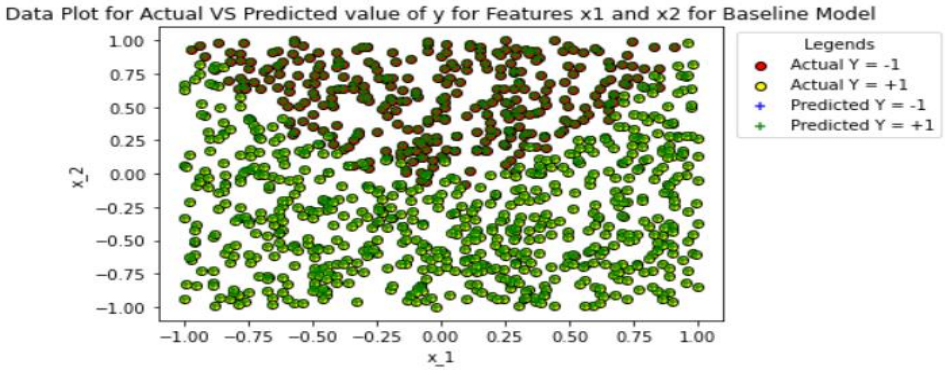
To plot the predictions of baseline model against original features I used below code:

```
plot.scatter(x1[y == -1], x2[y == -1], color='red', marker="o", edgecolors="black")
plot.scatter(x1[y == 1], x2[y == 1], color='yellow', marker="o", edgecolors="black")
plot.scatter(x1[baseline_model_prediction == -1], x2[baseline_model_prediction == -1], color='blue', marker="+", alpha=0.85)
plot.scatter(x1[baseline_model_prediction == 1], x2[baseline_model_prediction == 1], color='green', marker="+", alpha=0.85)
plot.title('Data Plot for Actual VS Predicted value of y for Features x1 and x2 for Baseline Model')
plot.xlabel("x_1")
plot.ylabel("x_2")
plot.legend(['Actual Y = -1', 'Actual Y = +1', 'Predicted Y = -1', 'Predicted Y = +1'], title="Legends", loc = 'lower right', bbox_to_anchor=(1.4, 0.6))
plot.show()
```

Output:

Name: Karan Dua
Student Id: 21331391

Course Code: CS7CS4-202223 MACHINE LEARNING



To compare the performance, I used below code:

```
true_positive = 0
true_negative = 0
false_positive = 0
false_negative = 0
for i in range(len(y)) :
    if y[i] == 1 and baseline_model_prediction[i] == 1 :
        true_positive += 1
    elif y[i] == -1 and baseline_model_prediction[i] == -1 :
        true_negative += 1
    elif y[i] == 1 and baseline_model_prediction[i] == -1 :
        false_negative += 1
    elif y[i] == -1 and baseline_model_prediction[i] == 1 :
        false_positive += 1

Accuracy = ((true_positive + true_negative) / (true_positive + true_negative + false_positive + false_negative)) * 100

y_pos_1_correctly_predicted = (true_positive/y_postive)*100

y_neg_1_correctly_predicted = (true_negative/y_negative)*100

actual_predicted_df = pd.DataFrame({"True Positive":[true_positive], "True Negative":[true_negative], "False Positive":[false_positive], "False Negative":[false_negative], "Accuracy":[Accuracy], "Label Y = 1 Accuracy":[y_pos_1_correctly_predicted], "Label Y = -1 Accuracy":[y_neg_1_correctly_predicted]})
display(actual_predicted_df)
```

Output:

	True Positive	True Negative	False Positive	False Negative	Accuracy	Label Y = 1 Accuracy	Label Y = -1 Accuracy
0	675	0	324	0	67.567568	100.0	0.0

As expected, the baseline model has incorrectly predicted all the y=-1 label and correctly predicted all y=1 label. The baseline model has predicted correctly 67.56% of the labels which is significantly higher than the expected value of 50%. This might cater well for this train dataset but model will face challenges to predict correct labels on test data as we have seen that train data is biased for label y=1. Furthermore, our model has predicted more labels correctly (97%) compared to baseline model hence I can conclude that we can accept out model.

Question (c)(iv): Quadratic equation

To find decision boundary, I have used the same concept as Question (i). We have the following model equation: **$0.600*x1 - 23.691*x2 + 25.257*x1^2 - 0.330*x2^2 - 0.243 = y$**

Now, for logistic regression Probability of event = $1 / (1 + e^{(m1*x1 + m2*x2 + m3*x1^2 + m4*x2^2 + c)})$

Therefore, for decision boundary, Probability of event = $1 / (1 + e^{(m1*x1 + m2*x2 + m3*x1^2 + m4*x2^2 + c)}) = ½$, Solving this equation, we get:

$e^{(m1*x1 + m2*x2 + m3*x1^2 + m4*x2^2 + c)} = 1$, therefore $m1*x1 + m2*x2 + m3*x1^2 + m4*x2^2 + c = 0$

This means that for each value of x1, I have to find value of x2 for which $m1*x1 + m2*x2 + m3*x1^2 + m4*x2^2 + c = 0$. Now, to get decision boundary we can get the following equation:

$m2*x2 + m4*x2^2 + m1*x1 + m3*x1^2 + c = 0$

Again, for each value of x1, we have to find x2 which satisfies the above condition.

For each value of x1, $m1*x1 + m3*x1^2 + c$ can be considered as constant -> x1_cons

Therefore, for given x1, we have to solve the quadratic equation to find x2 lying on decision boundary as:

$m4*x2^2 + m2*x2 + x1\_cons = 0$
Roots of this quadratic equation are: $(-m2 + (m2^2 - 4 * m4 * x1\_cons)^{1/2})/ 2 * m4$ and $(-m2 - (m2^2 - 4 * m4 * x1\_cons)^{1/2})/ 2 * m4$

To find x1_cons, I have used below code:

```
x1_cons = []
for i in range(len(x1)) :
   x1_cons.append(m1*x1[i] + m3*x1_2[i] + c)
```

To calculate roots of the quadratic equation I have used below code:

```
dcsn_bndry = []
for i in range(len(x1_cons)) :
   numerator = (m2 ** 2) - (4 * m4 * x1_cons[i])
   root1 = ( -m2 + math.sqrt(numerator) ) / (2 * m4)
   root2 = ( -m2 - math.sqrt(numerator) ) / (2 * m4)
   if -1 <= root1 <= 1 :
      dcsn_bndry.append(root1)
   else :
      dcsn_bndry.append(root2)
```

To plot the above decision boundary I used below code:

```
plot.scatter(x1[y == -1], x2[y == -1], color='red', marker="o", edgecolors="black")
plot.scatter(x1[y == 1], x2[y == 1], color='yellow', marker="o", edgecolors="black")
plot.scatter(x1[prdctns == -1], x2[prdctns == -1], color='blue', marker="+", alpha=0.85)
plot.scatter(x1[prdctns == 1], x2[prdctns == 1], color='green', marker="+", alpha=0.85)
plot.scatter(x1, dcsn_bndry, marker = "s", color = "black")
plot.title('Data Plot for Actual VS Predicted value of y for Features x1 and x2 for Logistic Regression')
plot.xlabel("x_1")
plot.ylabel("x_2")
plot.legend(['Actual Y = -1', 'Actual Y = +1', 'Predicted Y = -1', 'Predicted Y = +1', 'Decision Boundary'], title="Legends", loc = 'lower right', bbox_to_anchor=(1.41, 0.548))
plot.show()
```

Output: