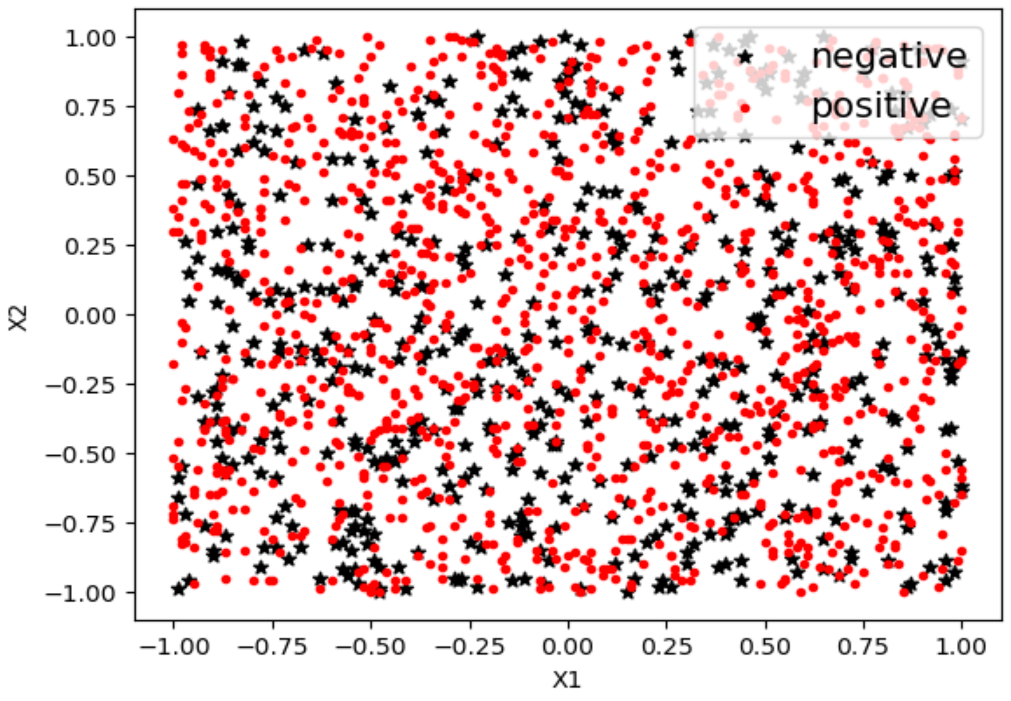
CS7CS4(# id:13-13--13-1) Machine Learning Assignment 3 Haneesh Solasa (22303392)

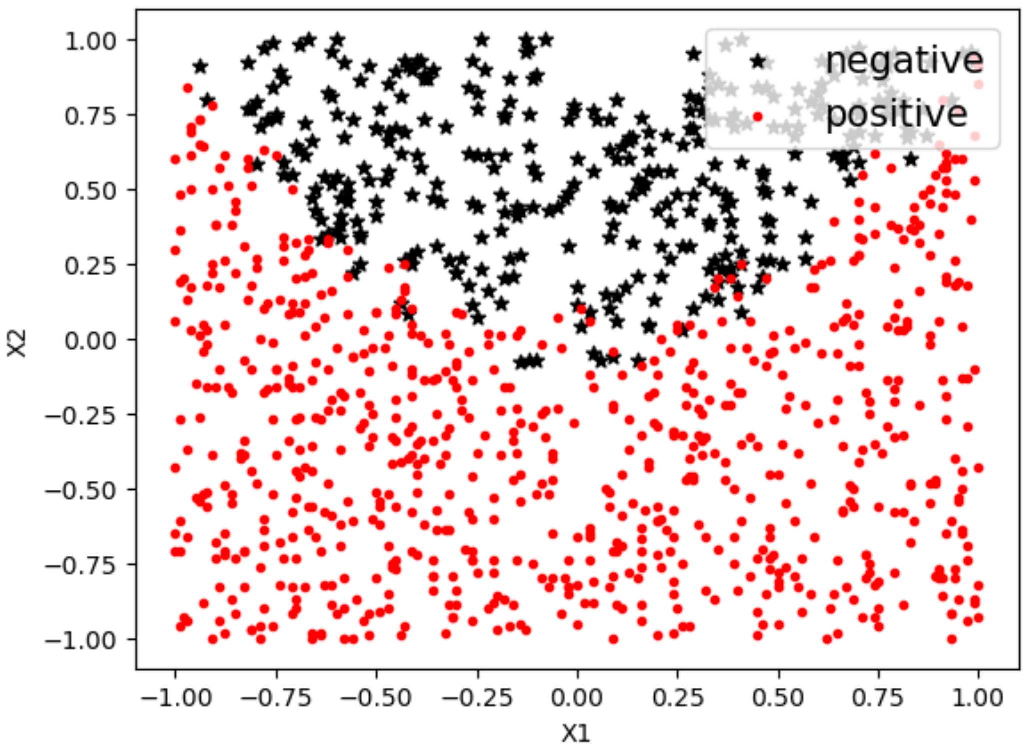
<u>Question i:</u>

The given dataset was split into two csv files so that it would be easier to work with. The scatter plots for both datasets are given below.

<u>Dataset 1</u>



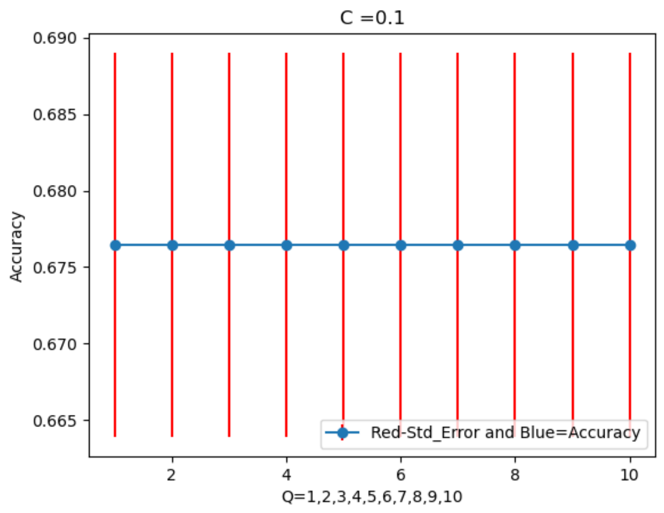
<u>Dataset 2</u>

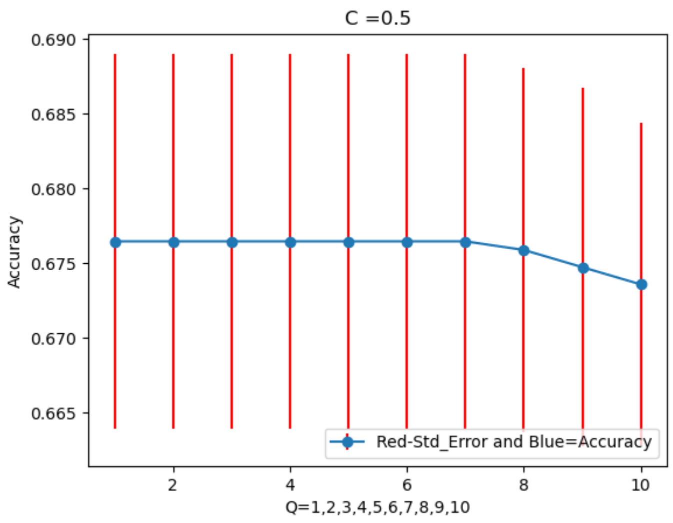


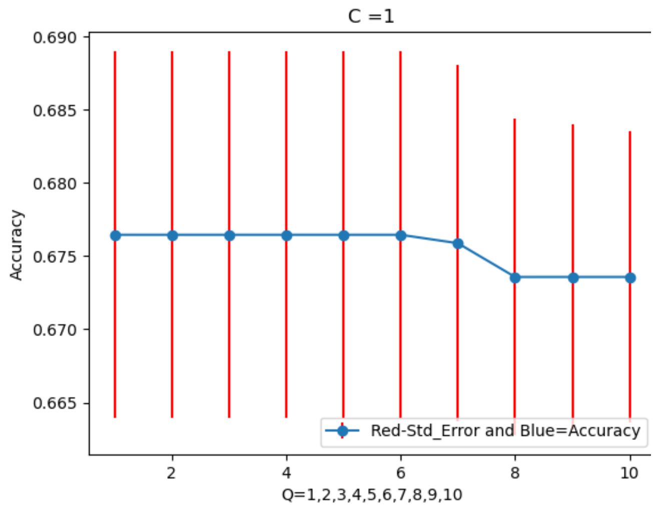
Dataset 1:

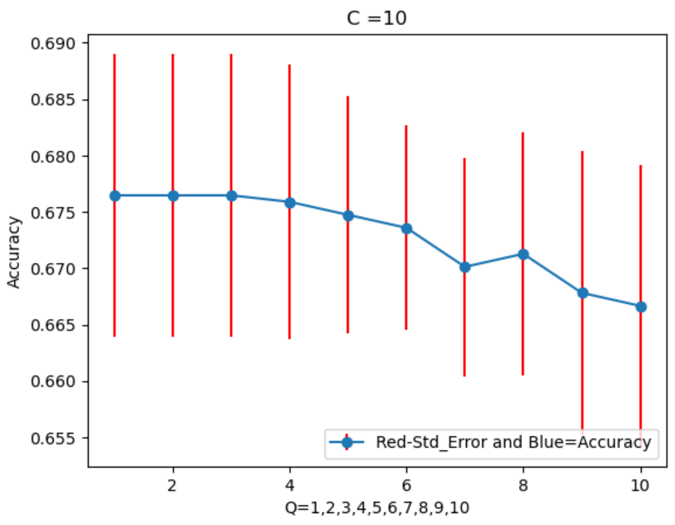
(a) Answer:

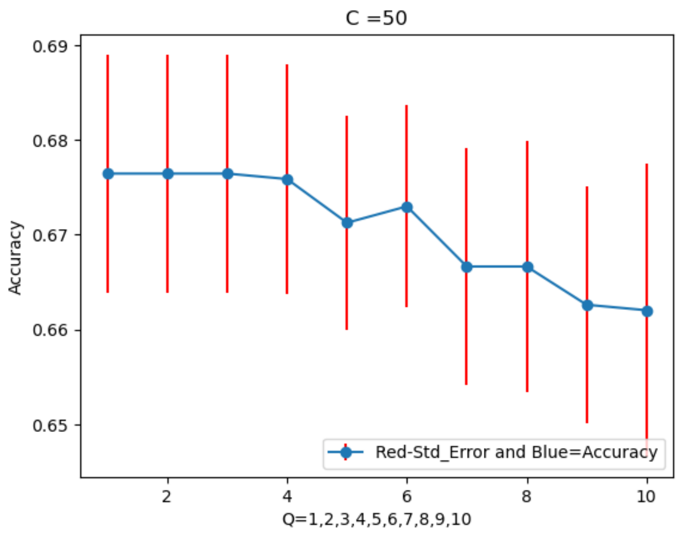
To choose the best order of polynomial features(Q) and the C value, I iterated through a list of different values of C and Q. Then using K-fold cross-validation (where k=5), a logistic regression model is trained with the number of obtained features based on the value of q. The accuracy is then calculated using accuracy_score in each step of the cross-validation and the mean, and the deviation of the error is calculated for each value of C and Q and the error bar is plotted using those values.

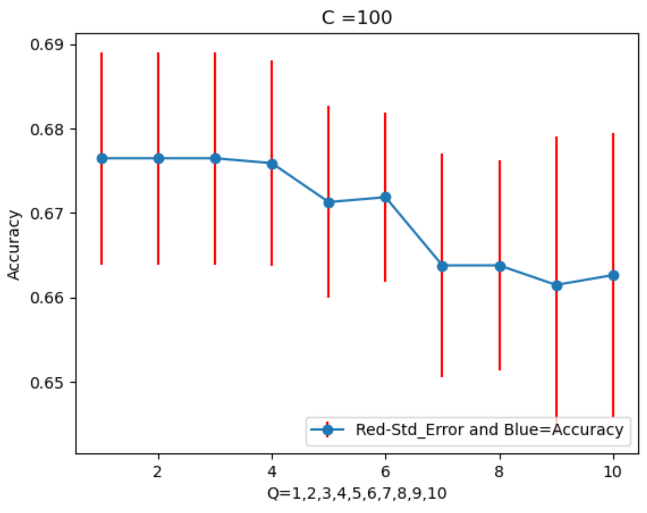












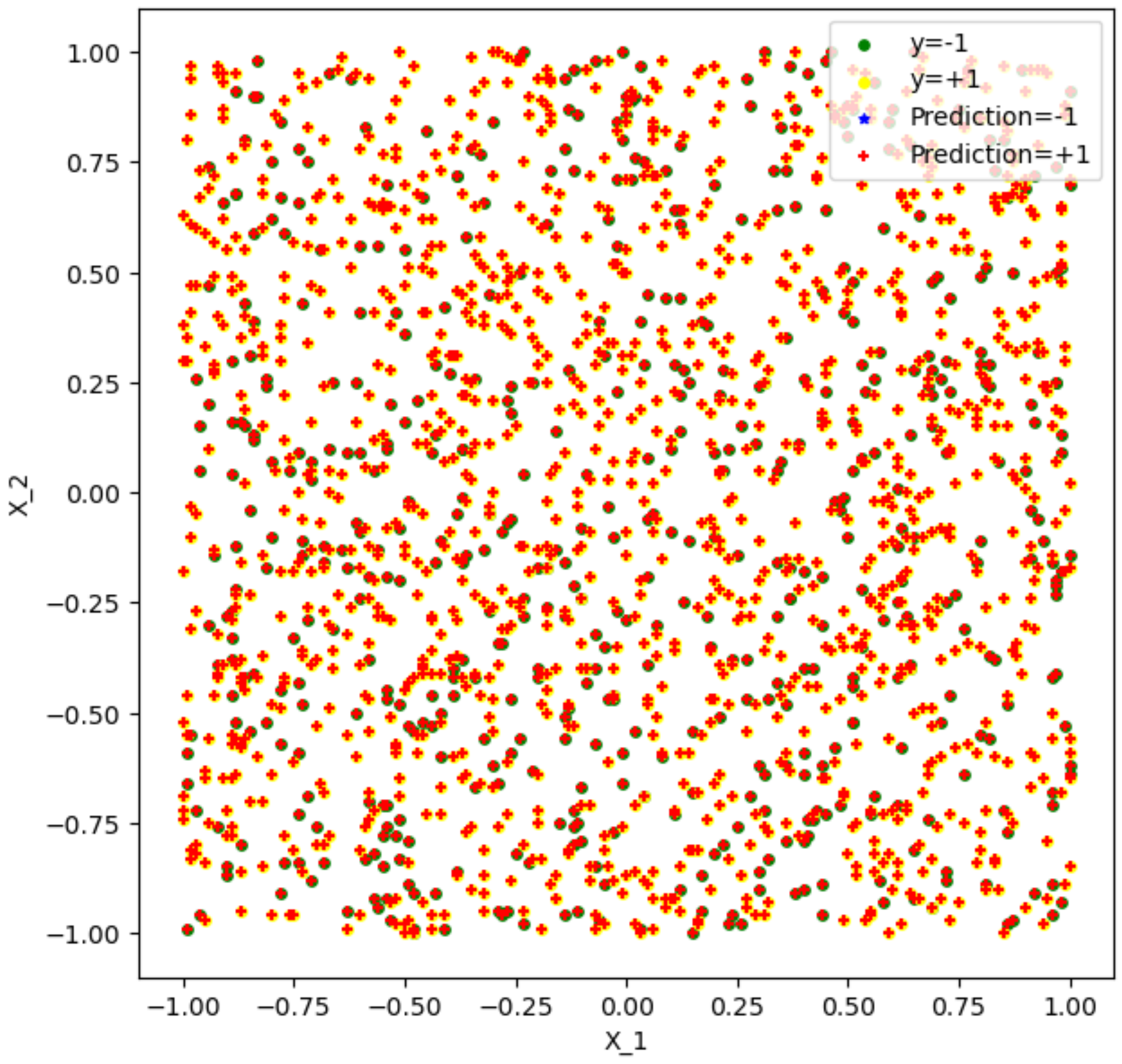
Since the dataset was not linear and has a lot of noise, the change in the value of C and Q did not change a lot in the predictions as given in the below table. The model was unable to predict the negative points at all. Below are the values of accuracy for each value of C & Q.

C: 0.1 Poly: 1=> Accuracy:0.68 (+/- 0.01) C: 0.1 Poly: 2=> Accuracy:0.68 (+/- 0.01) C: 0.1 Poly: 3=> Accuracy:0.68 (+/- 0.01) C: 0.1 Poly: 4=> Accuracy:0.68 (+/- 0.01) C: 0.1 Poly: 5=> Accuracy:0.68 (+/- 0.01) C: 0.1 Poly: 6=> Accuracy:0.68 (+/- 0.01) C: 0.1 Poly: 7=> Accuracy:0.68 (+/- 0.01) C: 0.1 Poly: 8=> Accuracy:0.68 (+/- 0.01) C: 0.1 Poly: 9=> Accuracy:0.68 (+/- 0.01) C: 0.1 Poly: 10=> Accuracy:0.68 (+/- 0.01)	**C: 0.5** Poly: 1=> Accuracy:0.68 (+/- 0.01) C: 0.5 Poly: 2=> Accuracy:0.68 (+/- 0.01) C: 0.5 Poly: 3=> Accuracy:0.68 (+/- 0.01) C: 0.5 Poly: 4=> Accuracy:0.68 (+/- 0.01) C: 0.5 Poly: 5=> Accuracy:0.68 (+/- 0.01) C: 0.5 Poly: 6=> Accuracy:0.68 (+/- 0.01) C: 0.5 Poly: 7=> Accuracy:0.68 (+/- 0.01) C: 0.5 Poly: 8=> Accuracy:0.68 (+/- 0.01) C: 0.5 Poly: 9=> Accuracy:0.67 (+/- 0.01) C: 0.5 Poly: 10=> Accuracy:0.67 (+/- 0.01)	**C: 1** Poly: 1=> Accuracy:0.68 (+/- 0.01) **C: 1 Poly: 2=> Accuracy:0.68 (+/- 0.01)** C: 1 Poly: 3=> Accuracy:0.68 (+/- 0.01) C: 1 Poly: 4=> Accuracy:0.68 (+/- 0.01) C: 1 Poly: 5=> Accuracy:0.68 (+/- 0.01) C: 1 Poly: 6=> Accuracy:0.68 (+/- 0.01) C: 1 Poly: 7=> Accuracy:0.68 (+/- 0.01) C: 1 Poly: 8=> Accuracy:0.67 (+/- 0.01) C: 1 Poly: 9=> Accuracy:0.67 (+/- 0.01) C: 1 Poly: 10=> Accuracy:0.67 (+/- 0.01)
C: 10 Poly: 1=> Accuracy:0.68 (+/- 0.01) C: 10 Poly: 2=> Accuracy:0.68 (+/- 0.01) C: 10 Poly: 3=> Accuracy:0.68 (+/- 0.01) C: 10 Poly: 4=> Accuracy:0.68 (+/- 0.01) C: 10 Poly: 5=> Accuracy:0.67 (+/- 0.01) C: 10 Poly: 6=> Accuracy:0.67 (+/- 0.01) C: 10 Poly: 7=> Accuracy:0.67 (+/- 0.01) C: 10 Poly: 8=> Accuracy:0.67 (+/- 0.01) C: 10 Poly: 9=> Accuracy:0.67 (+/- 0.01) C: 10 Poly: 10=> Accuracy:0.67 (+/- 0.01)	**C: 50** Poly: 1=> Accuracy:0.68 (+/- 0.01) C: 50 Poly: 2=> Accuracy:0.68 (+/- 0.01) C: 50 Poly: 3=> Accuracy:0.68 (+/- 0.01) C: 50 Poly: 4=> Accuracy:0.68 (+/- 0.01) C: 50 Poly: 5=> Accuracy:0.67 (+/- 0.01) C: 50 Poly: 6=> Accuracy:0.67 (+/- 0.01) C: 50 Poly: 7=> Accuracy:0.67 (+/- 0.01) C: 50 Poly: 8=> Accuracy:0.67 (+/- 0.01) C: 50 Poly: 9=> Accuracy:0.66 (+/- 0.01) C: 50 Poly: 10=> Accuracy:0.66 (+/- 0.02)	**C: 100** Poly: 1=> Accuracy:0.68 (+/- 0.01) C: 100 Poly: 2=> Accuracy:0.68 (+/- 0.01) C: 100 Poly: 3=> Accuracy:0.68 (+/- 0.01) C: 100 Poly: 4=> Accuracy:0.68 (+/- 0.01) C: 100 Poly: 5=> Accuracy:0.67 (+/- 0.01) C: 100 Poly: 6=> Accuracy:0.67 (+/- 0.01) C: 100 Poly: 7=> Accuracy:0.66 (+/- 0.01) C: 100 Poly: 8=> Accuracy:0.66 (+/- 0.01) C: 100 Poly: 9=> Accuracy:0.66 (+/- 0.02) C: 100 Poly: 10=> Accuracy:0.66 (+/- 0.02)

Since there is not much of a change in the value of accuracy with the increasing number of features, it's better to use a simpler model with polynomial features 1 or 2 to avoid overfitting. I am taking 2 in this case. Since there is no change in the accuracy with the varying values of C, to avoid underfitting a model, C should not be very small as the penalty will be big and to avoid overfitting, I am taking the value of C to be 1. Highlighted the value.

The scatter plot of the above Logistic Regression Model with Q = 2 and C = 1 is given below. You can see that the model is doing a very bad job doing the classification of -1.

Logistic Regression Model Predictions & Training Data with Q=2 and C=1

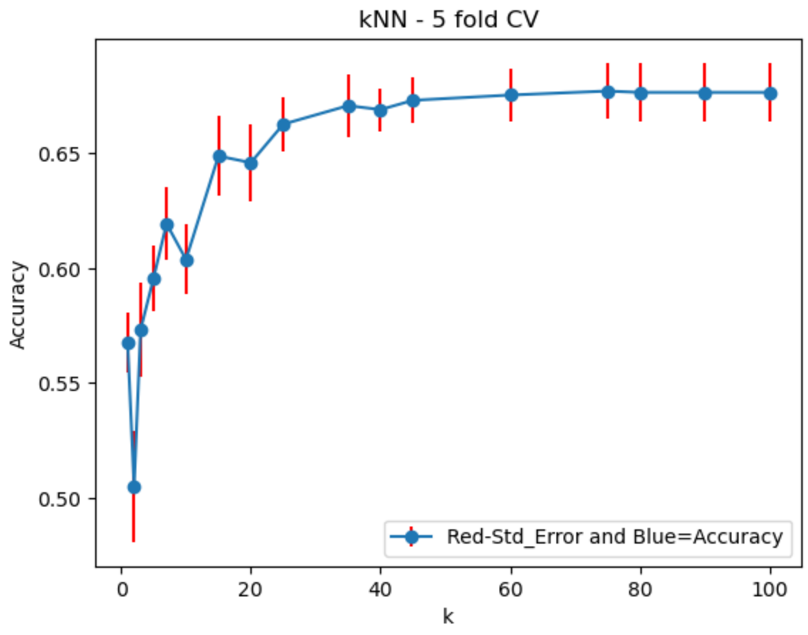


(b) Answer:

To choose the best value of K (number of neighbors), I iterated through different values of k just like what we did for the above model. Then using K-fold cross-validation (where k=5), a KNN model is trained based on the value of K. The accuracy is then calculated using accuracy_score in each step of the cross-validation and the mean, and the deviation of the error is calculated for each value of K and the error bar is plotted using those values.

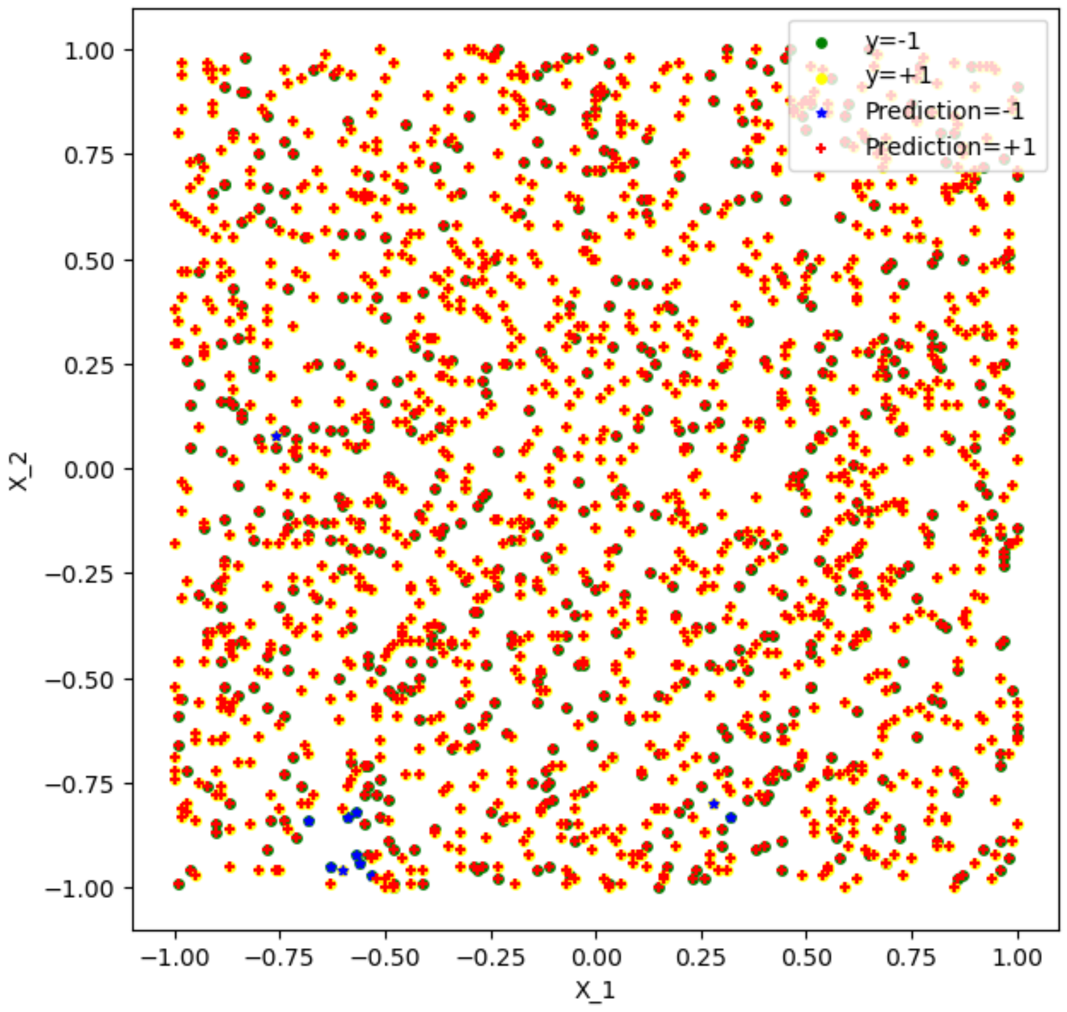
The kNN model could not fit the data properly because of the noise and irregularity of the data. The accuracy for the model went up to a certain level and then it remained constant with the increasing value of K. The values of the accuracy for each K value and the error bar are mentioned below.

```
K: 1=> Accuracy:0.57 (+/- 0.01)
K: 2=> Accuracy:0.51 (+/- 0.02)
K: 3=> Accuracy:0.57 (+/- 0.02)
K: 5=> Accuracy:0.60 (+/- 0.01)
K: 7=> Accuracy:0.62 (+/- 0.02)
K: 10=> Accuracy:0.60 (+/- 0.02)
K: 15=> Accuracy:0.65 (+/- 0.02)
K: 20=> Accuracy:0.65 (+/- 0.02)
K: 25=> Accuracy:0.66 (+/- 0.01)
K: 35=> Accuracy:0.67 (+/- 0.01)
K: 40=> Accuracy:0.67 (+/- 0.01)
K: 45=> Accuracy:0.67 (+/- 0.01)
K: 60=> Accuracy:0.68 (+/- 0.01)
K: 75=> Accuracy:0.68 (+/- 0.01)
K: 80=> Accuracy:0.68 (+/- 0.01)
K: 90=> Accuracy:0.68 (+/- 0.01)
K: 100=> Accuracy:0.68 (+/- 0.01)
```



From the above values of accuracy, we can say that the highest accuracy of the model is at K=60. So, the scatter plot of the prediction data and the training data when K=60 is given below.

kNN Model Training vs Prediction Data with K=60



(c) Answer:

Below are the calculations of Confusion_Matrix along with accuracies. I calculated this using the confusion_matrix library in sklearn. I also compared the matrix of the model with two dummy classifiers. One which predicts the most common class and another which randomly classifies the classes.

Logistic Regression Classifier:

Logistic Regression (Poly=2, C=1)	Predicted Negatives	Predicted Positives
Actual Negatives	0	165
Actual Positives	0	356

Accuracy = 0.68

The above confusion matrix states that, for (y=-1), the model predicted 0 out of 165 negatives and for (y=+1) the model predicted 356 out of 356 as positives. This is with an accuracy of 0.68.

Dummy Classifier (Most Common)	Predicted Negatives	Predicted Positives
Actual Negatives	0	165
Actual Positives	0	356

Accuracy = 0.68

The above confusion matrix states that, for (y=-1), the model predicted 0 out of 165 negatives and for (y=+1) the model predicted 356 out of 356 as positives. This is with an accuracy of 0.68.

Dummy Classifier (Random)	Predicted Negatives	Predicted Positives
Actual Negatives	81	84
Actual Positives	181	175

Accuracy = 0.49

The above confusion matrix states that, for (y=-1), the model predicted 81 out of 165 negatives and the rest of the 84 it predicted as positive and for (y=+1) the model predicted 175 out of 356 as positives and rest 181 as negatives. This is with an accuracy of 0.49.

kNN Classifier:

kNN Classifier (K=60)	Predicted Negatives	Predicted Positives
Actual Negatives	3	162
Actual Positives	15	341

Accuracy = 0.66

The above confusion matrix states that, for (y=-1), the model predicted 3 out of 165 negatives and the rest of the 162 it predicted as positive and for (y=+1) the model predicted 15 out of 356 as positives and rest 341 as negatives. This is with an accuracy of 0.66.

Dummy Classifier (Most Common)	Predicted Negatives	Predicted Positives
Actual Negatives	0	165
Actual Positives	0	356

Accuracy = 0.68

The above confusion matrix states that, for (y=-1), the model predicted 0 out of 165 negatives and for (y=+1) the model predicted 356 out of 356 as positives. This is with an accuracy of 0.68.

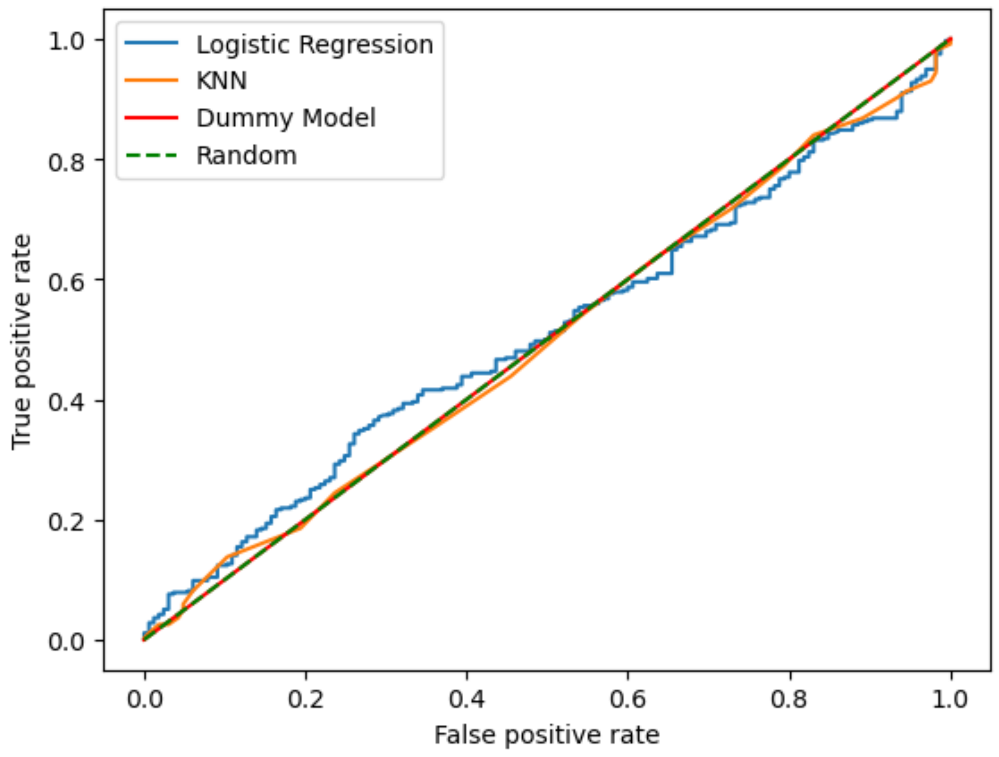
Dummy Classifier (Random)	Predicted Negatives	Predicted Positives
Actual Negatives	85	80
Actual Positives	183	173

Accuracy = 0.50

The above confusion matrix states that, for (y=-1), the model predicted 85 out of 165 negatives and the rest of the 80 it predicted as positive and for (y=+1) the model predicted 173 out of 356 as positives and rest 183 as negatives. This is with an accuracy of 0.50.

(d) Answer:

Below is the ROC Curve for Logistic Regression, KNN and Baseline Predictors.



This plot is plotted using roc_curve and it is between: Logistic Regression with Q=2 and C=1, kNN with K=60, Dummy Classifier which predicts most common class and a dummy classifier which predicts a random value every time.

(e) Answer:

Both models, kNN and the Logistic Models perform close to the Dummy Classifier which predicts the most common class every time. Varying the values of the parameters do not help that much as we are using the best parameter and it is close to the dummy classifier. You can see the accuracy, True Positive Rate, False Positive rate below for the Logistic, Dummy and kNN Models.

Logistic Regression with Q=2 and C=1:

Accuracy => (TN+TP)/(TN+TP+FN+FP) => 356/521 => 68.33
True Positive Rate => TP/(TP+FN) => 356/356 => 1
False Positive Rate => FP/(TN+FP) => 165/165 => 1
Precision => TP/(TP+FP) => 356/521 => 68.33

Dummy Classifier ("Most Common"):

Accuracy => (TN+TP)/(TN+TP+FN+FP) => 356/521 => 68.33
True Positive Rate => TP/(TP+FN) => 356/356 => 1
False Positive Rate => FP/(TN+FP) => 165/165 => 1
Precision => TP/(TP+FP) => 356/521 => 68.33

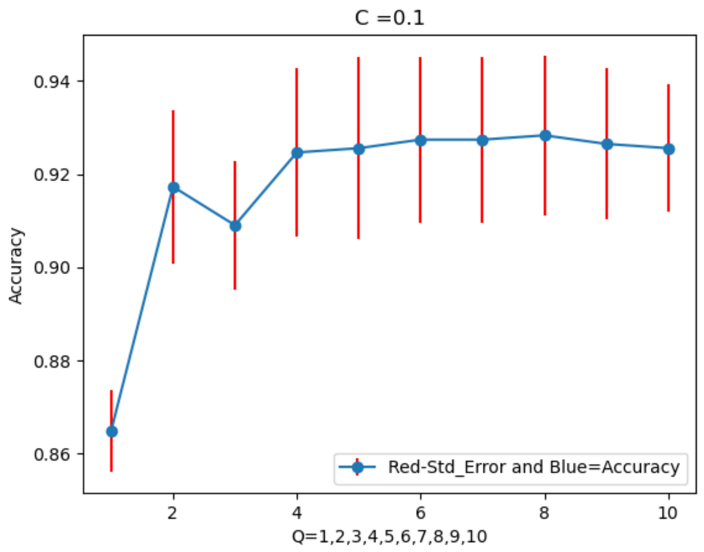
kNN Classifier:

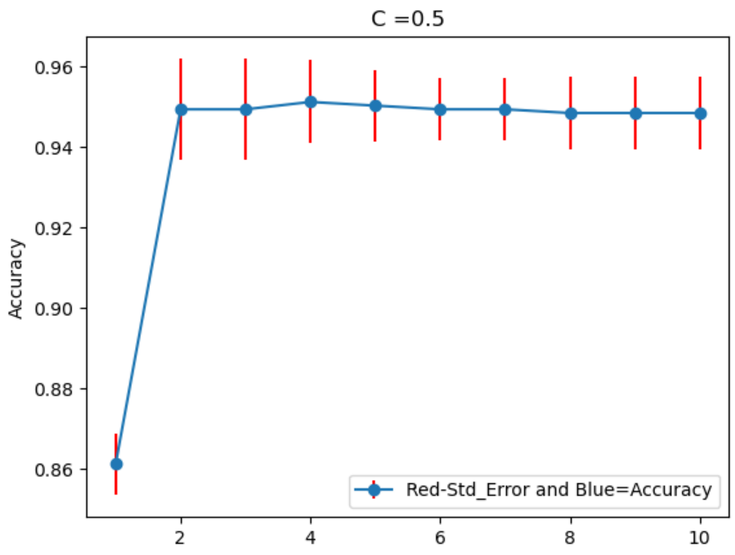
Accuracy => (TN+TP)/(TN+TP+FN+FP) => (3+341)/521 => 66.02
True Positive Rate => TP/(TP+FN) => 341/356 => 95.78
False Positive Rate => FP/(TN+FP) => 162/165 => 98.18
Precision => TP/(TP+FP) => 356/521 => 67.79

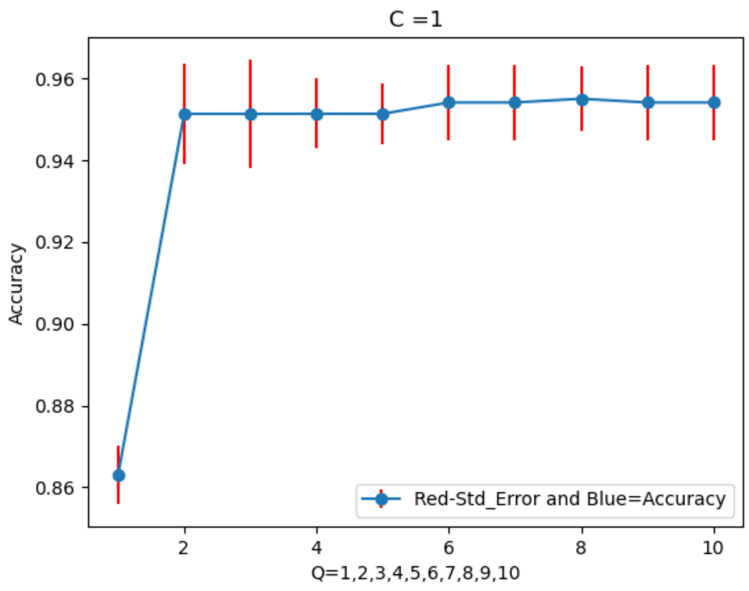
I would choose kNN out of these two because it is good in predicting nonlinear boundaries without increasing the feature count.

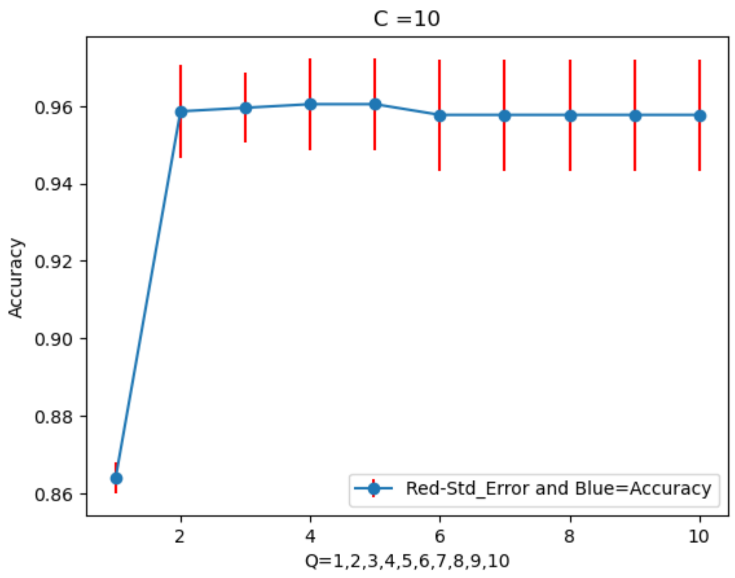
<u>Dataset (2):</u>

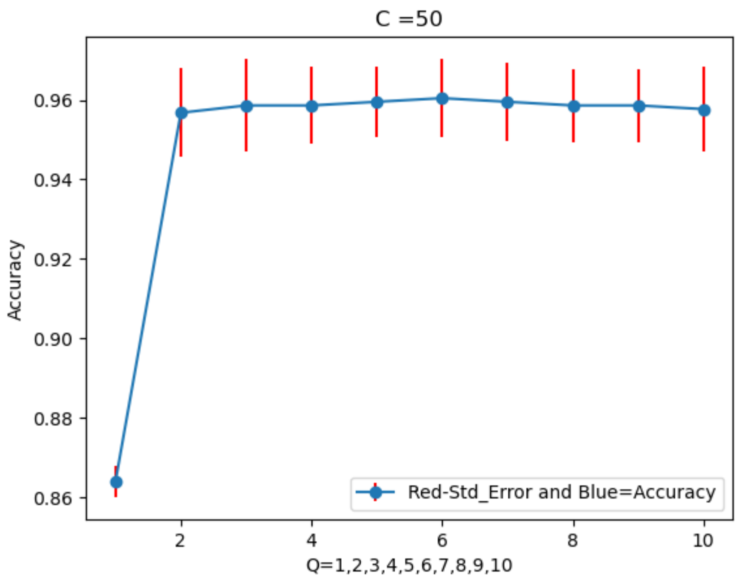
Using the similar approach as above, below are the error bars with a varying value of Q and C values.

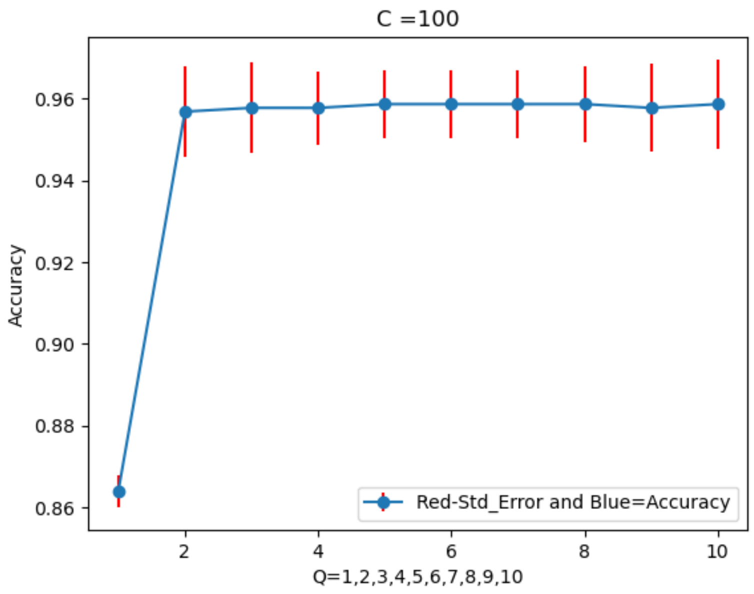












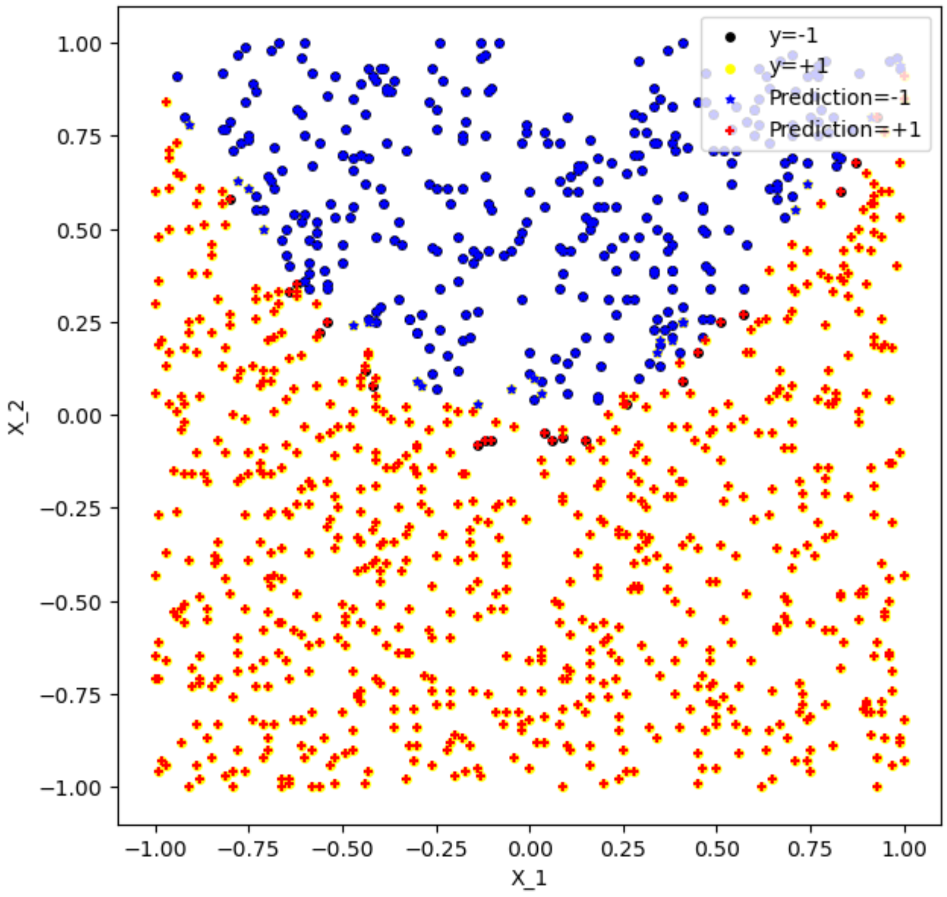
With the dataset being a lot better and having a lot lesser noise than that of the previous one, we have better accuracies for logistic regression. Below are the mean of accuracies for the variying C and Q values.

C: 0.1 Poly: 1=> Accuracy:0.86 (+/- 0.01) C: 0.1 Poly: 2=> Accuracy:0.92 (+/- 0.02) C: 0.1 Poly: 3=> Accuracy:0.91 (+/- 0.01) C: 0.1 Poly: 4=> Accuracy:0.92 (+/- 0.02) C: 0.1 Poly: 5=> Accuracy:0.93 (+/- 0.02) C: 0.1 Poly: 6=> Accuracy:0.93 (+/- 0.02) C: 0.1 Poly: 7=> Accuracy:0.93 (+/- 0.02) C: 0.1 Poly: 8=> Accuracy:0.93 (+/- 0.02) C: 0.1 Poly: 9=> Accuracy:0.93 (+/- 0.02) C: 0.1 Poly: 10=> Accuracy:0.93 (+/- 0.01)	**C: 0.5** Poly: 1=> Accuracy:0.86 (+/- 0.01) C: 0.5 Poly: 2=> Accuracy:0.95 (+/- 0.01) C: 0.5 Poly: 3=> Accuracy:0.95 (+/- 0.01) C: 0.5 Poly: 4=> Accuracy:0.95 (+/- 0.01) C: 0.5 Poly: 5=> Accuracy:0.95 (+/- 0.01) C: 0.5 Poly: 6=> Accuracy:0.95 (+/- 0.01) C: 0.5 Poly: 7=> Accuracy:0.95 (+/- 0.01) C: 0.5 Poly: 8=> Accuracy:0.95 (+/- 0.01) C: 0.5 Poly: 9=> Accuracy:0.95 (+/- 0.01) C: 0.5 Poly: 10=> Accuracy:0.95 (+/- 0.01)	**C: 1** Poly: 1=> Accuracy:0.86 (+/- 0.01) C: 1 Poly: 2=> Accuracy:0.95 (+/- 0.01) C: 1 Poly: 3=> Accuracy:0.95 (+/- 0.01) C: 1 Poly: 4=> Accuracy:0.95 (+/- 0.01) C: 1 Poly: 5=> Accuracy:0.95 (+/- 0.01) C: 1 Poly: 6=> Accuracy:0.95 (+/- 0.01) C: 1 Poly: 7=> Accuracy:0.95 (+/- 0.01) C: 1 Poly: 8=> Accuracy:0.95 (+/- 0.01) C: 1 Poly: 9=> Accuracy:0.95 (+/- 0.01) C: 1 Poly: 10=> Accuracy:0.95 (+/- 0.01)
C: 10 Poly: 1=> Accuracy:0.86 (+/- 0.00) **C: 10 Poly: 2=> Accuracy:0.96 (+/- 0.01)** C: 10 Poly: 3=> Accuracy:0.96 (+/- 0.01) C: 10 Poly: 4=> Accuracy:0.96 (+/- 0.01) C: 10 Poly: 5=> Accuracy:0.96 (+/- 0.01) C: 10 Poly: 6=> Accuracy:0.96 (+/- 0.01) C: 10 Poly: 7=> Accuracy:0.96 (+/- 0.01) C: 10 Poly: 8=> Accuracy:0.96 (+/- 0.01) C: 10 Poly: 9=> Accuracy:0.96 (+/- 0.01) C: 10 Poly: 10=> Accuracy:0.96 (+/- 0.01)	**C: 50** Poly: 1=> Accuracy:0.86 (+/- 0.00) C: 50 Poly: 2=> Accuracy:0.96 (+/- 0.01) C: 50 Poly: 3=> Accuracy:0.96 (+/- 0.01) C: 50 Poly: 4=> Accuracy:0.96 (+/- 0.01) C: 50 Poly: 5=> Accuracy:0.96 (+/- 0.01) C: 50 Poly: 6=> Accuracy:0.96 (+/- 0.01) C: 50 Poly: 7=> Accuracy:0.96 (+/- 0.01) C: 50 Poly: 8=> Accuracy:0.96 (+/- 0.01) C: 50 Poly: 9=> Accuracy:0.96 (+/- 0.01) C: 50 Poly: 10=> Accuracy:0.96 (+/- 0.01)	**C: 100** Poly: 1=> Accuracy:0.86 (+/- 0.00) C: 100 Poly: 2=> Accuracy:0.96 (+/- 0.01) C: 100 Poly: 3=> Accuracy:0.96 (+/- 0.01) C: 100 Poly: 4=> Accuracy:0.96 (+/- 0.01) C: 100 Poly: 5=> Accuracy:0.96 (+/- 0.01) C: 100 Poly: 6=> Accuracy:0.96 (+/- 0.01) C: 100 Poly: 7=> Accuracy:0.96 (+/- 0.01) C: 100 Poly: 8=> Accuracy:0.96 (+/- 0.01) C: 100 Poly: 9=> Accuracy:0.96 (+/- 0.01) C: 100 Poly: 10=> Accuracy:0.96 (+/- 0.01)

If we look at the mean of accuracies, 0.96 is the best accuracy that we can get. This can be achieved if Q>=2 and C>=10. If the accuracy is similar, it is better to take a smaller model to avoid overfitting, so we take Q=2. To avoid overfitting, we should not take C value to be very high. So, we can take C value to be 10.

The scatter plot of the model with Q=2 and C=10 is below.

Logistic Regression Predictions and training data with Q=2 and C=10

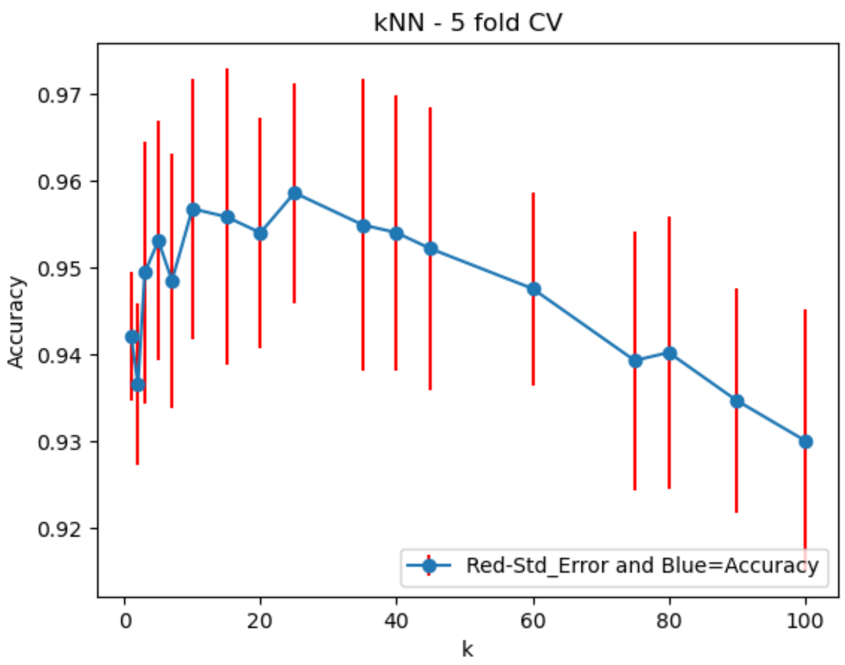


(b) Answer:

Using the above approach, I plotted mean of accuracies for varying values of K for kNN Model.

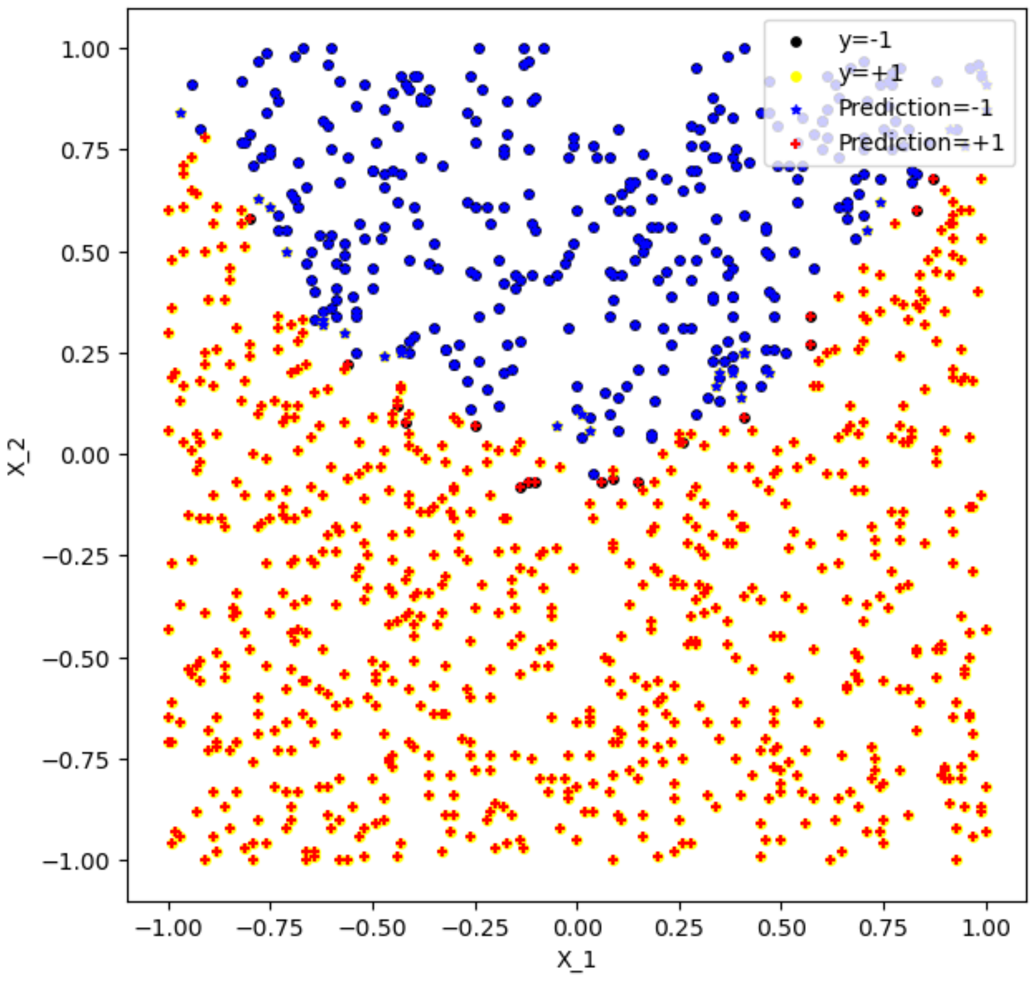
Below are the accuracies and the error bar plots.

K: 1=> Accuracy:0.94 (+/- 0.01)
K: 2=> Accuracy:0.94 (+/- 0.01)
K: 3=> Accuracy:0.95 (+/- 0.02)
K: 5=> Accuracy:0.95 (+/- 0.01)
K: 7=> Accuracy:0.95 (+/- 0.01)
K: 10=> Accuracy:0.96 (+/- 0.01)
K: 15=> Accuracy:0.96 (+/- 0.02)
K: 20=> Accuracy:0.95 (+/- 0.01)
K: 25=> Accuracy:0.96 (+/- 0.01)
K: 35=> Accuracy:0.95 (+/- 0.02)
K: 40=> Accuracy:0.95 (+/- 0.02)
K: 45=> Accuracy:0.95 (+/- 0.02)
K: 60=> Accuracy:0.95 (+/- 0.01)
K: 75=> Accuracy:0.94 (+/- 0.01)
K: 80=> Accuracy:0.94 (+/- 0.02)
K: 90=> Accuracy:0.93 (+/- 0.01)
K: 100=> Accuracy:0.93 (+/- 0.02)



If we see the above error bar and the accuracies, we can see that for increasing values of K, the accuracy is increasing up to a certain point and then starts decreasing. That is when the model is overfitting. So, we can take the K value to be 25. Below is the scatter plot for K value 25.

kNN Model predictions and training data when K=25



(c) Answer:

Below are the confusion matrices created using the above approach for Logistic Regression, kNN, Dummy Classifier which predicts common class and one which predicts random class.

Logistic Regression Confusion Matrix:

Logistic Regression (Poly=2, C=10)	Predicted Negatives	Predicted Positives
Actual Negatives	99	7
Actual Positives	7	214

Accuracy = 0.96

The above confusion matrix states that, for (y=-1), the model predicted 99 out of 106 negatives and the rest of the 7 it predicted as positive and for (y=+1) the model predicted 214 out of 221 as positives and rest 7 as negatives. This is with an accuracy of 0.96.

Dummy Classifier (Most Common)	Predicted Negatives	Predicted Positives
Actual Negatives	0	106
Actual Positives	0	221

Accuracy = 0.68

The above confusion matrix states that, for (y=-1), the model predicted 0 out of 106 negatives and for (y=+1) the model predicted 221 out of 221 as positives. This is with an accuracy of 0.68.

Dummy Classifier (Random)	Predicted Negatives	Predicted Positives
Actual Negatives	61	45
Actual Positives	119	102

Accuracy = 0.50

The above confusion matrix states that, for (y=-1), the model predicted 61 out of 106 negatives and the rest of the 45 it predicted as positive and for (y=+1) the model predicted 102 out of 221 as positives and rest 119 as negatives. This is with an accuracy of 0.50.

kNN Classifier:

kNN Classifier (K=25)	Predicted Negatives	Predicted Positives
Actual Negatives	99	7
Actual Positives	9	212

Accuracy = 0.95

The above confusion matrix states that, for (y=-1), the model predicted 99 out of 106 negatives and the rest of the 7 it predicted as positive and for (y=+1) the model predicted 212 out of 221 as positives and rest 9 as negatives. This is with an accuracy of 0.95.

Dummy Classifier (Most Common)	Predicted Negatives	Predicted Positives
Actual Negatives	0	106
Actual Positives	0	221

Accuracy = 0.68

The above confusion matrix states that, for (y=-1), the model predicted 0 out of 106 negatives and for (y=+1) the model predicted 221 out of 221 as positives. This is with an accuracy of 0.68.

Dummy Classifier (Random)	Predicted Negatives	Predicted Positives
Actual Negatives	53	53
Actual Positives	121	100

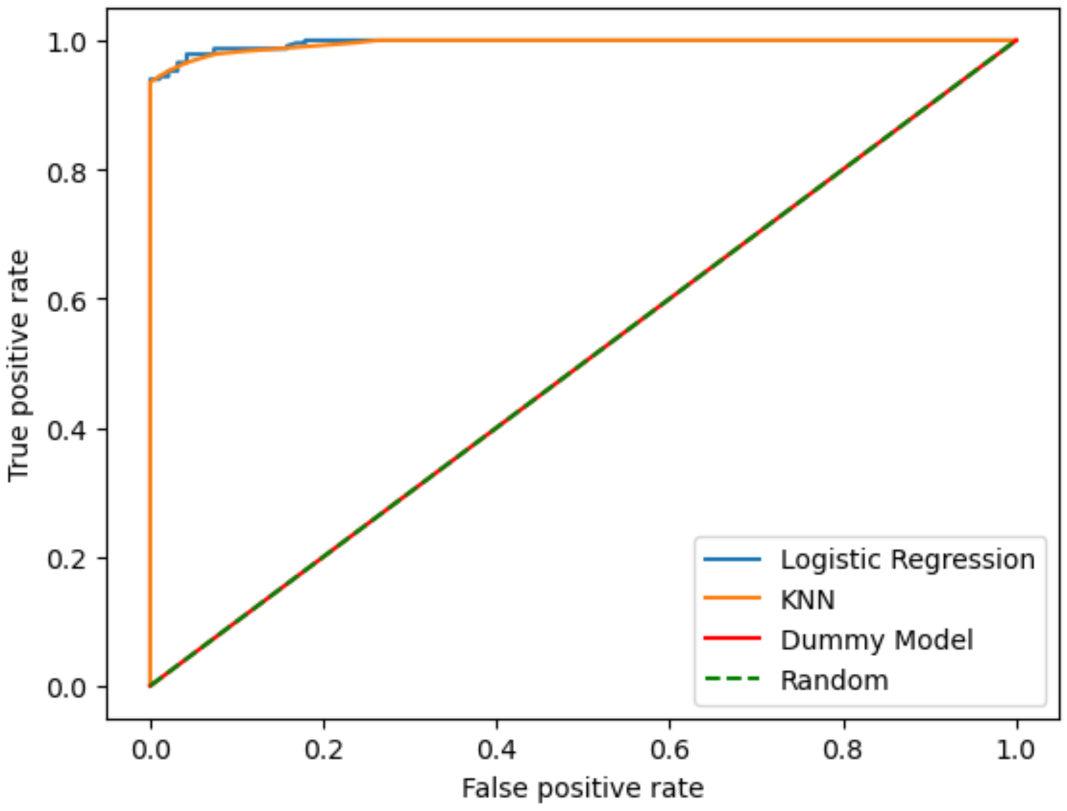
Accuracy = 0.47

The above confusion matrix states that, for (y=-1), the model predicted 53 out of 106 negatives and the rest of the 53 it predicted as positive and for (y=+1) the model predicted 100 out of 221 as positives and rest 121 as negatives. This is with an accuracy of 0.47.

(d) Answer:

Using the approaches as above, I plotted the ROC curve for the Logistic Regression, kNN Model, Dummy Classifier which predicts Common Class and the Dummy Classifier which predicts random class. Below is the ROC Curve.

ROC Curve



(e) Answer:

Both the models, Logistic regression and the kNN models have very great accuracies. With changing C values and Q values in Logistic Regression and similarly changing K values in kNN, we can get good accuracies. They outperform Dummy/Baseline Predictors. You can see the accuracy, True Positive Rate, False Positive rate below for the Logistic, Dummy and kNN Models.

Logistic Regression with Q=2 and C=10:

Accuracy => (TN+TP)/(TN+TP+FN+FP) => 313/327 => 0.957
True Positive Rate => TP/(TP+FN) => 214/221 => 0.968
False Positive Rate => FP/(TN+FP) => 7/106 => 0.066
Precision => TP/(TP+FP) => 214/221 => 0.968

Dummy Classifier ("Most Common"):

Accuracy => (TN+TP)/(TN+TP+FN+FP) => 221/327=>0.675
True Positive Rate => TP/(TP+FN) => 221/221 => 1
False Positive Rate => FP/(TN+FP) => 106/106 => 1
Precision => TP/(TP+FP) => 221/327 => 0.675

kNN Classifier:

Accuracy => (TN+TP)/(TN+TP+FN+FP) => 311/327=>0.951
True Positive Rate => TP/(TP+FN) => 212/221 => 0.959
False Positive Rate => FP/(TN+FP) => 7/106 => 0.066
Precision => TP/(TP+FP) => 212/219 => 0.968

In this case, I would choose Logistic Regression as the drawing a decision boundary is trivial as the data is spread out. In these situations, Logistic regression does a better job because of the increasing features. When you look at the accuracies, True Positive Rates, False Positive Rates of the above models, you can see that using Logistic Regression seem to be the best option.

<u>Appendix:</u>

Dataset(i):

```
# id:13-13--13-1
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import KFold
from sklearn.linear_model import LogisticRegression
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.metrics import accuracy_score

from sklearn.dummy import DummyClassifier
Dummy_Model_Final = DummyClassifier(strategy="most_frequent")
Random_Model_Final = DummyClassifier(strategy="uniform")

df1 = pd.read_csv("Week 4_1.csv")
```

```
X1 = df1.iloc[:, 0]
X2 = df1.iloc[:, 1]
X = np.column_stack((X1, X2))
y = np.array(df1.iloc[:, 2])

negative=plt.scatter(X1[y<0],X2[y<0],color='black',marker='*',label='negati
ve')
positive=plt.scatter(X1[y>0],X2[y>0],color='red',marker='.',label='positive
')
plt.xlabel("X1")
plt.ylabel("X2")
plt.legend(loc='upper right',fontsize=15)
plt.show()
```

(a):

```
#(a)
Poly = [1,2,3,4,5,6,7,8,9,10]
C=[0.1,0.5,1,10,50,100]
kf = KFold(n_splits=5)
Accuracy = []
temp = []
for j in C:
    mean_poly=[]
    std_poly=[]
    for i in Poly:
        X_poly = PolynomialFeatures(i).fit_transform(X)
        temp = []
        model = LogisticRegression(penalty='l2', C=j, max_iter=10000)
        for train, test in kf.split(X_poly):
            model.fit(X_poly[train], y[train])
            y_pred=model.predict(X_poly[test])
            temp.append(accuracy_score(y[test], y_pred))
        print("C: "+str(j)+" Poly: "+str(i)+ "=> Accuracy:%0.2f (+/-
%0.2f)"%(np.array(temp).mean(),np.array(temp).std()))
        mean_poly.append(np.array(temp).mean())
        std_poly.append(np.array(temp).std())
    plt.errorbar(Poly,mean_poly,yerr=std_poly,ecolor='red',fmt='-o')
    plt.legend(["Red-Std_Error and Blue=Accuracy"],loc=4)
    plt.xlabel("Q=1,2,3,4,5,6,7,8,9,10")
    plt.ylabel('Accuracy')
    plt.title("C ={}".format(j))

    plt.show()

from sklearn.metrics import classification_report,confusion_matrix
X_poly = PolynomialFeatures(2).fit_transform(X)
model = LogisticRegression(C=1,random_state = 1)
model.fit(X_poly ,y)
y_pred = model.predict(X_poly)
%matplotlib inline
```

```
plt.figure(figsize=(7, 7))
points_1=plt.scatter(X1[y<0],X2[y<0],marker="o",color="green",label="y=-
1",s=15)
points_2=plt.scatter(X1[y>0],X2[y>0],marker="o",color="yellow",label="y=+1"
,s=15)
points_3=plt.scatter(X1[y_pred<0],X2[y_pred<0],
marker="*",color="blue",label="Prediction=-1",s=15)
points_4=plt.scatter(X1[y_pred>0],X2[y_pred>0],marker="+",color="red",label
="Prediction=+1",s=15)
plt.xlabel('X_1')
plt.ylabel('X_2')
plt.legend(handles=[points_1,points_2,points_3,points_4],loc='upper right')
confusion_matrix(y, y_pred)
```

(b):

```
#(b)
kf = KFold(n_splits=5)
mean_poly=[]
std_poly=[]
k=[1,2,3,5,7,10,15,20,25,35,40,45,60,75,80,90,100]
from sklearn.neighbors import KNeighborsClassifier
for ki in k:
    model = KNeighborsClassifier(n_neighbors=ki,weights='uniform')
    temp=[]
    for train, test in kf.split(X):
        model.fit(X[train], y[train])
        y_pred=model.predict(X[test])
        temp.append(accuracy_score(y[test], y_pred))
    mean_poly.append(np.array(temp).mean())
    std_poly.append(np.array(temp).std())
    print("K: "+str(ki)+ "=> Accuracy:%0.2f (+/-
%0.2f)"%(np.array(temp).mean(),np.array(temp).std()))
# print(k, mean_poly)
plt.errorbar(k,mean_poly,yerr=std_poly,ecolor='red',fmt='-o')
plt.legend(["Red-Std_Error and Blue=Accuracy"],loc=4)
plt.xlabel('k')
plt.ylabel('Accuracy')
plt.title('kNN - 5 fold CV')
plt.show()

from sklearn.metrics import classification_report,confusion_matrix
k=60
model = KNeighborsClassifier(n_neighbors=k,weights='uniform')
model.fit(X ,y)
y_pred = model.predict(X)
%matplotlib inline
plt.figure(figsize=(7, 7))
points_1=plt.scatter(X1[y<0],X2[y<0],marker="o",color="green",label="y=-
1",s=15)
points_2=plt.scatter(X1[y>0],X2[y>0],marker="o",color="yellow",label="y=+1"
,s=15)
```

```
points_3=plt.scatter(X1[y_pred<0],X2[y_pred<0],
marker="*",color="blue",label="Prediction=-1",s=15)
points_4=plt.scatter(X1[y_pred>0],X2[y_pred>0],marker="+",color="red",label
="Prediction=+1",s=15)
plt.xlabel('X_1')
plt.ylabel('X_2')
plt.legend(handles=[points_1,points_2,points_3,points_4],loc='upper right')
confusion_matrix(y, y_pred)
```

(c):

```
#(c)
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
def Confusion_Matrix(model, X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3,random_state=3)
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)
    conf_matrix = confusion_matrix(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)

    print(conf_matrix)
    print("Accuracy = %0.2f"%accuracy)

X_poly = PolynomialFeatures(2).fit_transform(X)
Logistic_Model_Final=LogisticRegression(C=1)
Confusion_Matrix(Logistic_Model_Final,X_poly,y)
Confusion_Matrix(Dummy_Model_Final,X_poly,y)
Confusion_Matrix(Random_Model_Final,X_poly,y)

k=60
KNN_Model_Final = KNeighborsClassifier(n_neighbors=k,weights='uniform')
Confusion_Matrix(KNN_Model_Final,X,y)
Confusion_Matrix(Dummy_Model_Final,X,y)
Confusion_Matrix(Random_Model_Final,X,y)
```

(d):

```
#(d)
from sklearn.metrics import roc_curve


X_poly = PolynomialFeatures(2).fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_poly, y,
test_size=0.3, random_state=3)
LG_Model = LogisticRegression(C=1)
LG_Model.fit(X_train,y_train)
y_pred1 = LG_Model.predict(X_test)
```

```
false_positive, true_positive, _ =
roc_curve(y_test,LG_Model.decision_function(X_test))
plt.plot(false_positive,true_positive, label='Logistic Regression')


k=60
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=3)
KNN_Model_Final = KNeighborsClassifier(n_neighbors=k,weights='uniform')
KNN_Model_Final.fit(X_train,y_train)
y_pred2=KNN_Model_Final.predict_proba(X_test)
false_positive, true_positive, _ = roc_curve(y_test,y_pred2[:,1])
plt.plot(false_positive,true_positive, label='KNN')

Classify_Dummy = DummyClassifier(strategy="most_frequent")
Classify_Dummy.fit(X_train,y_train)
ypred3 = Classify_Dummy.predict_proba(X_test)
false_positive, true_positive, _ = roc_curve(y_test,ypred3[:,1])
plt.plot(false_positive,true_positive, label='Dummy Model', color ='red')


plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.plot([0, 1], [0, 1], color='green',linestyle='--',label ='Random')
#random classifier
plt.legend()
plt.show()
```

Dataset 2:

```
# id:13-13--13-1
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import KFold
from sklearn.linear_model import LogisticRegression
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score

from sklearn.dummy import DummyClassifier
Dummy_Model_Final = DummyClassifier(strategy="most_frequent")
Random_Model_Final = DummyClassifier(strategy="uniform")

df2= pd.read_csv("Week4_2.csv")

X1 = df2.iloc[:, 0]
X2 = df2.iloc[:, 1]
X = np.column_stack((X1, X2))
y = np.array(df2.iloc[:, 2])
```

(b):

```
#(a)
Poly = [1,2,3,4,5,6,7,8,9,10]
C=[0.1,0.5,1,10,50,100]
kf = KFold(n_splits=5)
Accuracy = []
temp = []
for j in C:
    mean_poly=[]
    std_poly=[]
    for i in Poly:
        X_poly = PolynomialFeatures(i).fit_transform(X)
        temp = []
        model = LogisticRegression(penalty='l2', C=j, max_iter=10000)
        for train, test in kf.split(X_poly):
            model.fit(X_poly[train], y[train])
            y_pred=model.predict(X_poly[test])
            temp.append(accuracy_score(y[test], y_pred))
        print("C: "+str(j)+" Poly: "+str(i)+ "=> Accuracy:%0.2f (+/-
%0.2f)"%(np.array(temp).mean(),np.array(temp).std()))
        mean_poly.append(np.array(temp).mean())
        std_poly.append(np.array(temp).std())
    plt.errorbar(Poly,mean_poly,yerr=std_poly,ecolor='red',fmt='-o')
    plt.legend(["Red-Std_Error and Blue=Accuracy"],loc=4)
    plt.xlabel("Q=1,2,3,4,5,6,7,8,9,10")
    plt.ylabel('Accuracy')
    plt.title("C ={}".format(j))

    plt.show()

from sklearn.metrics import classification_report,confusion_matrix
X_poly = PolynomialFeatures(2).fit_transform(X)
model = LogisticRegression(C=10,random_state = 1)
model.fit(X_poly ,y)
y_pred = model.predict(X_poly)
%matplotlib inline
plt.figure(figsize=(7, 7))
points_1=plt.scatter(X1[y<0],X2[y<0],marker="o",color="black",label="y=-
1",s=15)
points_2=plt.scatter(X1[y>0],X2[y>0],marker="o",color="yellow",label="y=+1"
,s=15)
points_3=plt.scatter(X1[y_pred<0],X2[y_pred<0],
marker="*",color="blue",label="Prediction=-1",s=15)
points_4=plt.scatter(X1[y_pred>0],X2[y_pred>0],marker="+",color="red",label
="Prediction=+1",s=15)
plt.xlabel('X_1')
plt.ylabel('X_2')
plt.legend(handles=[points_1,points_2,points_3,points_4],loc='upper right')
confusion_matrix(y, y_pred)
```

(b):

```
#(b)
kf = KFold(n_splits=5)
mean_poly=[]
std_poly=[]
k=[1,2,3,5,7,10,15,20,25,35,40,45,60,75,80,90,100]
from sklearn.neighbors import KNeighborsClassifier
for ki in k:
    model = KNeighborsClassifier(n_neighbors=ki,weights='uniform')
    temp=[]
    for train, test in kf.split(X):
        model.fit(X[train], y[train])
        y_pred=model.predict(X[test])
        temp.append(accuracy_score(y[test], y_pred))
    mean_poly.append(np.array(temp).mean())
    std_poly.append(np.array(temp).std())
    print("K: "+str(ki)+ "=> Accuracy:%0.2f (+/-
%0.2f)"%(np.array(temp).mean(),np.array(temp).std()))
# print(k, mean_poly)
plt.errorbar(k,mean_poly,yerr=std_poly,ecolor='red',fmt='-o')
plt.legend(["Red-Std_Error and Blue=Accuracy"],loc=4)
plt.xlabel('k')
plt.ylabel('Accuracy')
plt.title('kNN - 5 fold CV')
plt.show()

from sklearn.metrics import classification_report,confusion_matrix
k=25
model = KNeighborsClassifier(n_neighbors=k,weights='uniform')
model.fit(X ,y)
y_pred = model.predict(X)
%matplotlib inline
plt.figure(figsize=(7, 7))
points_1=plt.scatter(X1[y<0],X2[y<0],marker="o",color="black",label="y=-
1",s=15)
points_2=plt.scatter(X1[y>0],X2[y>0],marker="o",color="yellow",label="y=+1"
,s=15)
points_3=plt.scatter(X1[y_pred<0],X2[y_pred<0],
marker="*",color="blue",label="Prediction=-1",s=15)
points_4=plt.scatter(X1[y_pred>0],X2[y_pred>0],marker="+",color="red",label
="Prediction=+1",s=15)
plt.xlabel('X_1')
plt.ylabel('X_2')
plt.legend(handles=[points_1,points_2,points_3,points_4],loc='upper right')
confusion_matrix(y, y_pred)
```

(c):

```
#(c)
from sklearn.model_selection import train_test_split
def Confusion_Matrix(model, X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3,random_state=1)
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)
    conf_matrix = confusion_matrix(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)

    print(conf_matrix)
    print("Accuracy = %0.2f"%accuracy)

X_poly = PolynomialFeatures(2).fit_transform(X)
Logistic_Model_Final=LogisticRegression(C=10)
Confusion_Matrix(Logistic_Model_Final,X_poly,y)
Confusion_Matrix(Dummy_Model_Final,X_poly,y)
Confusion_Matrix(Random_Model_Final,X_poly,y)

k=25
KNN_Model_Final = KNeighborsClassifier(n_neighbors=k,weights='uniform')
Confusion_Matrix(KNN_Model_Final,X,y)
Confusion_Matrix(Dummy_Model_Final,X,y)
Confusion_Matrix(Random_Model_Final,X,y)
```

(d):

```
#(d)
from sklearn.metrics import roc_curve


X_poly = PolynomialFeatures(2).fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_poly, y,
test_size=0.3, random_state=3)
LG_Model = LogisticRegression(C=10)
LG_Model.fit(X_train,y_train)
y_pred1 = LG_Model.predict(X_test)
false_positive, true_positive, _ =
roc_curve(y_test,LG_Model.decision_function(X_test))
plt.plot(false_positive,true_positive, label='Logistic Regression')


k=10
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=3)
KNN_Model_Final = KNeighborsClassifier(n_neighbors=k,weights='uniform')
KNN_Model_Final.fit(X_train,y_train)
y_pred2=KNN_Model_Final.predict_proba(X_test)
false_positive, true_positive, _ = roc_curve(y_test,y_pred2[:,1])
plt.plot(false_positive,true_positive, label='KNN')

Classify_Dummy = DummyClassifier(strategy="most_frequent")
Classify_Dummy.fit(X_train,y_train)
ypred3 = Classify_Dummy.predict_proba(X_test)
```

```
false_positive, true_positive, _ = roc_curve(y_test,ypred3[:,1])
plt.plot(false_positive,true_positive, label='Dummy Model', color ='red')


plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.plot([0, 1], [0, 1], color='green',linestyle='--',label ='Random')
#random classifier
plt.legend()
plt.show()
```