Name: Karan Dua
Student Id: 21331391

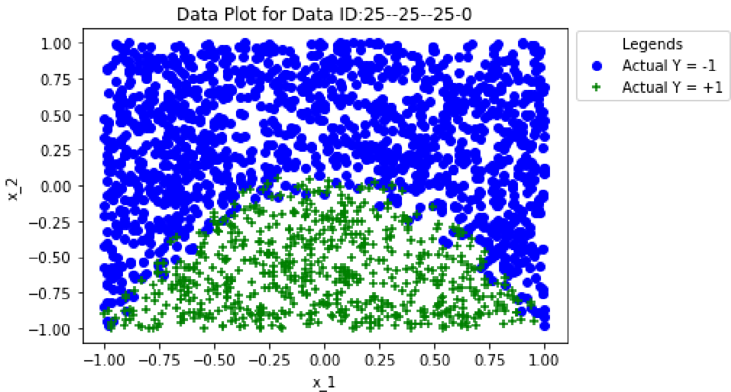
Course Code: CS7CS4-202223 MACHINE LEARNING

Question (i)(a) Reading the data, Visualising the data, Training Logistic Polynomial Features, selecting value of Maximum Polynomial Degree and Penalty Hyper Parameter C using cross validation

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. Please refer below head of the dataset**. My data id is: 25--25--25-0**:

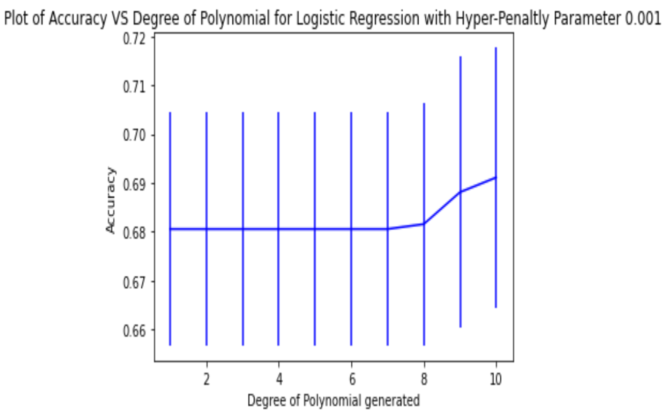
	x1	x2	y
0	-0.89	0.81	-1
1	0.33	0.98	-1
2	0.23	0.02	1
3	0.73	-0.66	1
4	-0.41	0.81	-1

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.

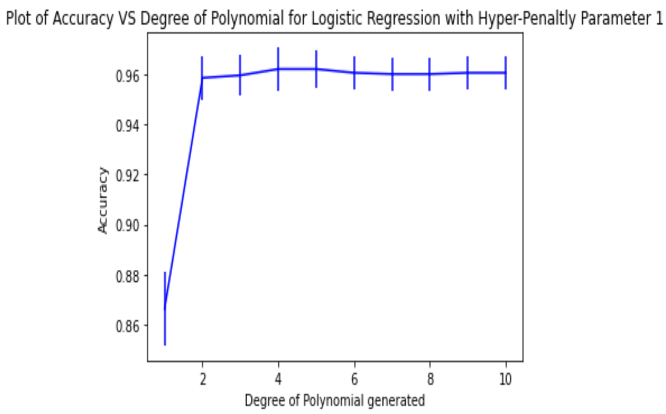


By looking at the scatter plot, I can identify that there is bias in the data provided. There are significantly less points for label y = +1 when compared to y = -1 and the data is not linearly separable which means a straight line cannot separate both label classes. Both can have implications on model's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

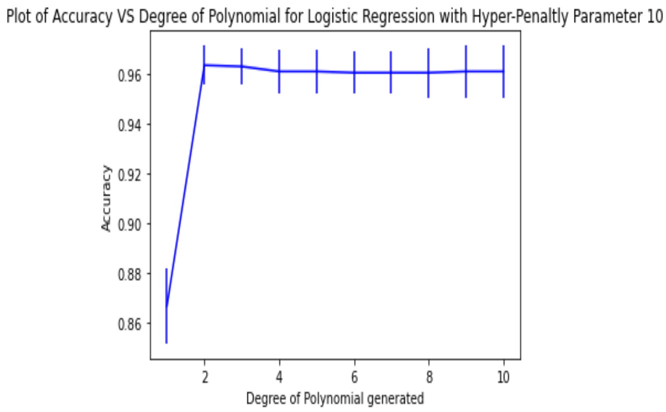
Now, we had to train a Logistic Regression Model with l2 penalty parameter to choose the maximum degree of polynomial that can be added to this data. To achieve this, I have created a function that uses 5-fold cross validation strategy to train LR model for different values of Penalty Parameter C and each for each value of C, I have added a range polynomial features to the dataset and trained the LR Model for each degree. For this part, I have chosen C range as [0.001, 1, 10, 1000] and Polynomial Degree range as [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. Using these parameters, I have calculated LR model's accuracy in each iteration of Cross validation and then I have plotted the Average accuracy and standard deviation on below plots:



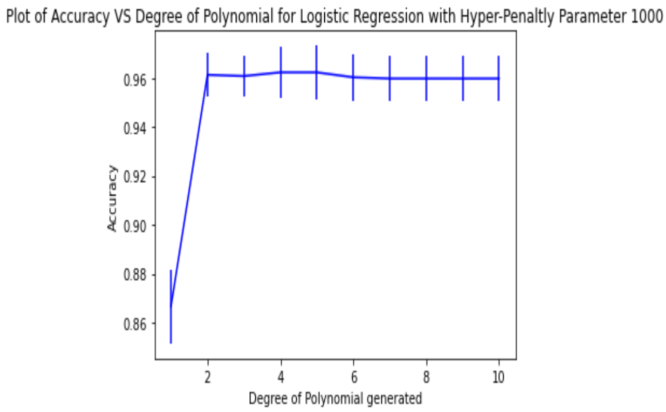
Plot 1: C = 0.001



Plot 2: C = 1



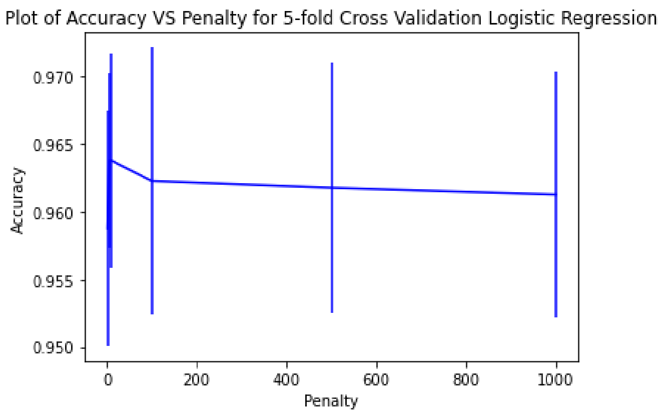
Plot 3: C = 10



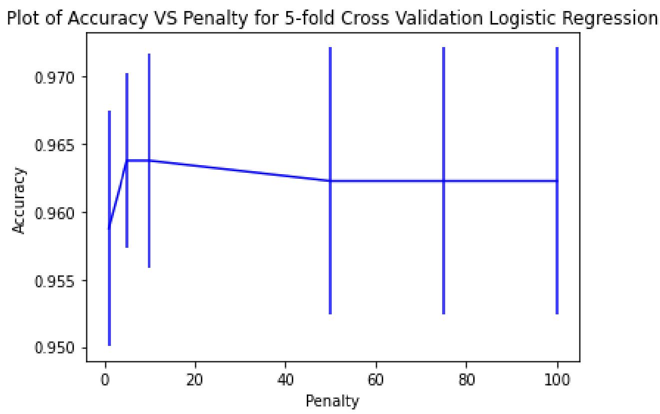
Plot 4: C = 1000

From Plot 1, we can Identify that the accuracy of the model is very low compared to the other plots. From plot 2, 3, 4 we can identify that model has almost similar performance for different penalty parameter and has a good performance for polynomial degree 2 and the standard deviation is also low. Therefore, I will choose the polynomial degree to be added to the data as 2 because we have good model performance on this degree, although for some higher degree 4, 5, 6 we see slightly better performance, but choosing lower degree will help us to keep the model simple.

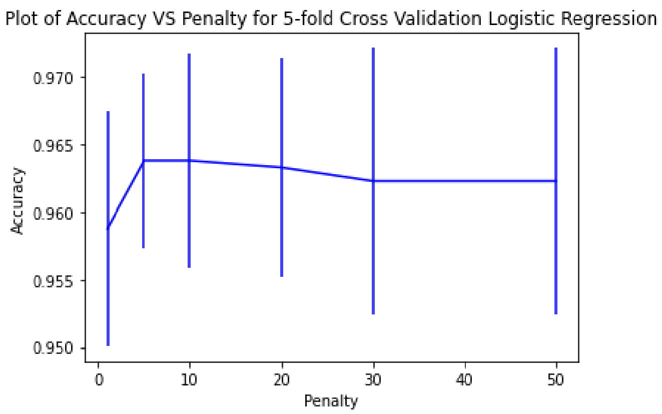
We chose the value of polynomial degree to be added to data as 2. Now we to find the optimal value of penalty parameter C for this degree. I have created a function that takes a range of penalty parameter and used 5-fold cross validation to train LR Model. In each iteration, I have calculated average accuracy and standard deviation and generated below plots.



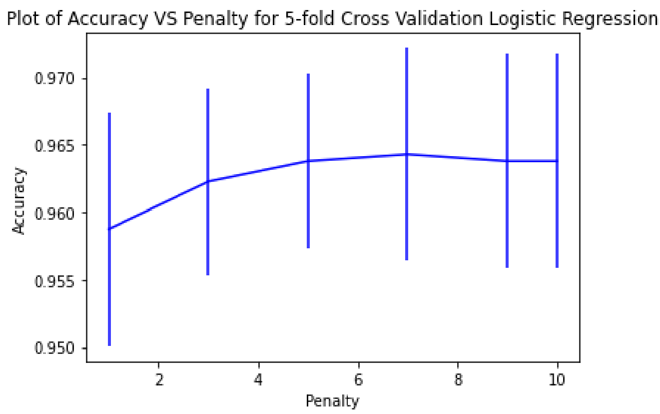
Plot 1: C = [1, 5, 10, 100, 500, 1000]



Plot 2: C = [1, 5, 10, 50, 75, 100]



Plot 3: C = [1, 5, 10, 20, 30, 50]

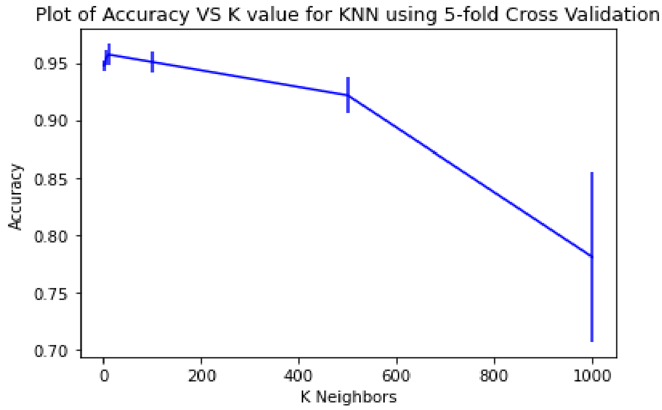


Plot 4: C = [1, 3, 5, 7, 9, 10]

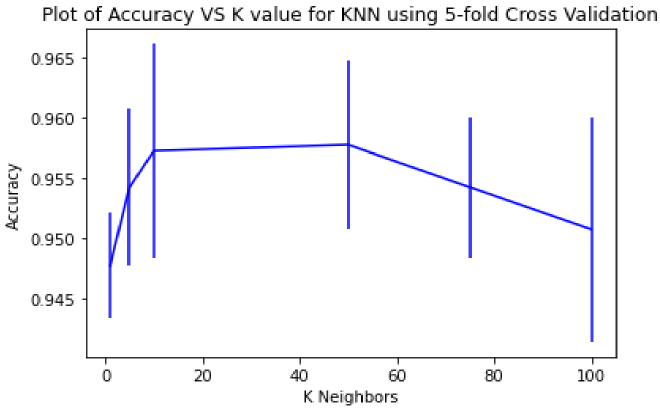
I have created Plot 1 on broad range of penalty parameters starting from 1 to 1000. By analysing this plot, I can see that Accuracy has converged by C=100. Again, I used same function to zoom into the plot by checking plot for penalty parameter 1 to 100. By analysing this plot, I can say that again Accuracy has converged by C=50. I have again plotted for C range 1 to 50 and then 1 to 10. Plot 4 shows that penalty parameter 5 is the optimal parameter for Logistic Regression. The primary reason for this is that Accuracy does not increase significantly for C greater than 5. Although for C=7, we have slightly higher accuracy, but I have chosen C=5 for two reasons. One, it has lower standard deviation. Second, lower penalty is also preferred for complex non-linear models as it penalises the complex model optimally.

Question (i)(b) Finding optimal number of neighbours(k) for KNN Model

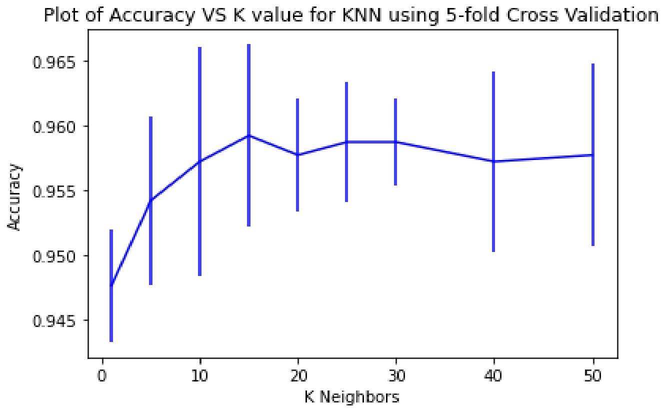
To find optimal value of K for KNN Model I have created a function that trains KNN model on a range of K values and for each K value, I have used 5-fold cross validation. For each iteration, I have calculated Accuracy and Standard deviation and generated below plots:



Plot 1: K = [1, 5, 10, 100, 500, 1000]



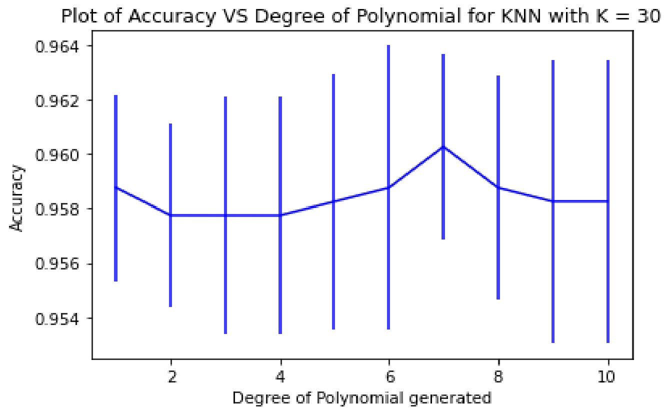
Plot 2: K = [1, 5, 10, 50, 75, 100]



Plot 3: K = [1, 5, 10, 15, 20, 25, 30, 40, 50]

I have created Plot 1 on broad range of K values starting from 1 to 1000. By analysing this plot, I can see that Accuracy has converged by K=100. Again, I used same function to zoom into the plot by checking plot for K range 1 to 100. By analysing this plot, I can say that again Accuracy has converged by K=50. I have again plotted for K range 1 to 50. Plot 3 shows that K = 30 is the optimal parameter for KNN. Although for K=15 we have slightly higher accuracy but standard deviation for K =30 is much smaller. Also, Accuracy has converged after K=30, therefore it can be considered as 'Elbow Point' for this graph.

I have also checked if adding polynomial features to KNN will help to improve the model's performance. For optimised K=30, I have added polynomial features in range 1 to 10. I have used 5-fold cross validation to get calculate accuracy and standard deviation, I have generated below plot for this:



From this plot we can conclude that adding higher degree polynomial does not significantly increase the accuracy. In-fact, there is decrease in accuracy for higher degrees. Although, for degree = 7, we see a slight increase in accuracy, but it will increase the complexity of our model, therefore, I have decided to not include it in our model.

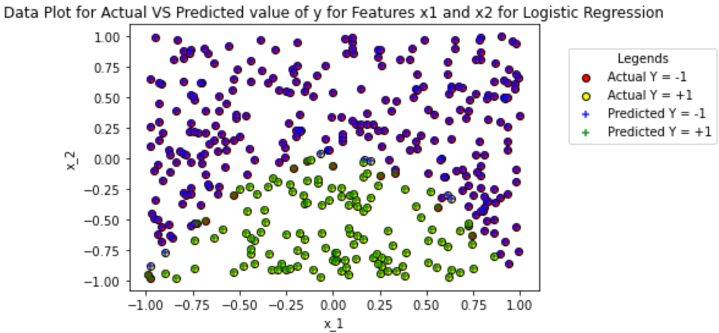
Question (i)(c) Final LR Model and KNN Model based on above hyper-parameters, Baseline models and Confusion Matrix

I have split the data in train and test data in 80:20 ratio. I have trained LR model for l2 penalty and C value = 5 and added degree = 2 features to data. Below are the model's coefficients and model's equation:

	Intercept	1	x1	x2	x1^2	x1 x2	x2^2
0	[0.196]	[-0.0]	[-0.051]	[-11.978]	[-12.607]	[0.061]	[0.836]

	Model Equation
0	(-0.051) * x1 + (-11.978) * x2 + (-12.607) * x1^2 + (0.061) * x1 x2 + (0.836) * x2^2 + (0.196)

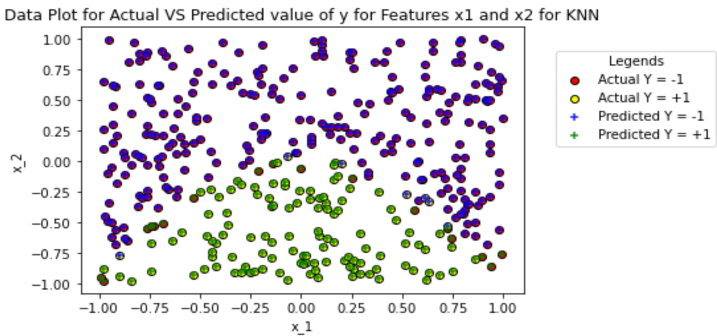
Then, I used test data to predict the labels. I have obtained below scatterplot for actual vs predicted datapoints.



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

Also, I can identify that there are certain misclassifications where test data marker 'Yellow o' is overlapped by Predicted marker is 'Blue +' and test data marker 'Red o' is overlapped by Predicted marker is 'Green +'

Similarly, for KNN, I have trained the model for optimised K = 30 and generated below scatterplot for actual vs. predicted data points.



Name: Karan Dua
Student Id: 21331391

Course Code: CS7CS4-202223 MACHINE LEARNING

Below are the Confusion Matrix for LR and KNN models:

	Predicted Negative	Predicted Postive
True Negative	264	13
True Positive	7	114

Confusion Matrix for Logistic Regression

	Predicted Negative	Predicted Positive
True Negative	260	17
True Positive	7	114

Confusion Matrix for KNN

I have used 2 Baseline Models i.e., Most frequent dummy baseline model (always predicts most frequent label) and Random Dummy baseline model (uses unform distribution). Below is the confusion matrix

	Predicted Negative	Predicted Postive
True Negative	277	0
True Positive	121	0

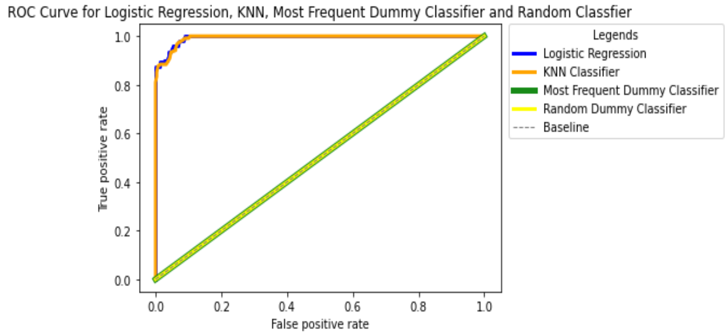
Confusion Matrix for Most Frequent

	Predicted Negative	Predicted Positive
True Negative	132	145
True Positive	64	57

Confusion Matrix for Random

Question (i)(d) ROC curve

I have generated ROC Curve for LR model, KNN model and Baseline models. Below is the curve generated:



Question (i)(e) Model Performance Evaluation using Confusion Matrix and ROC Curve and recommendations

Performance evaluation using Confusion Matrix:

LR Model Statistics:

	True Positive	True Negative	False Positive	False Negative	Accuracy	True Positive Rate	False Positive Rate	Precision
0	114	264	13	7	94.974874	0.942149	0.046931	0.897638

KNN Model Statistics:

	True Positive	True Negative	False Positive	False Negative	Accuracy	True Positive Rate	False Positive Rate	Precision
0	114	260	17	7	93.969849	0.942149	0.061372	0.870229

Both KNN and LR models have very high accuracy and are performing well. This means that both models were able to predict the labels correctly on test data. This is further confirmed as both the models have very high number of True Positive and True Negatives and very low False Positives and False Negatives. Furthermore, True Positive Rate (TP/(TP+FN) of both the models is identical i.e., 94.21%. TPR is also called as recall rate. Recall means proportion of actual positives that were predicted correctly by the model. It is model's ability to detect positive samples. High recall rate shows that our models are highly sensitive to positive. This means that our models are giving high quality predictions. Also, False Positive Rate (FP/(FP + TN) of the LR model is 4.69% which is lower than KNN at 6.31%. This means that LR model is performing better than KNN model on this data. This is further proven by calculating Precision (TP/(TP+FP)) of both the models. Precision represents the model's reliability to in predicting a sample as positive. This means it represent the proportion of positive predictions that were truly positive. LR as 89.76% precision which is higher than KNN at 87.02%.
When compared to baseline models, Most Frequent Baseline model has an Accuracy of 69.84% and Random Baseline classifier that has an accuracy of 47.48%. Both LR and KNN have performed better than these models. Based on our analysis of Confusion Matrix we can recommend LR model for this data.

When we look at ROC curves, again both LR and KNN models are performing very well. For a model that predicts 100% labels correctly i.e., its Accuracy is 100%. This means that True Positive Rate is 100% and False Positive Rate is 0%. When we plot ROC curve for this model, we get a point on the Top-Left corner of the plot where TPR is 1.0 and FPR is 0.0. Now, to check which model is performing better, we check the graph which is closest to this Top-Left corner point. Based on the above graph, graphs for both the models overlaps for most of the points but graph for LR model is slightly closer to the Top-Left corner point.

Therefore, we can conclude that LR model is performing better than KNN model. When compared to Baseline model, ROC curves for both Most Frequent Classifier and Random Classifier lies are 45 degrees. Therefore, both of our models are performing better than baseline models.

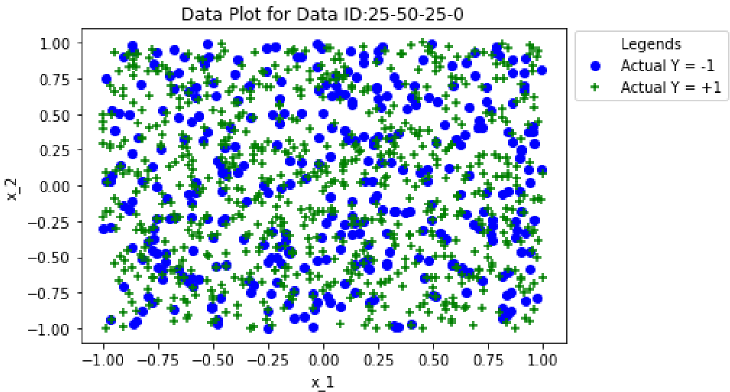
Thus, we can conclude that for this data, both the models are performing better than baseline models when analysed using Confusion Matrix Statistics and ROC curves and LR performs slightly better than KNN, therefore LR is recommended model.

Question (ii)(a) Reading the data, Visualising the data, Training Logistic Polynomial Features, selecting value of Maximum Polynomial Degree and Penalty Hyper Parameter C using cross validation

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. Please refer below head of the dataset. **My data id is: 25--25--25-0**:

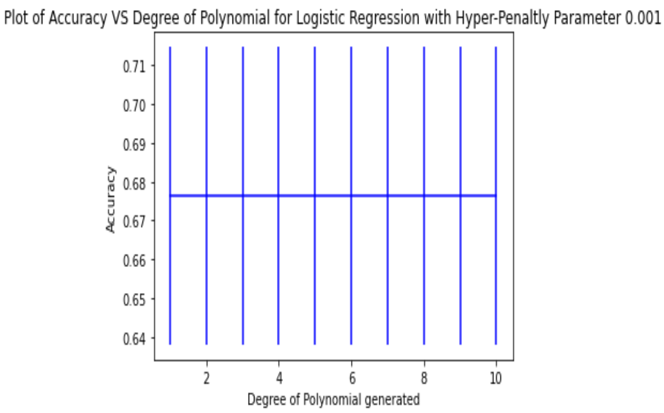
	x1	x2	y
0	-0.17	-0.34	-1
1	-0.82	0.33	1
2	-0.23	-0.26	1
3	-1.00	0.50	1
4	0.37	0.71	1

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.

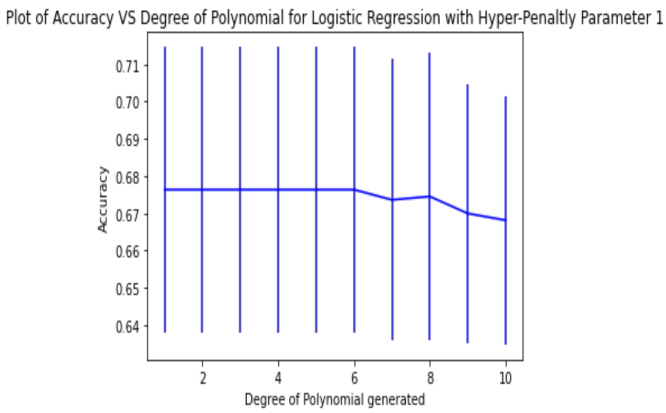


By looking at the scatter plot, I can identify that the data is not linearly separable which means a straight line cannot separate both label classes. Therefore, it is not possible to get decision boundary for this dataset. This can have implications on model's performance, which can be checked later.

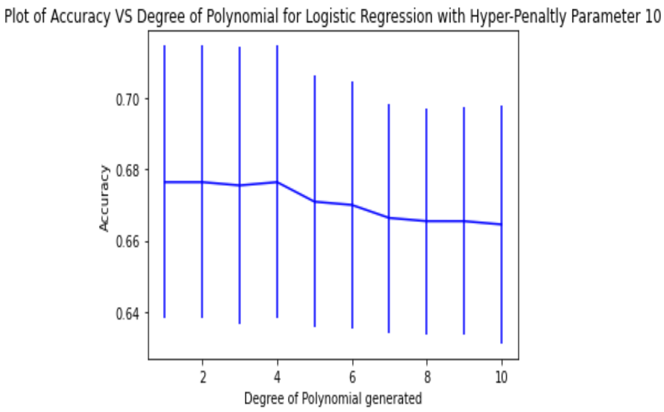
For Logistic Regression, to identify the polynomial degree that can be added to data I have followed same steps as mentioned in (i)(a). I have obtained below plots for degree range for give penalty parameter C:



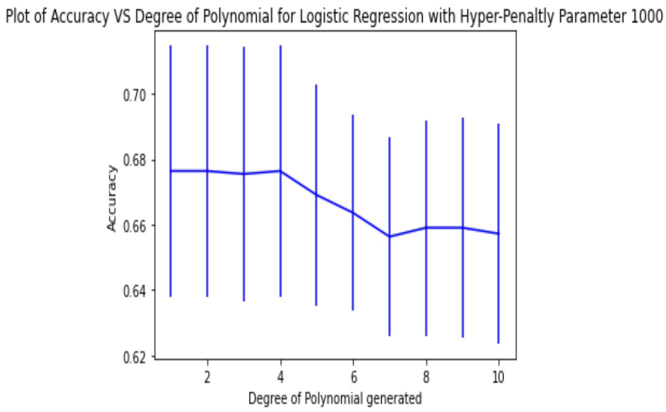
Plot 1: C = 0.001



Plot 2: C = 1



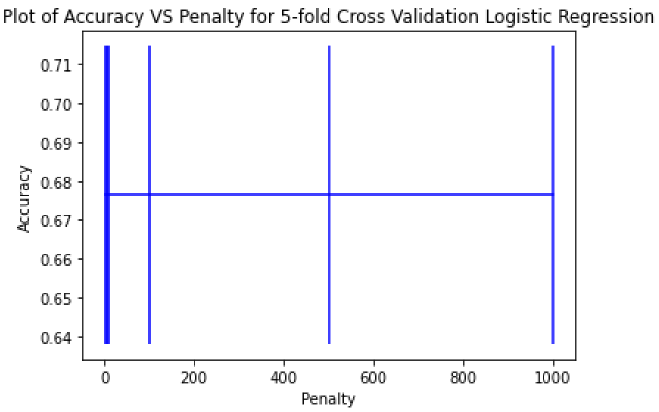
Plot 3: C = 10



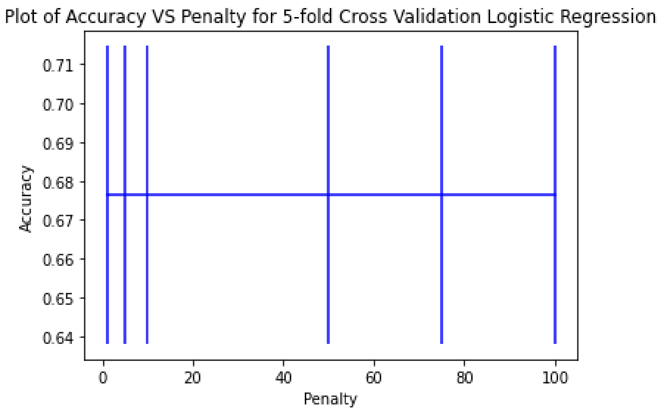
Plot 4: C = 1000

From Plot 1, 2, 3 and 4, we can identify that for any value of penalty parameter, adding higher degree polynomials to data has either given same accuracy or resulted in decrease in accuracy. This could be because the data is linearly non-separable and LR Model is not able to identify a decision boundary for this data. Adding polynomial features is not helping the LR model to get the decision boundary and choosing lower degree will help us to keep the model simple. Therefore, I have decided to not to add any higher degree polynomial to the data.

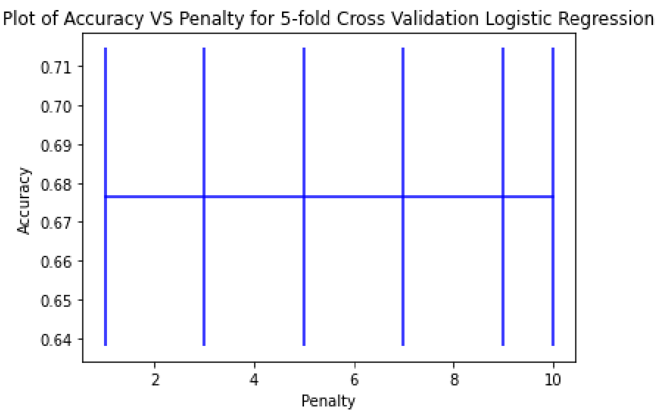
We chose the value of polynomial degree to be added to data as 1. Now we to find the optimal value of penalty parameter C for this degree. I have followed same steps as {i)(a) and generated below plots.



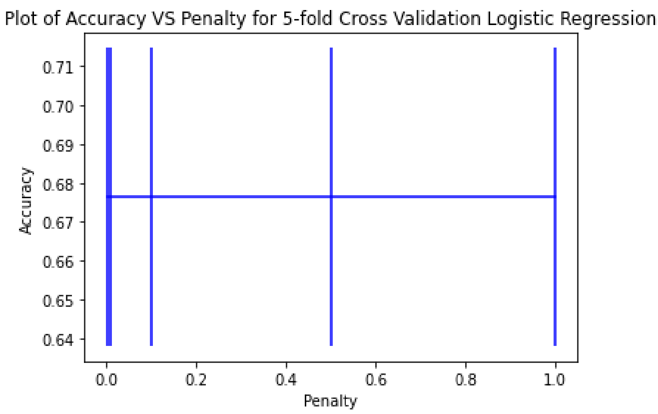
Plot 1: C = [1, 5, 10, 100, 500, 1000]



Plot 2: C = [1, 5, 10, 50, 75, 100]



Plot 3: C = [1, 3, 5, 7, 9, 10]

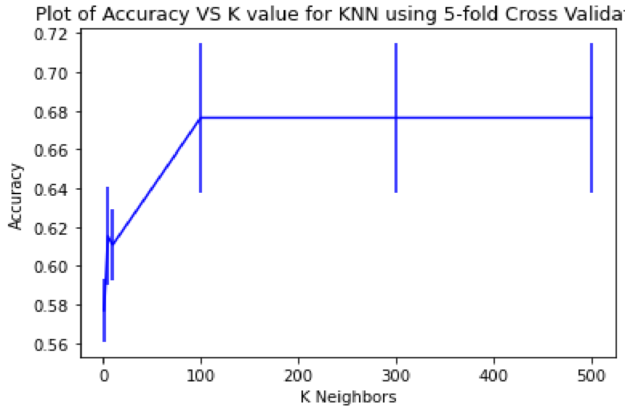


Plot 4: C = [0.001, 0.01, 0.1, 0.5, 1]

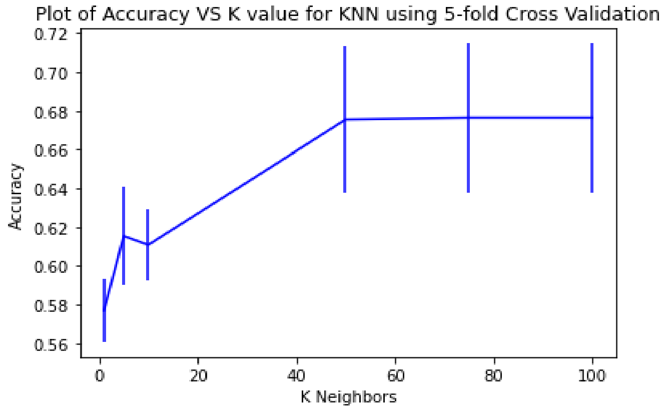
From Plot 1, we can analyse that accuracy has not changes at all and remained constant for entire range from 1 to 1000. Again, in Plot 2 and 3, I zoomed into the plot by checking plot for penalty parameter 1 to 100 and range 1 to 50. The accuracy or standard deviation of the model has not changed. Then I generated Plot 4 for range of C between 0 and 1. Again, the accuracy and standard deviation has remained constant. Therefore, I have chosen C=1 because there was no improvement in performance of model by adding penalty and lower penalty is also preferred for complex non-linear models as it penalises the complex model optimally.

Question (ii)(b) Finding optimal number of neighbours(k) for KNN Model

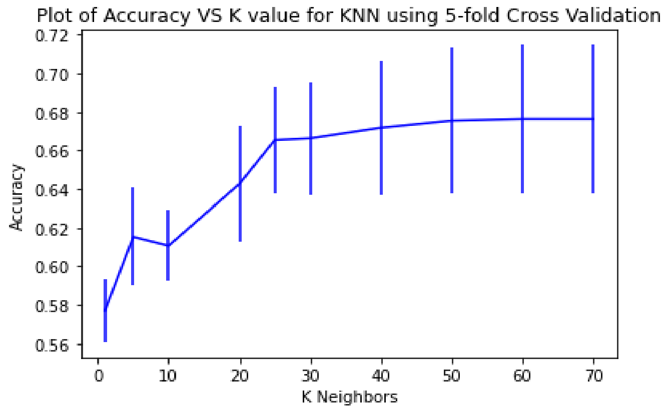
I have followed same steps as (i)(b) to find optimal value of K and generated below plots:



Plot 1: K = [1, 5, 10, 100, 500]



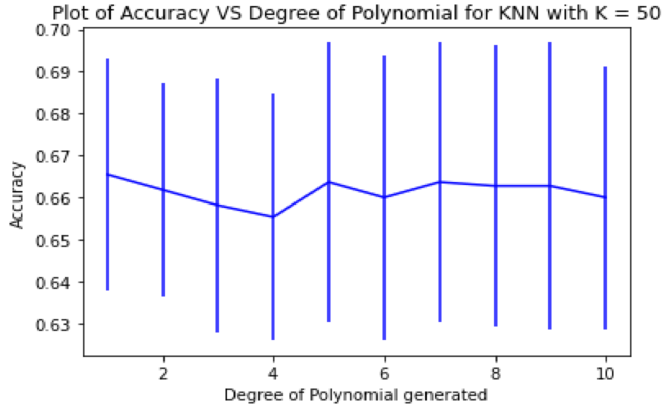
Plot 2: K = [1, 5, 10, 50, 75, 100]



Plot 3: K = [1, 5, 10, 20, 25, 30, 40, 50, 60, 70]

I have created Plot 1 on broad range of K values starting from 1 to 1000. By analysing this plot, I can see that Accuracy has converged by K=100. Again, I used same function to zoom into the plot by checking plot for K range 1 to 100. By analysing this plot, I can say that again Accuracy has converged by K=70. I have again plotted for K range 1 to 70. Plot 3 shows that K=25 can be considered as 'Elbow Point' for this graph. Accuracy does not increase significantly after this point even though we have increase K value significantly. Therefore, K = 25 is the optimal parameter for KNN.

I have also checked if adding polynomial features to KNN will help to improve the model's performance. For optimised K=25, I have added polynomial features in range 1 to 10. I have used 5-fold cross validation to get calculate accuracy and standard deviation, I have generated below plot for this:



Name: Karan Dua
Student Id: 21331391

Course Code: CS7CS4-202223 MACHINE LEARNING

From this plot we can conclude that adding higher degree polynomial does not significantly increase the accuracy. Although, for degree = 5, we see a slight increase in accuracy, but it will increase the complexity of our model, therefore, I have decided to not include it in our model.

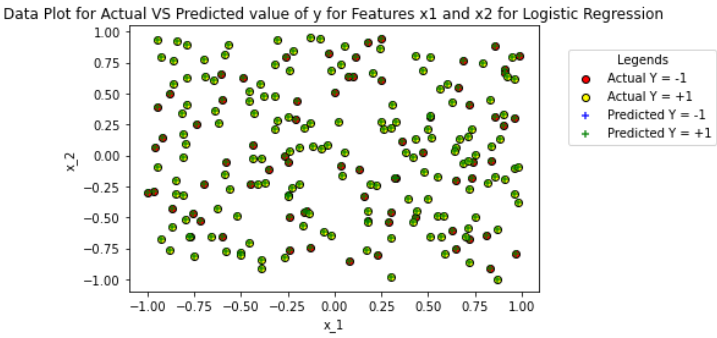
Question (ii)(c) Final LR Model and KNN Model based on above hyper-parameters, Baseline models and Confusion Matrix

I have split the data in train and test data in 80:20 ratio. I have trained LR model for l2 penalty and C value = 1 and didn't include any high degree polynomial features to data. Below are the model's coefficients and model's equation:

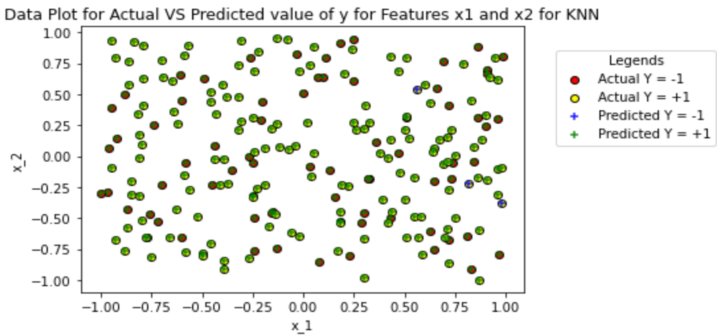
	Intercept	1	x1	x2
0	[0.736]	[0.0]	[-0.074]	[-0.042]

	Model Equation
0	(-0.074) * x1 + (-0.042) * x2 + (0.736)

Then, I used test data to predict the labels. I have obtained below scatterplot for actual vs predicted datapoints.



Similarly, for KNN, I have trained the model for optimised K = 25 and generated below scatterplot for actual vs. predicted data points.



From these plots we can identify that, there are some misclassification errors done by the model. We will confirm the same while evaluating the model's performance.

Below are the Confusion Matrix for LR and KNN models:

	Predicted Negative	Predicted Positive
True Negative	0	71
True Positive	0	149

Confusion Matrix for Logistic Regression

	Predicted Negative	Predicted Positive
True Negative	0	71
True Positive	3	146

Confusion Matrix for KNN

I have used 2 Baseline Models i.e., Most frequent dummy baseline model (always predicts most frequent label) and Random Dummy baseline model (uses unform distribution). Below is the confusion matrix

	Predicted Negative	Predicted Positive
True Negative	0	71
True Positive	0	149

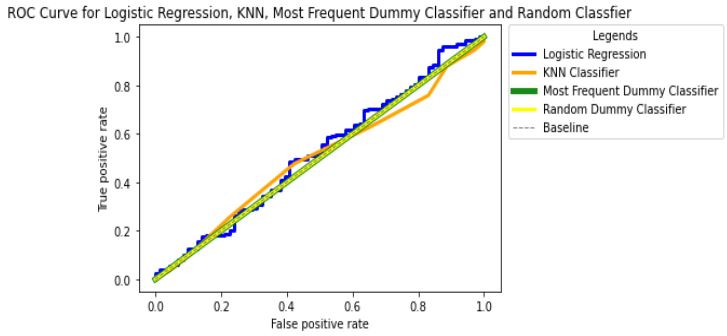
Confusion Matrix for Most Frequent

	Predicted Negative	Predicted Positive
True Negative	40	31
True Positive	71	78

Confusion Matrix for Random

Question (ii)(d) ROC curve

I have generated ROC Curve for LR model, KNN model and Baseline models. Below is the curve generated:



Question (ii)(e) Model Performance Evaluation using Confusion Matrix and ROC Curve and recommendations

Performance evaluation using Confusion Matrix:

LR Model Statistics:

	True Positive	True Negative	False Positive	False Negative	Accuracy	True Positive Rate	False Positive Rate	Precision
0	149	0	71	0	67.727273	1.0	1.0	0.677273

KNN Model Statistics:

	True Positive	True Negative	False Positive	False Negative	Accuracy	True Positive Rate	False Positive Rate	Precision
0	146	0	71	3	66.363636	0.979866	1.0	0.672811

Both KNN and LR models have low accuracy, and their performance is not up to the mark. LR model always predicted the majority class in the test data which is y = 1. Similarly, the LR model is not able to correctly predict y = -1 label for any datapoint in test data. Because of this the model's True Positive Rate and False Positive Rate is 100%, which can be considered worst case scenario for any model. Also, the model's precision is 67.28%. This means that this model has high recall and low precision.

KNN has lower accuracy than LR and is True Positive rate of 97.98% is lower than KNN. This means that KNN was even not able to predict the majority class y = 1 correctly for all test datapoints. It has same FPR 100% as LR which means it has incorrectly labelled all the y = -1 points. Its precision of 67.28% is also lower than LR model. When compared to baseline models, LR has same accuracy as Most Frequent baseline model as both models predict the majority class always and KNN has lower accuracy than Most Frequent Baseline model. For Random Classification model, the accuracy comes out to be 53.63% which is lower than both KNN and LR. Therefore, we can conclude that based on Confusion Matrix statistics analysis we can use either Most Frequent baseline model or Logistic Regression model as both have same accuracy. I will prefer Baseline model in this scenario because it is much easier to implement, and no parameter tuning is required.

From ROC curves analysis, we can see that ROC curves for LR and KNN model are close to the baseline 45-degree line. This means that their performance is like baseline models like Most Frequent Baseline model and Random Baseline model. None of these models come close to the Top Left corner point which is the optimal point in ROC Curve. Again, we can say that we can use Most Frequent baseline model based on ROC curve analysis as it is easier to implement.

The main reason for LR and KNN to underperform for this data is that the data is not linearly separable, and no decision boundary can be drawn in the data. Even for KNN, selecting optimal number of neighbours is difficult both label points are close to each other, and no small patches can be identified in the data. Also, selecting higher N will lead to overfitting of training data.

Thus, we can conclude that for this data, both the models are underperforming and are no better than baseline models when analysed using Confusion Matrix Statistics and ROC curves. Therefore, Most Frequent baseline model is recommended model as it is very easy to implement, and no parameter tuning is required.

Name: Karan Dua
Student Id: 21331391

Course Code: CS7CS4-202223 MACHINE LEARNING

Appendix: Code for Question 1 - All Parts

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plot
from IPython.display import display
import random


from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.dummy import DummyClassifier
from sklearn.metrics import roc_curve

df = pd.read_csv("data_week4_1.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]

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:25--25--25-0')
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()

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]})
display(actual_df)

poly_degree_range = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
penalty_parameter_range = [0.001, 1, 10, 1000]

def calculate_accuracy_stddev_for_given_penalty(penalty_parameter) :
  k_fold_split = 5

  k_fold_split_function =  KFold(n_splits = k_fold_split)

  accuracy_poly_degree = []
  standard_deviation_poly_degree = []
  Logistic_Regression_Model = LogisticRegression(penalty = 'l2', C = penalty_parameter, max_iter = 10000)

  for poly_degree in poly_degree_range :
    poly_features_function = PolynomialFeatures(poly_degree)
    x_poly_features  = poly_features_function.fit_transform(x)
    accuaracy_fold = []
    for train_data_index, test_data_index in k_fold_split_function.split(x_poly_features):
      Logistic_Regression_Model.fit(x_poly_features[train_data_index], y[train_data_index])
      predictions = Logistic_Regression_Model.predict(x_poly_features[test_data_index])
      accuaracy_fold.append(accuracy_score(y[test_data_index], predictions))
```

```
        accuracy_poly_degree.append(np.array(accuaracy_fold).mean())
        standard_deviation_poly_degree.append(np.array(accuaracy_fold).std())

    return accuracy_poly_degree, standard_deviation_poly_degree

for penalty_parameter in penalty_parameter_range :

    accuracy_poly_degree, standard_deviation_poly_degree =
calculate_accuracy_stddev_for_given_penalty(penalty_parameter)
    plot.figure()
    plot.errorbar(poly_degree_range, accuracy_poly_degree, yerr = standard_deviation_poly_degree, color = 'blue')
    plot.xlabel('Degree of Polynomial generated')
    plot.ylabel('Accuracy')
    plot.title('Plot of Accuracy VS Degree of Polynomial for Logistic Regression with Hyper-Penaltly Parameter
{}'.format(penalty_parameter))
    plot.show()


def calculate_accuracy_stddev_for_penalty_ranges(penalty_parameter_ranges, poly_degree) :

    k_fold_split = 5

    k_fold_split_function =  KFold(n_splits = k_fold_split)

    accuracy_penalty = []
    standard_deviation_penalty = []

    poly_features_function = PolynomialFeatures(poly_degree)
    x_poly_features  = poly_features_function.fit_transform(x)

    for penalty_parameter in penalty_parameter_ranges :

        Logistic_Regression_Model = LogisticRegression(penalty = 'l2', C = penalty_parameter, max_iter = 10000)
        accuaracy_fold = []
        for train_data_index, test_data_index in k_fold_split_function.split(x_poly_features):
            Logistic_Regression_Model.fit(x_poly_features[train_data_index], y[train_data_index])
            predictions = Logistic_Regression_Model.predict(x_poly_features[test_data_index])
            accuaracy_fold.append(accuracy_score(y[test_data_index], predictions))

        accuracy_penalty.append(np.array(accuaracy_fold).mean())
        standard_deviation_penalty.append(np.array(accuaracy_fold).std())

    return accuracy_penalty, standard_deviation_penalty
penalty_parameters = [1, 5, 10, 100, 500, 1000]
poly_degree = 2
accuracy_penalty, standard_deviation_penalty = calculate_accuracy_stddev_for_penalty_ranges(penalty_parameters,
poly_degree)

plot.figure()
plot.errorbar(penalty_parameters, accuracy_penalty, yerr = standard_deviation_penalty, color = 'blue')
plot.xlabel('Penalty')
plot.ylabel('Accuracy')
plot.title('Plot of Accuracy VS Penalty for 5-fold Cross Validation Logistic Regression')
plot.show()

penalty_parameters = [1, 5, 10, 50, 75, 100]
poly_degree = 2
```

```
accuracy_penalty, standard_deviation_penalty = calculate_accuracy_stddev_for_penalty_ranges(penalty_parameters, poly_degree)

plot.figure()
plot.errorbar(penalty_parameters, accuracy_penalty, yerr = standard_deviation_penalty, color = 'blue')
plot.xlabel('Penalty')
plot.ylabel('Accuracy')
plot.title('Plot of Accuracy VS Penalty for 5-fold Cross Validation Logistic Regression')
plot.show()

penalty_parameters = [1, 5, 10, 20, 30, 50]
poly_degree = 2
accuracy_penalty, standard_deviation_penalty = calculate_accuracy_stddev_for_penalty_ranges(penalty_parameters, poly_degree)

plot.figure()
plot.errorbar(penalty_parameters, accuracy_penalty, yerr = standard_deviation_penalty, color = 'blue')
plot.xlabel('Penalty')
plot.ylabel('Accuracy')
plot.title('Plot of Accuracy VS Penalty for 5-fold Cross Validation Logistic Regression')
plot.show()

penalty_parameters = [1, 3, 5, 7, 9, 10]
poly_degree = 2
accuracy_penalty, standard_deviation_penalty = calculate_accuracy_stddev_for_penalty_ranges(penalty_parameters, poly_degree)

plot.figure()
plot.errorbar(penalty_parameters, accuracy_penalty, yerr = standard_deviation_penalty, color = 'blue')
plot.xlabel('Penalty')
plot.ylabel('Accuracy')
plot.title('Plot of Accuracy VS Penalty for 5-fold Cross Validation Logistic Regression')
plot.show()

def calculate_accuracy_stddev_for_K_ranges(K_range) :

    k_fold_split = 5

    k_fold_split_function =  KFold(n_splits = k_fold_split)

    accuracy_k = []
    standard_deviation_k = []

    for k in K_range :

        KNN_Model = KNeighborsClassifier(n_neighbors = k, weights = 'uniform')
        accuaracy_fold = []
        for train_data_index, test_data_index in k_fold_split_function.split(x):
            KNN_Model.fit(x[train_data_index], y[train_data_index])
            predictions = KNN_Model.predict(x[test_data_index])
            accuaracy_fold.append(accuracy_score(y[test_data_index], predictions))

        accuracy_k.append(np.array(accuaracy_fold).mean())
        standard_deviation_k.append(np.array(accuaracy_fold).std())

    return accuracy_k, standard_deviation_k

K_range = [1, 5, 10, 100, 500, 1000]
accuracy_k, standard_deviation_k = calculate_accuracy_stddev_for_K_ranges(K_range)
```

```
plot.figure()
plot.errorbar(K_range, accuracy_k, yerr = standard_deviation_k, color = 'blue')
plot.xlabel('K Neighbors')
plot.ylabel('Accuracy')
plot.title('Plot of Accuracy VS K value for KNN using 5-fold Cross Validation')
plot.show()

K_range = [1, 5, 10, 50, 75, 100]
accuracy_k, standard_deviation_k = calculate_accuracy_stddev_for_K_ranges(K_range)

plot.figure()
plot.errorbar(K_range, accuracy_k, yerr = standard_deviation_k, color = 'blue')
plot.xlabel('K Neighbors')
plot.ylabel('Accuracy')
plot.title('Plot of Accuracy VS K value for KNN using 5-fold Cross Validation')
plot.show()

K_range = [1, 5, 10, 15, 20, 25, 30, 40, 50]
accuracy_k, standard_deviation_k = calculate_accuracy_stddev_for_K_ranges(K_range)

plot.figure()
plot.errorbar(K_range, accuracy_k, yerr = standard_deviation_k, color = 'blue')
plot.xlabel('K Neighbors')
plot.ylabel('Accuracy')
plot.title('Plot of Accuracy VS K value for KNN using 5-fold Cross Validation')
plot.show()

k_optimised = 30
poly_degree_range = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

k_fold_split = 5

k_fold_split_function =  KFold(n_splits = k_fold_split)

accuracy_poly_degree = []
standard_deviation_poly_degree = []
KNN_Model = KNeighborsClassifier(n_neighbors = k_optimised, weights = 'uniform')

for poly_degree in poly_degree_range :
   poly_features_function = PolynomialFeatures(poly_degree)
   x_poly_features  = poly_features_function.fit_transform(x)
   accuaracy_fold = []
   for train_data_index, test_data_index in k_fold_split_function.split(x_poly_features):
      KNN_Model.fit(x_poly_features[train_data_index], y[train_data_index])
      predictions = KNN_Model.predict(x_poly_features[test_data_index])
      accuaracy_fold.append(accuracy_score(y[test_data_index], predictions))

   accuracy_poly_degree.append(np.array(accuaracy_fold).mean())
   standard_deviation_poly_degree.append(np.array(accuaracy_fold).std())

plot.figure()
plot.errorbar(poly_degree_range, accuracy_poly_degree, yerr = standard_deviation_poly_degree, color = 'blue')
plot.xlabel('Degree of Polynomial generated')
plot.ylabel('Accuracy')
plot.title('Plot of Accuracy VS Degree of Polynomial for KNN with K = 30')
plot.show()

random.seed(123)
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, train_size = 0.8)

penalty_parameter = 5
poly_degree = 2

poly_features_function = PolynomialFeatures(poly_degree)
x_train_poly_features  = poly_features_function.fit_transform(x_train)
x_test_poly_features  = poly_features_function.fit_transform(x_test)

temp_df = pd.DataFrame(x, columns = ['x1', 'x2'])
feature_names = poly_features_function.get_feature_names(temp_df.columns)
feature_names.insert(0, 'Intercept')

Logistic_Regression_Model_Final = LogisticRegression(penalty = 'l2', C = penalty_parameter, max_iter = 10000)
Logistic_Regression_Model_Final.fit(x_train_poly_features, y_train)


logistic_regression_params_df = pd.DataFrame(columns = feature_names)
model_dict = {}
model_dict['Intercept'] = [np.around(Logistic_Regression_Model_Final.intercept_[0], decimals = 3)]
for i in range(1, 7) :
    model_dict[feature_names[i]] = [np.around(Logistic_Regression_Model_Final.coef_[0][i-1], decimals = 3)]

logistic_regression_params_df = logistic_regression_params_df.append(model_dict, ignore_index = True)
logistic_regression_params_df = logistic_regression_params_df.style.applymap(lambda x:'white-space:nowrap')
display(logistic_regression_params_df)


logistic_regression_model_equation_df = pd.DataFrame(columns = ['Model Equation'])
model_eq_dict = {}
equation_string = ''
for i in range(1, 7) :
    coeff = np.around(Logistic_Regression_Model_Final.coef_[0][i-1], decimals = 3)
    if coeff != 0 :
        equation_string += '(' + str(coeff) + ')' + ' * ' + feature_names[i] + ' + '

equation_string += '(' + str(np.around(Logistic_Regression_Model_Final.intercept_[0], decimals = 3)) + ')'
model_eq_dict['Model Equation'] = equation_string

logistic_regression_model_equation_df = logistic_regression_model_equation_df.append(model_eq_dict, ignore_index = True)
logistic_regression_model_equation_df = logistic_regression_model_equation_df.style.set_properties(**{'text-align': 'left'})
logistic_regression_model_equation_df = logistic_regression_model_equation_df.applymap(lambda x:'white-space:nowrap')
display(logistic_regression_model_equation_df)


LR_Predictions = Logistic_Regression_Model_Final.predict(x_test_poly_features)

test_temp_df = pd.DataFrame(x_test, columns = ['x1','x2'])
x1_test = test_temp_df[['x1']]
x2_test = test_temp_df[['x2']]


plot.scatter(x1_test[np.array(y_test) == -1], x2_test[np.array(y_test) == -1], color='red', marker="o", edgecolors="black")
plot.scatter(x1_test[np.array(y_test) == 1], x2_test[np.array(y_test) == 1], color='yellow', marker="o", edgecolors="black")
plot.scatter(x1_test[LR_Predictions == -1], x2_test[LR_Predictions == -1], color='blue', marker="+", alpha=0.85)
plot.scatter(x1_test[LR_Predictions == 1], x2_test[LR_Predictions == 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=(1.45, 0.53))
plot.show()

k_optimised = 30

KNN_Model_Final = KNeighborsClassifier(n_neighbors = k_optimised, weights = 'uniform')

KNN_Model_Final.fit(x_train, y_train)
KNN_predictions = KNN_Model_Final.predict(x_test)

test_temp_df = pd.DataFrame(x_test, columns = ['x1','x2'])
x1_test = test_temp_df[['x1']]
x2_test = test_temp_df[['x2']]


plot.scatter(x1_test[np.array(y_test) == -1], x2_test[np.array(y_test) == -1], color='red', marker="o", edgecolors="black")
plot.scatter(x1_test[np.array(y_test) == 1], x2_test[np.array(y_test) == 1], color='yellow', marker="o", edgecolors="black")
plot.scatter(x1_test[KNN_predictions == -1], x2_test[KNN_predictions == -1], color='blue', marker="+", alpha=0.85)
plot.scatter(x1_test[KNN_predictions == 1], x2_test[KNN_predictions == 1], color='green', marker="+", alpha=0.85)
plot.title('Data Plot for Actual VS Predicted value of y for Features x1 and x2 for KNN')
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=(1.45, 0.53))
plot.show()

y_test_postive = np.count_nonzero(y_test == 1)
y_test_negative = np.count_nonzero(y_test == -1)

actual_test_df = pd.DataFrame({"Test Data Actual Positive":[y_test_postive], "Test Data Actual Negative":[y_test_negative]})
display(actual_test_df)

LR_true_positive = 0
LR_true_negative = 0
LR_false_positive = 0
LR_false_negative = 0
for i in range(len(y_test)) :
  if np.array(y_test)[i] == 1 and LR_Predictions[i] == 1 :
    LR_true_positive += 1
  elif np.array(y_test)[i] == -1 and LR_Predictions[i] == -1 :
    LR_true_negative += 1
  elif np.array(y_test)[i] == 1 and LR_Predictions[i] == -1 :
    LR_false_negative += 1
  elif np.array(y_test)[i] == -1 and LR_Predictions[i] == 1 :
    LR_false_positive += 1

LR_Accuracy = ((LR_true_positive + LR_true_negative) / (LR_true_positive + LR_true_negative + LR_false_positive + LR_false_negative)) * 100

LR_True_Positive_Rate = (LR_true_positive) / (LR_true_positive + LR_false_negative)
LR_False_Positive_Rate = (LR_false_positive) / (LR_false_positive + LR_true_negative)
LR_Precision = (LR_true_positive) / (LR_true_positive + LR_false_positive)

LR_actual_predicted_df = pd.DataFrame({"True Positive":[LR_true_positive], "True Negative":[LR_true_negative], "False Positive":[LR_false_positive], "False Negative":[LR_false_negative], "Accuracy":[LR_Accuracy], "True Positive Rate":[LR_True_Positive_Rate], "False Positive Rate":[LR_False_Positive_Rate], "Precision":[LR_Precision]})
```

```
display(LR_actual_predicted_df)

KNN_true_positive = 0
KNN_true_negative = 0
KNN_false_positive = 0
KNN_false_negative = 0
for i in range(len(y_test)) :
    if np.array(y_test)[i] == 1 and KNN_predictions[i] == 1 :
        KNN_true_positive += 1
    elif np.array(y_test)[i] == -1 and KNN_predictions[i] == -1 :
        KNN_true_negative += 1
    elif np.array(y_test)[i] == 1 and KNN_predictions[i] == -1 :
        KNN_false_negative += 1
    elif np.array(y_test)[i] == -1 and KNN_predictions[i] == 1 :
        KNN_false_positive += 1

KNN_Accuracy = ((KNN_true_positive + KNN_true_negative) / (KNN_true_positive + KNN_true_negative + KNN_false_positive
+ KNN_false_negative)) * 100

KNN_y_pos_1_correctly_predicted = (KNN_true_positive/y_test_postive)*100

KNN_y_neg_1_correctly_predicted = (KNN_true_negative/y_test_negative)*100

KNN_True_Positive_Rate = (KNN_true_positive) / (KNN_true_positive + KNN_false_negative)
KNN_False_Positive_Rate = (KNN_false_positive) / (KNN_false_positive + KNN_true_negative)
KNN_Precision = (KNN_true_positive) / (KNN_true_positive + KNN_false_positive)

KNN_actual_predicted_df = pd.DataFrame({"True Positive":[KNN_true_positive], "True Negative":[KNN_true_negative], "False
Positive":[KNN_false_positive], "False Negative":[KNN_false_negative], "Accuracy":[KNN_Accuracy], "True Positive
Rate":[KNN_True_Positive_Rate], "False Positive Rate":[KNN_False_Positive_Rate], "Precision":[KNN_Precision]})
display(KNN_actual_predicted_df)

LR_Confusion_Matrix = confusion_matrix(y_test, LR_Predictions)
LR_Confusion_Matrix_df = pd.DataFrame(LR_Confusion_Matrix, index = ['True Negative', 'True Positive'], columns =
['Predicted Negative', 'Predicted Postive'])
display(LR_Confusion_Matrix_df)

KNN_Confusion_Matrix = confusion_matrix(y_test, KNN_predictions)
KNN_Confusion_Matrix_df = pd.DataFrame(KNN_Confusion_Matrix, index = ['True Negative', 'True Positive'], columns =
['Predicted Negative', 'Predicted Positive'])
display(KNN_Confusion_Matrix_df)

Most_Frequent_Dummy_Model = DummyClassifier(strategy = "most_frequent")
Most_Frequent_Dummy_Model.fit(x_train, y_train)

Most_Frequent_Dummy_Predictions = Most_Frequent_Dummy_Model.predict(x_test)

Most_Frequent_Dummy_Confusion_Matrix = confusion_matrix(y_test, Most_Frequent_Dummy_Predictions)
Most_Frequent_Dummy_Confusion_Matrix_df = pd.DataFrame(Most_Frequent_Dummy_Confusion_Matrix, index = ['True
Negative', 'True Positive'], columns = ['Predicted Negative', 'Predicted Positive'])
display(Most_Frequent_Dummy_Confusion_Matrix_df)

Random_Dummy_Model = DummyClassifier(strategy = "uniform")
Random_Dummy_Model.fit(x_train, y_train)

Random_Dummy_Predictions = Random_Dummy_Model.predict(x_test)

Random_Dummy_Confusion_Matrix = confusion_matrix(y_test, Random_Dummy_Predictions)
```

```
Random_Dummy_Confusion_Matrix_df = pd.DataFrame(Random_Dummy_Confusion_Matrix, index = ['True Negative', 'True Positive'], columns = ['Predicted Negative', 'Predicted Positive'])
display(Random_Dummy_Confusion_Matrix_df)

Logistic_Regression_Score = Logistic_Regression_Model_Final.decision_function(x_test_poly_features)
LR_FPR, LR_TPR, LR_Threshold = roc_curve(y_test, Logistic_Regression_Score)

KNN_Score = KNN_Model_Final.predict_proba(x_test)
KNN_FPR, KNN_TPR, KNN_Threshold = roc_curve(y_test, KNN_Score[:,1])

Most_Frequent_Score = Most_Frequent_Dummy_Model.predict_proba(x_test)
MF_FPR, MF_TPR, MF_Threshold = roc_curve(y_test, Most_Frequent_Score[:,1])

Random_Dummy_Score = Random_Dummy_Model.predict_proba(x_test)
RD_FPR, RD_TPR, RD_Threshold = roc_curve(y_test, Random_Dummy_Score[:,1])

plot.figure()

plot.plot(LR_FPR, LR_TPR, label = 'Logistic Regression', color = 'blue', linewidth=3)
plot.plot(KNN_FPR, KNN_TPR, label = 'KNN Classifier', color = 'orange', linewidth=3)
plot.plot(MF_FPR, MF_TPR, label = 'Most Frequent Dummy Classifier', color = 'green', alpha = 0.9, linewidth=5)
plot.plot(RD_FPR, RD_TPR, label = 'Random Dummy Classifier', color = 'yellow', linewidth=3)
plot.plot([0, 1], [0, 1], color='grey', linestyle='--', linewidth=1)
plot.ylabel("True positive rate")
plot.xlabel("False positive rate")
plot.title('ROC Curve for Logistic Regression, KNN, Most Frequent Dummy Classifier and Random Classfier')
plot.legend(['Logistic Regression', 'KNN Classifier', 'Most Frequent Dummy Classifier', 'Random Dummy Classifier', 'Baseline'], title="Legends", loc = 'lower right', bbox_to_anchor=(1.63, 0.55))
plot.show()
```

Name: Karan Dua
Student Id: 21331391

Appendix: Code for Question 2 - All Parts

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plot
from IPython.display import display
import random


from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.dummy import DummyClassifier
from sklearn.metrics import roc_curve

df = pd.read_csv("data_week4_2.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]

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:25-50-25-0')
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()

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]})
display(actual_df)

poly_degree_range = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
penalty_parameter_range = [0.001, 1, 10, 1000]

def calculate_accuracy_stddev_for_given_penalty(penalty_parameter) :
  k_fold_split = 5

  k_fold_split_function =  KFold(n_splits = k_fold_split)

  accuracy_poly_degree = []
  standard_deviation_poly_degree = []
  Logistic_Regression_Model = LogisticRegression(penalty = 'l2', C = penalty_parameter, max_iter = 10000)

  for poly_degree in poly_degree_range :
    poly_features_function = PolynomialFeatures(poly_degree)
    x_poly_features  = poly_features_function.fit_transform(x)
    accuaracy_fold = []
    for train_data_index, test_data_index in k_fold_split_function.split(x_poly_features):
      Logistic_Regression_Model.fit(x_poly_features[train_data_index], y[train_data_index])
      predictions = Logistic_Regression_Model.predict(x_poly_features[test_data_index])
```

```
        accuaracy_fold.append(accuracy_score(y[test_data_index], predictions))

      accuracy_poly_degree.append(np.array(accuaracy_fold).mean())
      standard_deviation_poly_degree.append(np.array(accuaracy_fold).std())

   return accuracy_poly_degree, standard_deviation_poly_degree

for penalty_parameter in penalty_parameter_range :

   accuracy_poly_degree,                                        standard_deviation_poly_degree                                        =
calculate_accuracy_stddev_for_given_penalty(penalty_parameter)
   plot.figure()
   plot.errorbar(poly_degree_range, accuracy_poly_degree, yerr = standard_deviation_poly_degree, color = 'blue')
   plot.xlabel('Degree of Polynomial generated')
   plot.ylabel('Accuracy')
   plot.title('Plot of Accuracy VS Degree of Polynomial for Logistic Regression with Hyper-Penaltly Parameter
{}'.format(penalty_parameter))
   plot.show()


def calculate_accuracy_stddev_for_penalty_ranges(penalty_parameter_ranges, poly_degree) :

   k_fold_split = 5

   k_fold_split_function =  KFold(n_splits = k_fold_split)

   accuracy_penalty = []
   standard_deviation_penalty = []

   poly_features_function = PolynomialFeatures(poly_degree)
   x_poly_features  = poly_features_function.fit_transform(x)

   for penalty_parameter in penalty_parameter_ranges :

      Logistic_Regression_Model = LogisticRegression(penalty = 'l2', C = penalty_parameter, max_iter = 10000)
      accuaracy_fold = []
      for train_data_index, test_data_index in k_fold_split_function.split(x_poly_features):
         Logistic_Regression_Model.fit(x_poly_features[train_data_index], y[train_data_index])
         predictions = Logistic_Regression_Model.predict(x_poly_features[test_data_index])
         accuaracy_fold.append(accuracy_score(y[test_data_index], predictions))

      accuracy_penalty.append(np.array(accuaracy_fold).mean())
      standard_deviation_penalty.append(np.array(accuaracy_fold).std())

   return accuracy_penalty, standard_deviation_penalty

penalty_parameters = [1, 5, 10, 100, 500, 1000]
poly_degree = 1
accuracy_penalty,    standard_deviation_penalty    =    calculate_accuracy_stddev_for_penalty_ranges(penalty_parameters,
poly_degree)

plot.figure()
plot.errorbar(penalty_parameters, accuracy_penalty, yerr = standard_deviation_penalty, color = 'blue')
plot.xlabel('Penalty')
plot.ylabel('Accuracy')
plot.title('Plot of Accuracy VS Penalty for 5-fold Cross Validation Logistic Regression')
plot.show()

penalty_parameters = [1, 5, 10, 50, 75, 100]
```

```
poly_degree = 1
accuracy_penalty, standard_deviation_penalty = calculate_accuracy_stddev_for_penalty_ranges(penalty_parameters, poly_degree)

plot.figure()
plot.errorbar(penalty_parameters, accuracy_penalty, yerr = standard_deviation_penalty, color = 'blue')
plot.xlabel('Penalty')
plot.ylabel('Accuracy')
plot.title('Plot of Accuracy VS Penalty for 5-fold Cross Validation Logistic Regression')
plot.show()

penalty_parameters = [1, 3, 5, 7, 9, 10]
poly_degree = 1
accuracy_penalty, standard_deviation_penalty = calculate_accuracy_stddev_for_penalty_ranges(penalty_parameters, poly_degree)

plot.figure()
plot.errorbar(penalty_parameters, accuracy_penalty, yerr = standard_deviation_penalty, color = 'blue')
plot.xlabel('Penalty')
plot.ylabel('Accuracy')
plot.title('Plot of Accuracy VS Penalty for 5-fold Cross Validation Logistic Regression')
plot.show()

penalty_parameters = [0.001, 0.01, 0.1, 0.5, 1]
poly_degree = 1
accuracy_penalty, standard_deviation_penalty = calculate_accuracy_stddev_for_penalty_ranges(penalty_parameters, poly_degree)

plot.figure()
plot.errorbar(penalty_parameters, accuracy_penalty, yerr = standard_deviation_penalty, color = 'blue')
plot.xlabel('Penalty')
plot.ylabel('Accuracy')
plot.title('Plot of Accuracy VS Penalty for 5-fold Cross Validation Logistic Regression')
plot.show()

def calculate_accuracy_stddev_for_K_ranges(K_range) :

  k_fold_split = 5

  k_fold_split_function =  KFold(n_splits = k_fold_split)

  accuracy_k = []
  standard_deviation_k = []

  for k in K_range :

    KNN_Model = KNeighborsClassifier(n_neighbors = k, weights = 'uniform')
    accuaracy_fold = []
    for train_data_index, test_data_index in k_fold_split_function.split(x):
      KNN_Model.fit(x[train_data_index], y[train_data_index])
      predictions = KNN_Model.predict(x[test_data_index])
      accuaracy_fold.append(accuracy_score(y[test_data_index], predictions))

    accuracy_k.append(np.array(accuaracy_fold).mean())
    standard_deviation_k.append(np.array(accuaracy_fold).std())

  return accuracy_k, standard_deviation_k
```

```
K_range = [1, 5, 10, 100, 300, 500]
accuracy_k, standard_deviation_k = calculate_accuracy_stddev_for_K_ranges(K_range)

plot.figure()
plot.errorbar(K_range, accuracy_k, yerr = standard_deviation_k, color = 'blue')
plot.xlabel('K Neighbors')
plot.ylabel('Accuracy')
plot.title('Plot of Accuracy VS K value for KNN using 5-fold Cross Validation')
plot.show()

K_range = [1, 5, 10, 50, 75, 100]
accuracy_k, standard_deviation_k = calculate_accuracy_stddev_for_K_ranges(K_range)

plot.figure()
plot.errorbar(K_range, accuracy_k, yerr = standard_deviation_k, color = 'blue')
plot.xlabel('K Neighbors')
plot.ylabel('Accuracy')
plot.title('Plot of Accuracy VS K value for KNN using 5-fold Cross Validation')
plot.show()

K_range = [1, 5, 10, 20, 25, 30, 40, 50, 60, 70]
accuracy_k, standard_deviation_k = calculate_accuracy_stddev_for_K_ranges(K_range)

plot.figure()
plot.errorbar(K_range, accuracy_k, yerr = standard_deviation_k, color = 'blue')
plot.xlabel('K Neighbors')
plot.ylabel('Accuracy')
plot.title('Plot of Accuracy VS K value for KNN using 5-fold Cross Validation')
plot.show()

k_optimised = 25
poly_degree_range = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

k_fold_split = 5

k_fold_split_function =  KFold(n_splits = k_fold_split)

accuracy_poly_degree = []
standard_deviation_poly_degree = []
KNN_Model = KNeighborsClassifier(n_neighbors = k_optimised, weights = 'uniform')

for poly_degree in poly_degree_range :
    poly_features_function = PolynomialFeatures(poly_degree)
    x_poly_features  = poly_features_function.fit_transform(x)
    accuaracy_fold = []
    for train_data_index, test_data_index in k_fold_split_function.split(x_poly_features):
        KNN_Model.fit(x_poly_features[train_data_index], y[train_data_index])
        predictions = KNN_Model.predict(x_poly_features[test_data_index])
        accuaracy_fold.append(accuracy_score(y[test_data_index], predictions))

    accuracy_poly_degree.append(np.array(accuaracy_fold).mean())
    standard_deviation_poly_degree.append(np.array(accuaracy_fold).std())

plot.figure()
plot.errorbar(poly_degree_range, accuracy_poly_degree, yerr = standard_deviation_poly_degree, color = 'blue')
plot.xlabel('Degree of Polynomial generated')
plot.ylabel('Accuracy')
plot.title('Plot of Accuracy VS Degree of Polynomial for KNN with K = 50')
plot.show()
```

```
random.seed(123)

x_train, x_test, y_train, y_test = train_test_split(x, y, train_size = 0.8)


temp_df = pd.DataFrame(x, columns = ['x1', 'x2'])
feature_names = poly_features_function.get_feature_names(temp_df.columns)
feature_names.insert(0, 'Intercept')

penalty_parameter = 1
poly_degree = 1

poly_features_function = PolynomialFeatures(poly_degree)
x_train_poly_features  = poly_features_function.fit_transform(x_train)
x_test_poly_features  = poly_features_function.fit_transform(x_test)

temp_df = pd.DataFrame(x, columns = ['x1', 'x2'])
feature_names = poly_features_function.get_feature_names(temp_df.columns)
feature_names.insert(0, 'Intercept')

Logistic_Regression_Model_Final = LogisticRegression(penalty = 'l2', C = penalty_parameter, max_iter = 10000)
Logistic_Regression_Model_Final.fit(x_train_poly_features, y_train)


logistic_regression_params_df = pd.DataFrame(columns = feature_names)
model_dict = {}
model_dict['Intercept'] = [np.around(Logistic_Regression_Model_Final.intercept_[0], decimals = 3)]
for i in range(1, 4) :
    model_dict[feature_names[i]] = [np.around(Logistic_Regression_Model_Final.coef_[0][i-1], decimals = 3)]

logistic_regression_params_df = logistic_regression_params_df.append(model_dict, ignore_index = True)
logistic_regression_params_df = logistic_regression_params_df.style.applymap(lambda x:'white-space:nowrap')
display(logistic_regression_params_df)


logistic_regression_model_equation_df = pd.DataFrame(columns = ['Model Equation'])
model_eq_dict = {}
equation_string = ''
for i in range(1, 4) :
    coeff = np.around(Logistic_Regression_Model_Final.coef_[0][i-1], decimals = 3)
    if coeff != 0 :
        equation_string += '(' + str(coeff) + ')' + ' * ' + feature_names[i] + ' + '

equation_string += '(' + str(np.around(Logistic_Regression_Model_Final.intercept_[0], decimals = 3)) + ')'
model_eq_dict['Model Equation'] = equation_string

logistic_regression_model_equation_df = logistic_regression_model_equation_df.append(model_eq_dict, ignore_index = True)
logistic_regression_model_equation_df = logistic_regression_model_equation_df.style.set_properties(**{'text-align': 'left'})
logistic_regression_model_equation_df = logistic_regression_model_equation_df.applymap(lambda x:'white-space:nowrap')
display(logistic_regression_model_equation_df)


LR_Predictions = Logistic_Regression_Model_Final.predict(x_test_poly_features)

test_temp_df = pd.DataFrame(x_test, columns = ['x1','x2'])
x1_test = test_temp_df[['x1']]
x2_test = test_temp_df[['x2']]
```

```
plot.scatter(x1_test[np.array(y_test) == -1], x2_test[np.array(y_test) == -1], color='red', marker="o", edgecolors="black")
plot.scatter(x1_test[np.array(y_test) == 1], x2_test[np.array(y_test) == 1], color='yellow', marker="o", edgecolors="black")
plot.scatter(x1_test[LR_Predictions == -1], x2_test[LR_Predictions == -1], color='blue', marker="+", alpha=0.85)
plot.scatter(x1_test[LR_Predictions == 1], x2_test[LR_Predictions == 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=(1.45, 0.53))
plot.show()


k_optimised = 25

KNN_Model_Final = KNeighborsClassifier(n_neighbors = k_optimised, weights = 'uniform')

KNN_Model_Final.fit(x_train, y_train)
KNN_predictions = KNN_Model_Final.predict(x_test)

test_temp_df = pd.DataFrame(x_test, columns = ['x1','x2'])
x1_test = test_temp_df[['x1']]
x2_test = test_temp_df[['x2']]


plot.scatter(x1_test[np.array(y_test) == -1], x2_test[np.array(y_test) == -1], color='red', marker="o", edgecolors="black")
plot.scatter(x1_test[np.array(y_test) == 1], x2_test[np.array(y_test) == 1], color='yellow', marker="o", edgecolors="black")
plot.scatter(x1_test[KNN_predictions == -1], x2_test[KNN_predictions == -1], color='blue', marker="+", alpha=0.85)
plot.scatter(x1_test[KNN_predictions == 1], x2_test[KNN_predictions == 1], color='green', marker="+", alpha=0.85)
plot.title('Data Plot for Actual VS Predicted value of y for Features x1 and x2 for KNN')
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=(1.45, 0.53))
plot.show()

y_test_postive = np.count_nonzero(y_test == 1)
y_test_negative = np.count_nonzero(y_test == -1)

actual_test_df = pd.DataFrame({"Test Data Actual Positive":[y_test_postive], "Test Data Actual Negative":[y_test_negative]})
display(actual_test_df)

LR_true_positive = 0
LR_true_negative = 0
LR_false_positive = 0
LR_false_negative = 0
for i in range(len(y_test)) :
    if np.array(y_test)[i] == 1 and LR_Predictions[i] == 1 :
        LR_true_positive += 1
    elif np.array(y_test)[i] == -1 and LR_Predictions[i] == -1 :
        LR_true_negative += 1
    elif np.array(y_test)[i] == 1 and LR_Predictions[i] == -1 :
        LR_false_negative += 1
    elif np.array(y_test)[i] == -1 and LR_Predictions[i] == 1 :
        LR_false_positive += 1

LR_Accuracy = ((LR_true_positive + LR_true_negative) / (LR_true_positive + LR_true_negative + LR_false_positive +
LR_false_negative)) * 100
```

```
LR_y_pos_1_correctly_predicted = (LR_true_positive/y_test_postive)*100

LR_y_neg_1_correctly_predicted = (LR_true_negative/y_test_negative)*100

LR_True_Positive_Rate = (LR_true_positive) / (LR_true_positive + LR_false_negative)
LR_False_Positive_Rate = (LR_false_positive) / (LR_false_positive + LR_true_negative)
LR_Precision = (LR_true_positive) / (LR_true_positive + LR_false_positive)

LR_actual_predicted_df = pd.DataFrame({"True Positive":[LR_true_positive], "True Negative":[LR_true_negative], "False Positive":[LR_false_positive], "False Negative":[LR_false_negative], "Accuracy":[LR_Accuracy], "True Positive Rate":[LR_True_Positive_Rate], "False Positive Rate":[LR_False_Positive_Rate], "Precision":[LR_Precision]})
display(LR_actual_predicted_df)


KNN_true_positive = 0
KNN_true_negative = 0
KNN_false_positive = 0
KNN_false_negative = 0
for i in range(len(y_test)) :
    if np.array(y_test)[i] == 1 and KNN_predictions[i] == 1 :
        KNN_true_positive += 1
    elif np.array(y_test)[i] == -1 and KNN_predictions[i] == -1 :
        KNN_true_negative += 1
    elif np.array(y_test)[i] == 1 and KNN_predictions[i] == -1 :
        KNN_false_negative += 1
    elif np.array(y_test)[i] == -1 and KNN_predictions[i] == 1 :
        KNN_false_positive += 1

KNN_Accuracy = ((KNN_true_positive + KNN_true_negative) / (KNN_true_positive + KNN_true_negative + KNN_false_positive + KNN_false_negative)) * 100

KNN_y_pos_1_correctly_predicted = (KNN_true_positive/y_test_postive)*100

KNN_y_neg_1_correctly_predicted = (KNN_true_negative/y_test_negative)*100

KNN_True_Positive_Rate = (KNN_true_positive) / (KNN_true_positive + KNN_false_negative)
KNN_False_Positive_Rate = (KNN_false_positive) / (KNN_false_positive + KNN_true_negative)
KNN_Precision = (KNN_true_positive) / (KNN_true_positive + KNN_false_positive)

KNN_actual_predicted_df = pd.DataFrame({"True Positive":[KNN_true_positive], "True Negative":[KNN_true_negative], "False Positive":[KNN_false_positive], "False Negative":[KNN_false_negative], "Accuracy":[KNN_Accuracy], "True Positive Rate":[KNN_True_Positive_Rate], "False Positive Rate":[KNN_False_Positive_Rate], "Precision":[KNN_Precision]})
display(KNN_actual_predicted_df)

LR_Confusion_Matrix = confusion_matrix(y_test, LR_Predictions)
LR_Confusion_Matrix_df = pd.DataFrame(LR_Confusion_Matrix, index = ['True Negative', 'True Positive'], columns = ['Predicted Negative', 'Predicted Positive'])
display(LR_Confusion_Matrix_df)

KNN_Confusion_Matrix = confusion_matrix(y_test, KNN_predictions)
KNN_Confusion_Matrix_df = pd.DataFrame(KNN_Confusion_Matrix, index = ['True Negative', 'True Positive'], columns = ['Predicted Negative', 'Predicted Positive'])
display(KNN_Confusion_Matrix_df)

Most_Frequent_Dummy_Model = DummyClassifier(strategy = "most_frequent")
Most_Frequent_Dummy_Model.fit(x_train, y_train)

Most_Frequent_Dummy_Predictions = Most_Frequent_Dummy_Model.predict(x_test)
```

```
Most_Frequent_Dummy_Confusion_Matrix = confusion_matrix(y_test, Most_Frequent_Dummy_Predictions)
Most_Frequent_Dummy_Confusion_Matrix_df = pd.DataFrame(Most_Frequent_Dummy_Confusion_Matrix, index = ['True Negative', 'True Positive'], columns = ['Predicted Negative', 'Predicted Positive'])
display(Most_Frequent_Dummy_Confusion_Matrix_df)

Random_Dummy_Model = DummyClassifier(strategy = "uniform")
Random_Dummy_Model.fit(x_train, y_train)

Random_Dummy_Predictions = Random_Dummy_Model.predict(x_test)

Random_Dummy_Confusion_Matrix = confusion_matrix(y_test, Random_Dummy_Predictions)
Random_Dummy_Confusion_Matrix_df = pd.DataFrame(Random_Dummy_Confusion_Matrix, index = ['True Negative', 'True Positive'], columns = ['Predicted Negative', 'Predicted Positive'])
display(Random_Dummy_Confusion_Matrix_df)

Logistic_Regression_Score = Logistic_Regression_Model_Final.decision_function(x_test_poly_features)
LR_FPR, LR_TPR, LR_Threshold = roc_curve(y_test, Logistic_Regression_Score)

KNN_Score = KNN_Model_Final.predict_proba(x_test)
KNN_FPR, KNN_TPR, KNN_Threshold = roc_curve(y_test, KNN_Score[:,1])

Most_Frequent_Score = Most_Frequent_Dummy_Model.predict_proba(x_test)
MF_FPR, MF_TPR, MF_Threshold = roc_curve(y_test, Most_Frequent_Score[:,1])

Random_Dummy_Score = Random_Dummy_Model.predict_proba(x_test)
RD_FPR, RD_TPR, RD_Threshold = roc_curve(y_test, Random_Dummy_Score[:,1])

plot.figure()

plot.plot(LR_FPR, LR_TPR, label = 'Logistic Regression', color = 'blue', linewidth=3)
plot.plot(KNN_FPR, KNN_TPR, label = 'KNN Classifier', color = 'orange', linewidth=3)
plot.plot(MF_FPR, MF_TPR, label = 'Most Frequent Dummy Classifier', color = 'green', alpha = 0.9, linewidth=5)
plot.plot(RD_FPR, RD_TPR, label = 'Random Dummy Classifier', color = 'yellow', linewidth=3)
plot.plot([0, 1], [0, 1], color='grey', linestyle='--', linewidth=1)
plot.ylabel("True positive rate")
plot.xlabel("False positive rate")
plot.title('ROC Curve for Logistic Regression, KNN, Most Frequent Dummy Classifier and Random Classfier')
plot.legend(['Logistic Regression', 'KNN Classifier', 'Most Frequent Dummy Classifier', 'Random Dummy Classifier', 'Baseline'], title="Legends", loc = 'lower right', bbox_to_anchor=(1.63, 0.55))
plot.show()
```