Question-1 (Notebook-1)

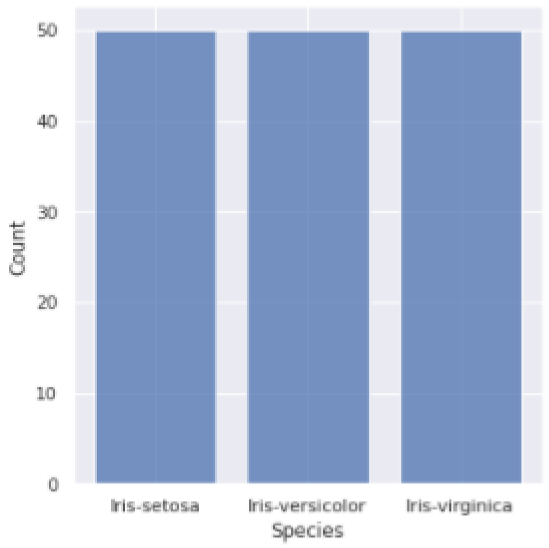
1. **Preprocessing the data. :**

a.) Plot the distribution of the target variable.

Given the iris dataset , we have to plot the distribution of the target variable . Target variable includes the flower categories . The target distribution will give us an idea about the count of each flower category

```
sns.displot(iris, x="Species", shrink=.8)
```

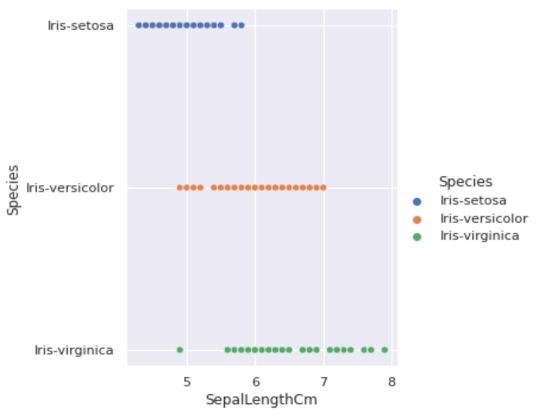
Shrink scale the width of each bar relative to the binwidth by 0.8 factor.

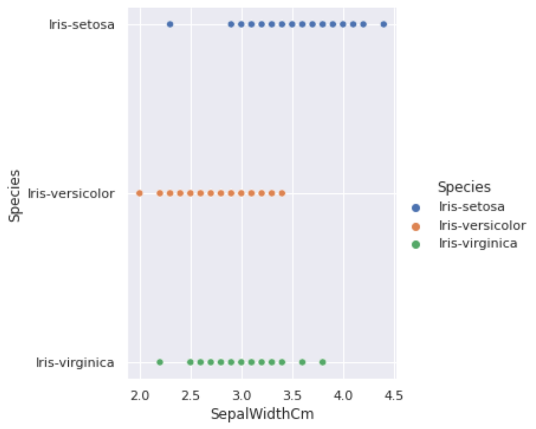


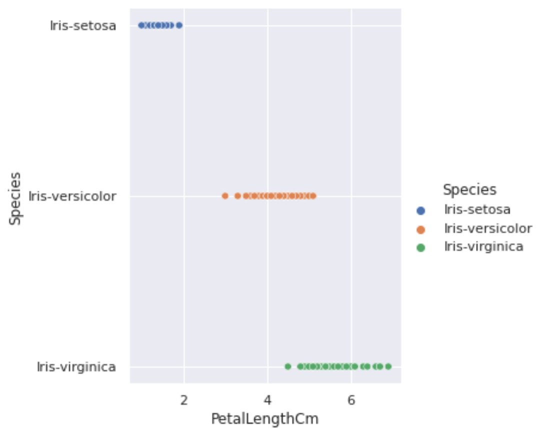
b.) Visualize the distribution of data for every feature.

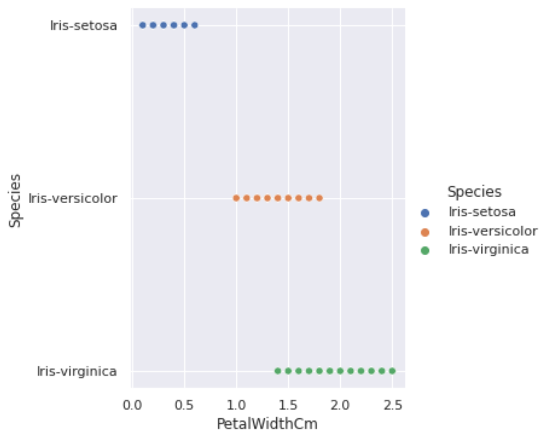
There are 4 features . The following plots show the distribution of classes i.e target categories for every feature.

```
sns.relplot(x=features[0], y="Species", hue="Species", data=iris);

sns.relplot(x=features[1], y="Species", hue="Species", data=iris);

sns.relplot(x=features[2], y="Species", hue="Species", data=iris);

sns.relplot(x=features[3], y="Species", hue="Species", data=iris);
```









We also encoded target categories into numerical values using label encoder

```
le = LabelEncoder()

iris['Species'] = le.fit_transform(iris['Species'])
```

2. **Perform boosting-based classification using Decision Tree as the base classifier.**

Before classification , the data is splitted into testing and training data using sklearn's train_test split function

```
X_train,   X_test,   y_train,   y_test   =   train_test_split(   X,   y,
test_size=0.2)
```

80% is training data and rest of 20% is testing data

The below code fits a decision tree classifier on train data

```
dt_clf = dt_clf.fit(X_train,y_train)
```

3. **Perform cross validation over the data and calculate accuracy for a weak learner.**

   We use sklearn's cv_scores function to find cross validation score over the data for weak decision tree classifier fitted over the data .

   ```
   scores = cross_val_score(dt_clf, X, y, cv=5)
   ```

   We observed that some of the cross validation scores are 1 which is due to the fact that decision trees are prone to overfitting.

   Accuracy of weak learner i.e decision tree in our case, can be calculated using the below code

   ```
   score_dt = dt_clf.score(X_test, y_test)
   ```

4. **Build the AdaBoost model using the weak learner by increasing the number of trees from 1 to 5 with a step of 1. Compute the model performance.**

   We have to build Adaboost model using decision tree as the weak classifier with changing the number of trees everytime. This can be easily done using a for loop

   ```
   estimators = list(range(1, 5, 1))

   ada_scores = []

   for n_est in estimators:

    ada_clf = AdaBoostClassifier(dt_clf, n_estimators=n_est)
   ```

```
 ada_clf.fit(X_train, y_train)

 ada_score = ada_clf.score(X_test, y_test)

 ada_scores.append(ada_score)

ada_scores
```

The above code fits the adaboost model with decision tree as weak classifier and with changing the n_estimators parameter in each iteration. Ada_scores is a list which stores the accuracy for each such classifier.

We used grid search to find the best value of n_estimator for this adaboost model using the below code.

```
scorer = metrics.make_scorer(metrics.f1_score, average = 'weighted')

parameters = {'n_estimators':[2,3,4]}

grid_search  =  GridSearchCV(ada_clf,  parameters,  scoring=scorer,
cv=5)

grid_search.fit(X_train, y_train)

print(grid_search.best_params_)

print(grid_search.best_score_)
```

Grid_search.best_score gives the best score for performance of the classifier

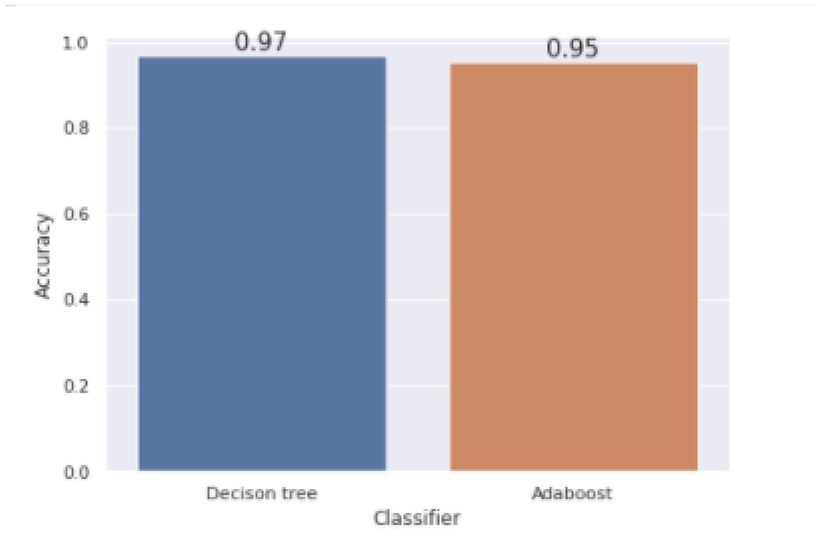
To compute the model performance , I used cv_scores for adaboost classifier.

```
ada_cv_scores      =      cross_val_score(AdaBoostClassifier(dt_clf,
n_estimators=2), X, y, cv=5)

ada_mean_score = ada_cv_scores.mean()
```

Ada_mean_score gives the mean of all scores present in ada_cv_scores

To compare the scores of both models , i.e simple decision classifier and adaboost model, we used the below code and plotted scores corresponding to each classifier

```
classifiers = ["Decison tree", "Adaboost"]

accuracies = [score_dt, ada_mean_score]

data = {"Classifier":classifiers ,

         "Accuracy": accuracies}

df = pd.DataFrame(data, columns=['Classifier', 'Accuracy'])

plots = sns.barplot(x="Classifier", y="Accuracy", data=df)
```



Question-2 (Bayes classification) (Notebook-1)

1. **Estimate the accuracy of Naive Bayes algorithm using 5-fold cross validation on the data set. Plot the ROC AUC curve for different values of parameters.**

We fit the naive bayes model on our training data .

```
clf_bayes = GaussianNB()

clf_bayes= clf_bayes.fit(X_train, y_train)
```

The accuracy of Naive bayes algorithm using 5 fold cross validation can be found using the below code

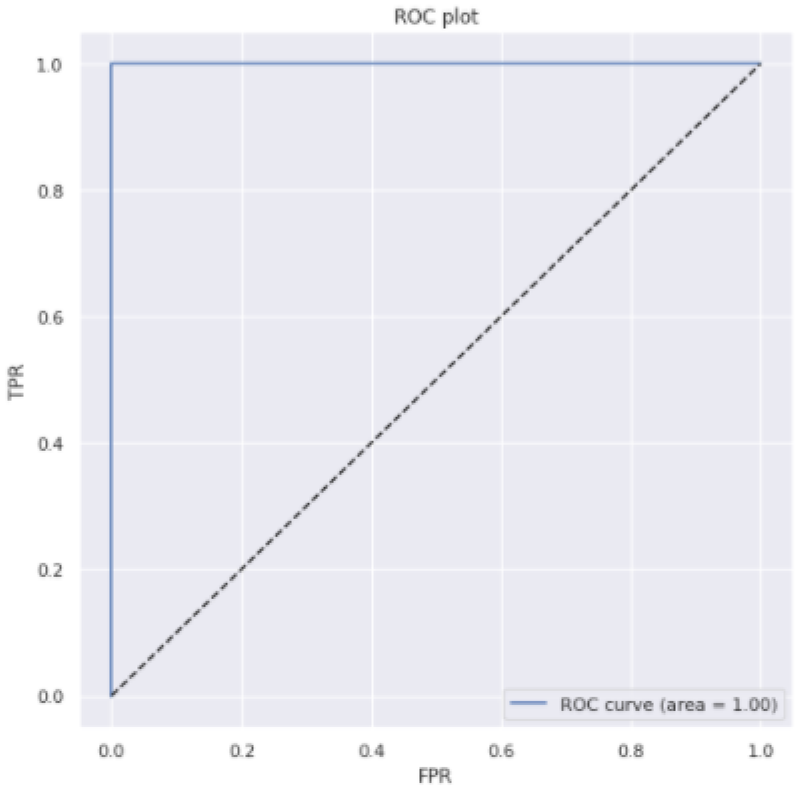
```
scores_bayes = cross_val_score(clf_bayes, X, y, cv=5)
```

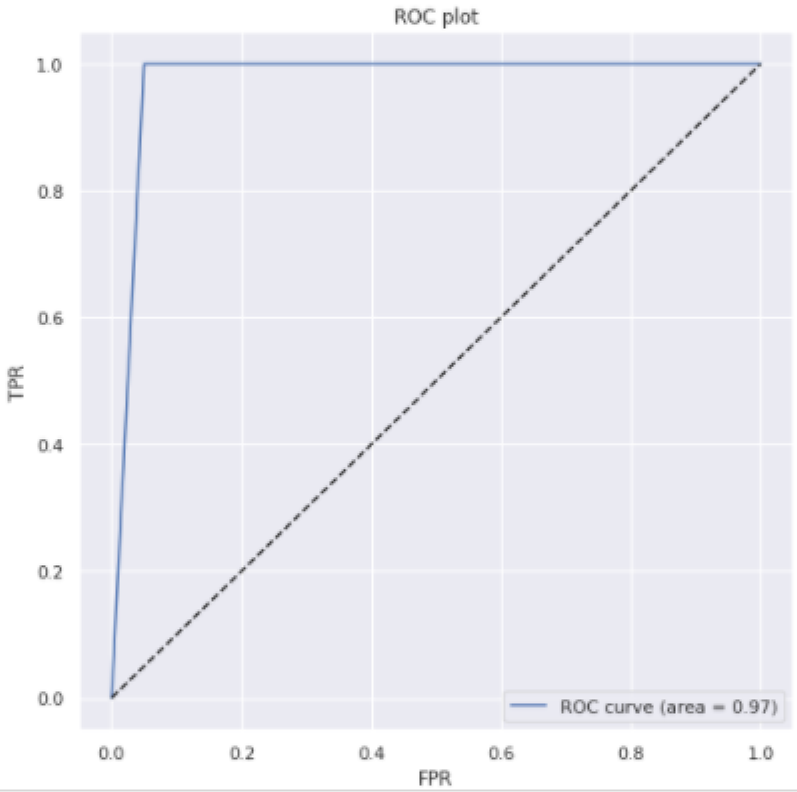
ROC curve is generally plotted for binary classification problems . But we can also plot ROC curves for multiclass classification problems also by taking one class as positive and rest of the other as negative . Here the target has 3 classes ,so 3 ROC plots will be plotted. Every time , one of the classes will be taken as positive and other two as negative .

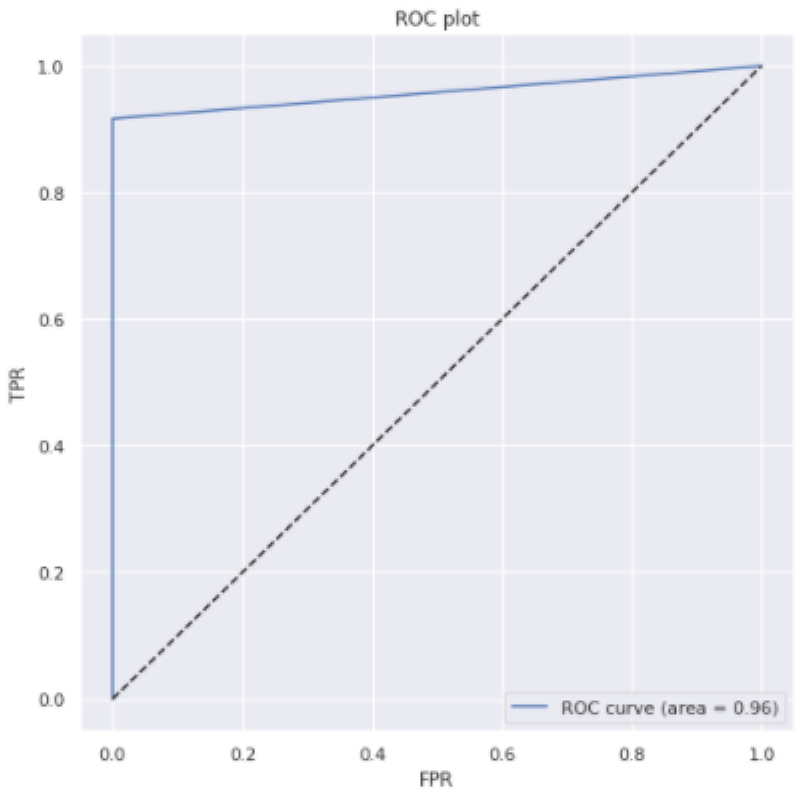
```
y = label_binarize(y, classes=[0,1,2])

n_classes = 3




clf = OneVsRestClassifier(clf_bayes)

y_score = clf.fit(X_train, y_train).predict(X_test)



fpr = dict()

tpr = dict()

roc_auc = dict()

for i in range(n_classes):
     fpr[i],tpr[i], _ = roc_curve(np.array(pd.get_dummies(y_test))[:,
i], np.array(pd.get_dummies(y_score))[:, i])

    roc_auc[i] = auc(fpr[i], tpr[i])



for i in range(n_classes):

    plt.figure()

      plt.plot(fpr[i],  tpr[i],  label='ROC  curve  (area  =  %0.2f)'  %
roc_auc[i])

    plt.plot([0, 1], [0, 1], 'k--')

    plt.xlabel('FPR')

    plt.ylabel('TPR')

    plt.title('ROC plot')

    plt.legend(loc="lower right")

    plt.show()
```

Label_binarize will binarize labels in a one-vs-all fashion.

We predict multi-class targets by using OneVsRestClassifier.


ROC plot
TPR
FPR
ROC curve (area = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0


ROC plot
TPR
FPR
ROC curve (area = 0.97)
0.0
0.2
0.4
0.6
0.8
1.0



The 1st ROC plot has area =1 which is the characteristic of an ideal ROC curve. This shows that the accuracy is 100% for this classification .

2. **Use linear discriminant function to calculate the accuracy on the classification task with 80% training and 20% testing data.**

   The linear discriminant function can be used to calculate the accuracy on the classification task using the below code

```
clf_lda = LinearDiscriminantAnalysis()

clf_lda.fit(X_train, y_train)

clf_lda.score(X_test, y_test)
```

3. **Calculate the Bayes risk**

Given: λ=

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

The conditional risk can be calculated as :

$$\begin{aligned} R(\alpha_1|\mathbf{x}) &= \lambda_{11}P(\omega_1|\mathbf{x}) + \lambda_{12}P(\omega_2|\mathbf{x}) \\ R(\alpha_2|\mathbf{x}) &= \lambda_{21}P(\omega_1|\mathbf{x}) + \lambda_{22}P(\omega_2|\mathbf{x}). \end{aligned}$$

Where R(aplha1 | x) is the risk in taking action 1 given the feature vector x. Similarly R(alpha2 | x) is the risk in taking action 2 given the feature vector x

To find risk we need to find posterior probabilities which can be found using the below code . The code gives a 30 x 3 matrix , where 30 is the size of our test data and 3 columns correspond to probability of each target variable / category ie. probabilities that the flower belongs to 'Iris-setosa', 'Iris-versicolor', 'Iris-virginica' respectively .

```
probab = clf_bayes.predict_proba(X_test)
```

To get risk corresponding to each sample for each class , we need to multiply the probability matrix and the transpose of given loss matrix as illustrated .

$$\begin{pmatrix} P(w1|x1) & P(w2|x1) & P(w3|x1) \\ P(w1|x2) & P(w2|x2) & P(w3|x2) \\ P(w1|x3) & P(w2|x3) & P(w3|x3) \\ & \vdots & \\ P(w1|x30) & P(w2|x30) & P(w3|x30) \end{pmatrix} \qquad \begin{pmatrix} \text{lambda } 11 & \text{lambda } 21 & \text{lambda } 31 \\ \text{lambda } 12 & \text{lambda } 22 & \text{lambda } 32 \\ \text{lambda } 13 & \text{lambda } 23 & \text{lambda } 33 \end{pmatrix}$$

This is done using the code below

```
loss = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

np.transpose(loss)

risk = np.matmul(probab , loss)
```

The resultant is a 30 x 3 matrix . For each sample(row), each column depicts the risk of classifying as class corresponding to the column for that particular feature

Question 3: Visualisation in Bayesian Decision Theory

DATASET 1: **(Notebook-2)**

Consider the height of the car and its cost is given. If the cost of a car > 550 then the label is 1, otherwise 0.

a. **Create the labels from the given data.**

Given the dataset , with features as “Height” and “Price”, we need to create the labels for the data. The condition given is that if the cost of a car > 550, the label assigned to the sample is 1 , otherwise it is 0

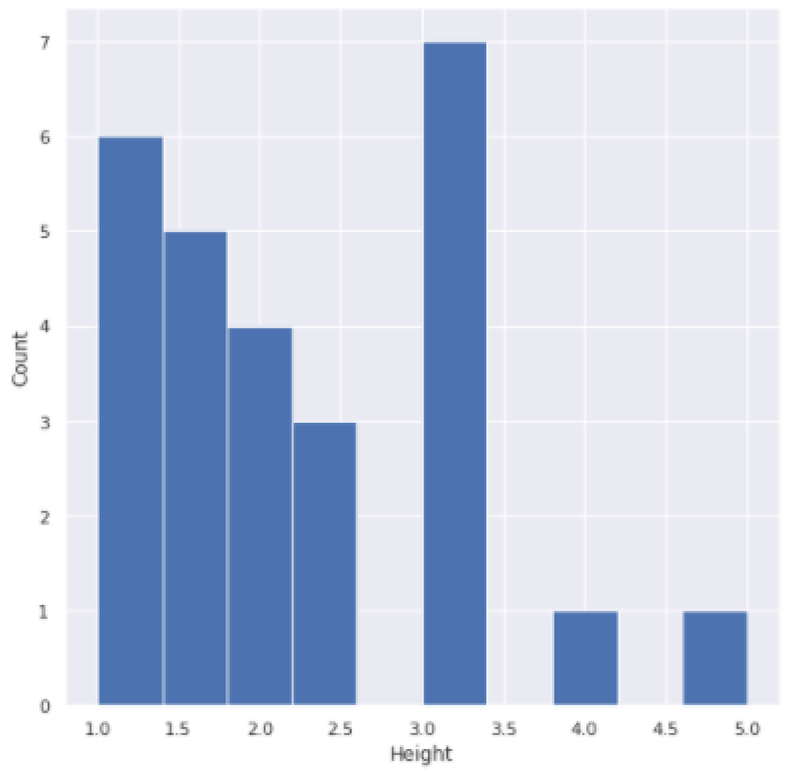
```
label = []

x_c1 = []

x_c2 = []

c1_count = 0

c2_count = 0

for i in range(data.shape[0]):

  if(data[i][1] > 550):

    label.append(1)

    x_c1.append(data[i][0])

    c1_count += 1

  else:

    label.append(0)

    x_c2.append(data[i][0])

    c2_count += 1
```

The above code give label to each sample in dataset . x_c1 consists of heights for which label is 1. x_c2 consists of heights for which label is 0. c1_count is the count of number of samples labelled as 1. Similarly c2_count is the count of number of samples labelled as 0.

b. Plot the distribution of samples using histogram.

The distribution of heights can be plotted using the code below

```
plt.hist(x)

plt.xlabel("Height")
plt.ylabel("Count")

plt.show()
```



The plot gives information about the count corresponding to a height.

c. **Determine the prior probability for both the classes.**

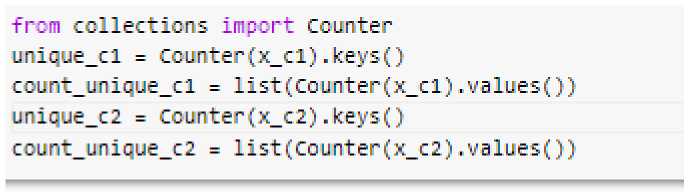
The prior probability of classes can be calculated by simply using the formula

p(class1) = number of samples in class 1 / total number of samples

```
prob_c1 = c1_count/no_of_samples

prob_c2 = c2_count/no_of_samples
```

The code above does the same thing

e. Plot the count of each unique element for each class

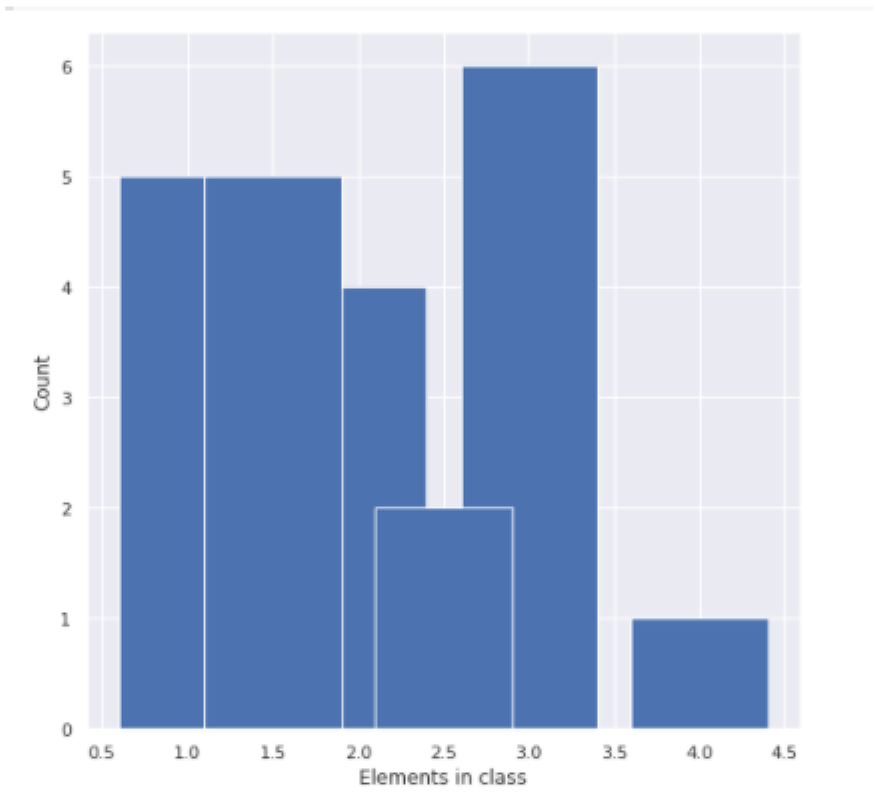


```
from collections import Counter
unique_c1 = Counter(x_c1).keys()
count_unique_c1 = list(Counter(x_c1).values())
unique_c2 = Counter(x_c2).keys()
count_unique_c2 = list(Counter(x_c2).values())
```

The above code is used to find the unique heights corresponding to each class . unique_c1 will consists of all unique heights for which label is 1 . count_unique_c1 will consists of count of each unique height present in unique_c1. Similarly unique_c2 and count_unique_c2 consists of unique heights for which label is 0 and the count of each unique height present in unique_c2 respectively

```
plt.bar(unique_c1,count_unique_c1)

plt.xlabel("Elements in class")

plt.ylabel("Count")

plt.show()
```

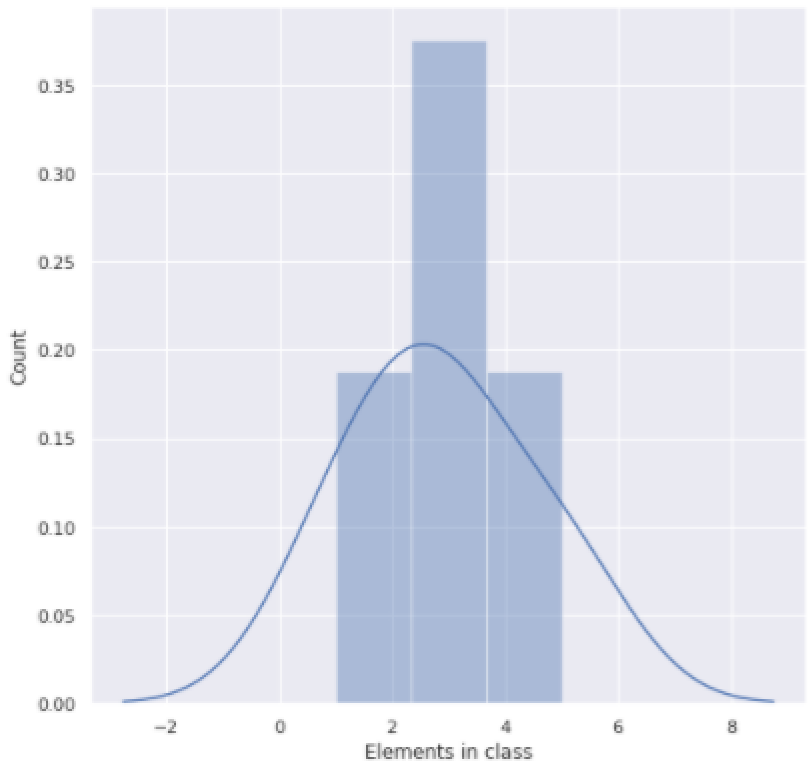
The above code plots the bar graph for height in c1 class



This plot is different from the distribution because of the reason that the heights and the count of heights in this plot is for those samples which belong to class c1 i.e for which label is 1

```
sns.distplot(x_c2)

plt.xlabel("Elements in class")

plt.ylabel("Count")

plt.show()
```

The above code will give another plot for class c2



The above graph gives information about the count of unique heights for class c2

d. Determine the likelihood / class conditional probabilities for the classes.

Likelihood or class conditional probabilities are basically p(x|c) which can be easily calculated using the below code

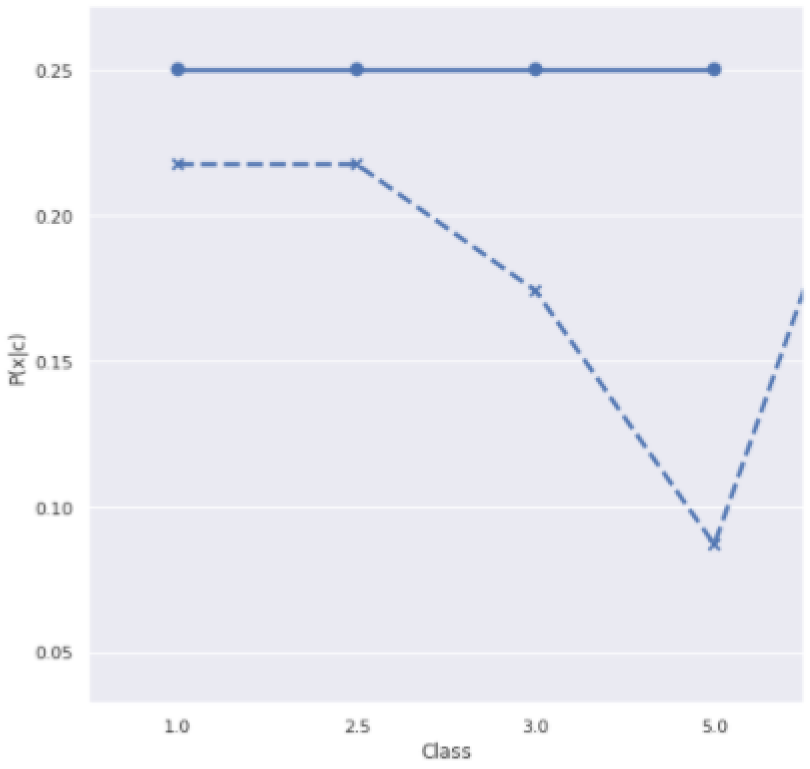
```
y1 = np.array(count_unique_c1)/len(x_c1)

y2 = np.array(count_unique_c2)/len(x_c2)
```

y1 is p (x | c1) and y2 is p(x | c2)

The plot of likelihood can be plotted using

```
sns.pointplot(list(unique_c1),y1, linestyles="--",markers =['x'] )

sns.pointplot(list(unique_c2),y2,linestyles='-',markers=['o'])
```

```
plt.xlabel("Class")

plt.ylabel("P(x|c)")

plt.show()
```



f. Calculate the P(C1|x) and P(C2|x) i.e posterior probabilities and plot them in a single graph.

p(c1 | x) = p(x | c1) p(c1) / p(x)

p(c2 | x) = p(x | c2) p(c2) / p(x)

```
posterior_c1 = []

i=0

for height in unique_c1:

  posterior_c1.append((likelihood_c1[i]* prob_c1)/ ( counts[height]
/df.shape[0]))
```

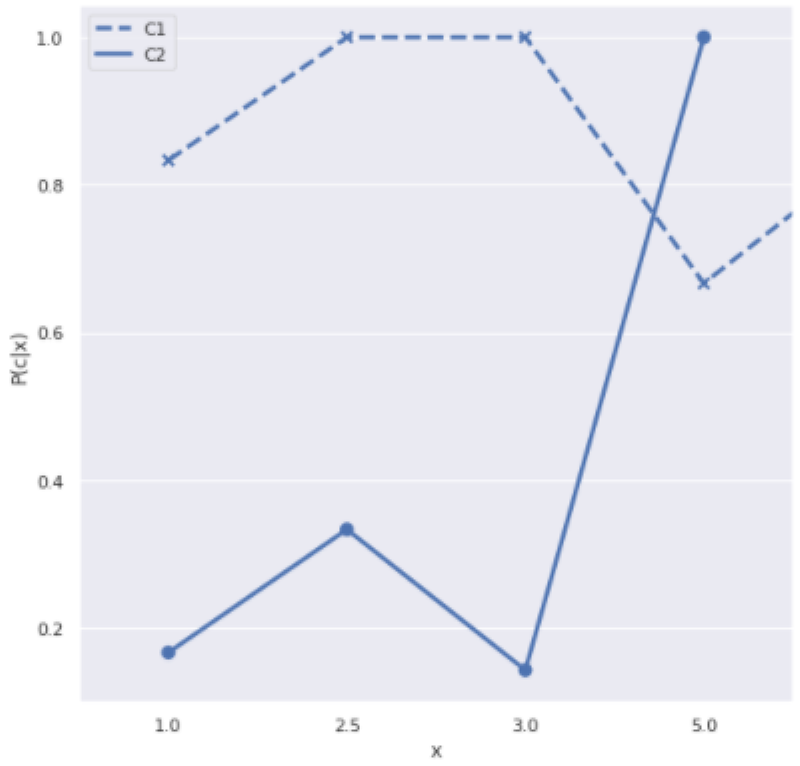
```
  i= i+1
```

The above code stores the posterior probability corresponding to different heights for c1 class in `posterior_c1` list

```
posterior_c2 = []

i=0

for height in unique_c2:

   posterior_c2.append((likelihood_c2[i]* prob_c2)/ ( counts[height]
/df.shape[0]))

  i= i+1
```

Similarily the above code stores posterior probabilities corresponding to different heights for c2 class in `posterior_c2` list.

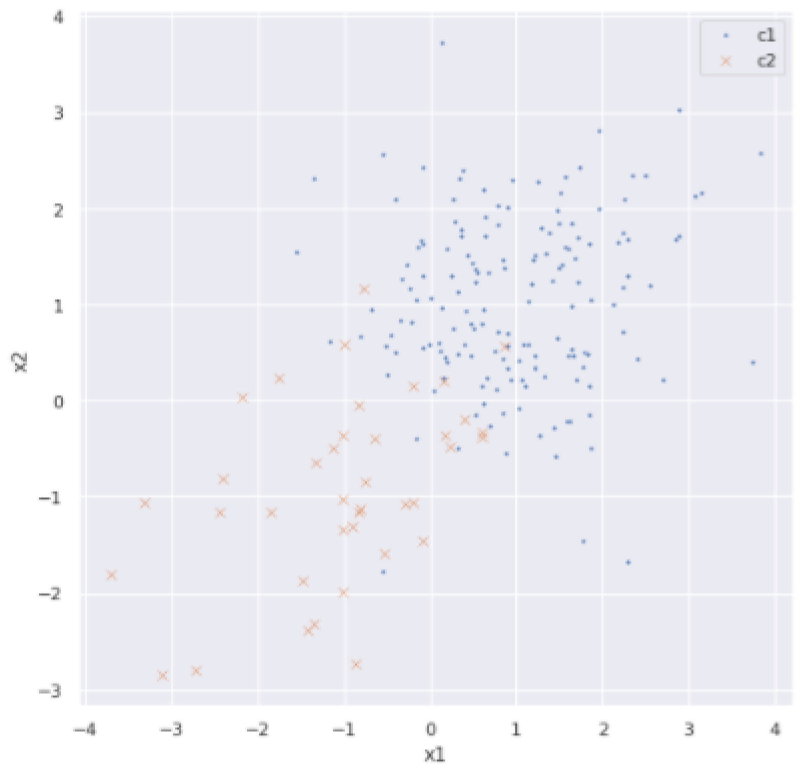
```
sns.pointplot(list(unique_c1), posterior_c1, linestyles="--",markers
=['x'] )

sns.pointplot(list(unique_c2),posterior_c2,linestyles='-',markers=['
o'])
```



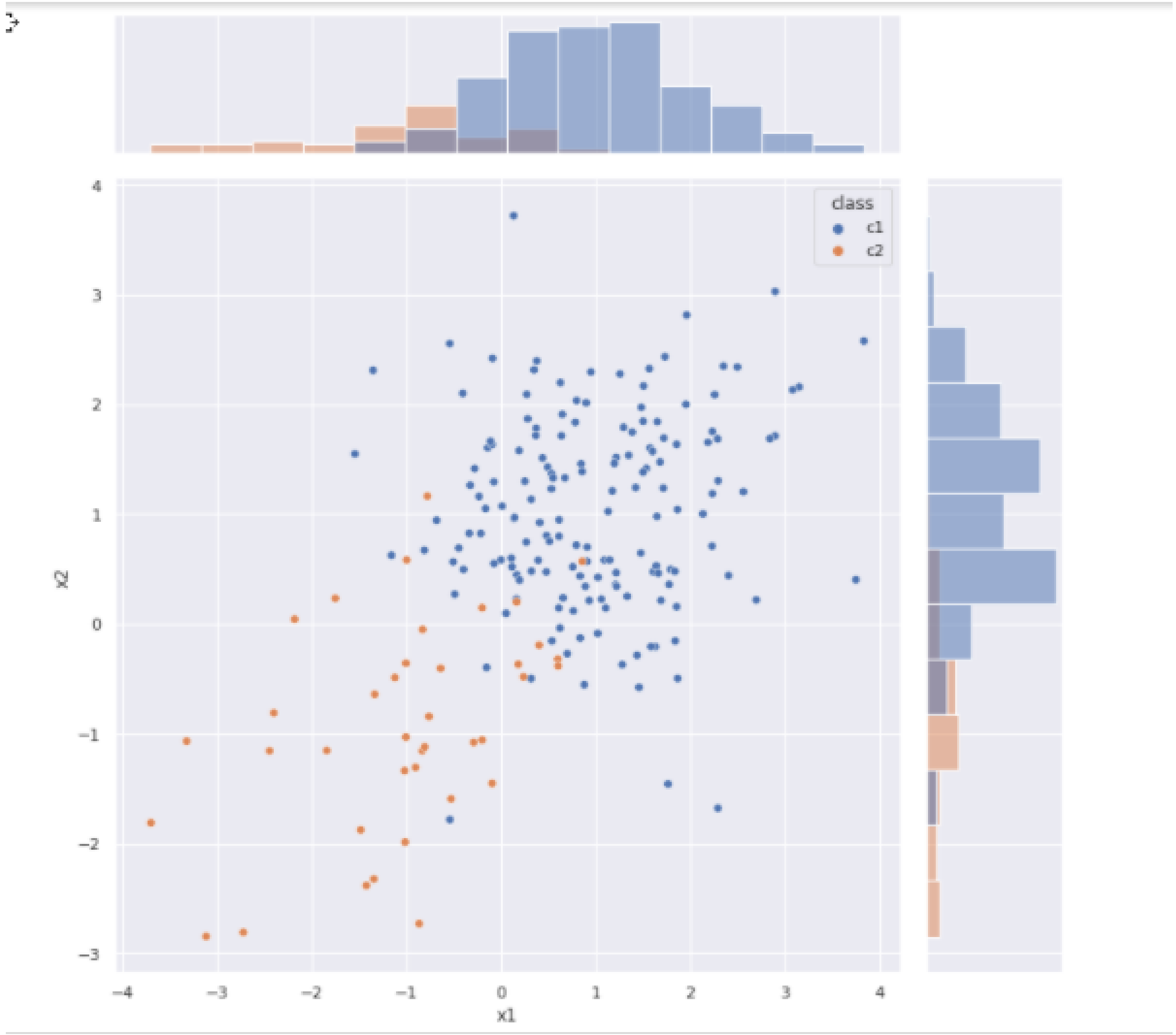
DATASET 2: (Notebook-2)

For this dataset, we are given two csv files , one for which samples belong to c1 class, and other for which samples belong to c2 class.

We already have labels in this case i.e data is already classified into c1 and c2 classes



The above plot shows the distribution of samples in the feature space .



The above plot is plotted using the code below

```
frames = [df_c1_class, df_c2_class]

result = pd.concat(frames)

s = sns.JointGrid(data= result, x="x1", y="x2",  height=10, hue="class")

g = s.plot(sns.scatterplot, sns.histplot)
```

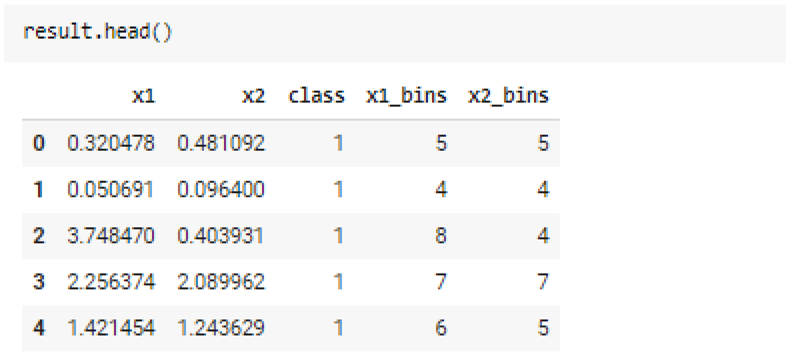
Firstly , we concatenate the dataframes formed from taking data from each csv file .Then we plot the Joingrid plot . The plot shows the distribution of samples from each class in the feature space as well as it also gives information about the count of x1 and x2 in the dataset

Now to find the posterior probabilities and count of unique elements , we will discretize the data . We observed that the count of unique elements without discretization is 1 for each unique value. Therefore we will divide our data into 8 bins .

```
bin_x1 = pd.cut(result["x1"],right=False,bins=8,labels=[1,2,3,4,5,6,7,8])

bin_x2 = pd.cut(result["x2"],right=False,bins=8,labels=[1,2,3,4,5,6,7,8])

result["x1_bins"]= bin_x1

result["x2_bins"] = bin_x2
```

Using pd.cut , we divided the “x1” as well as “x2” data into 8 bins and labelled the bins as 1,2,3,4,5,6,7,8 . After that we added the x1_bins and x2_bins columbine our result dataframe

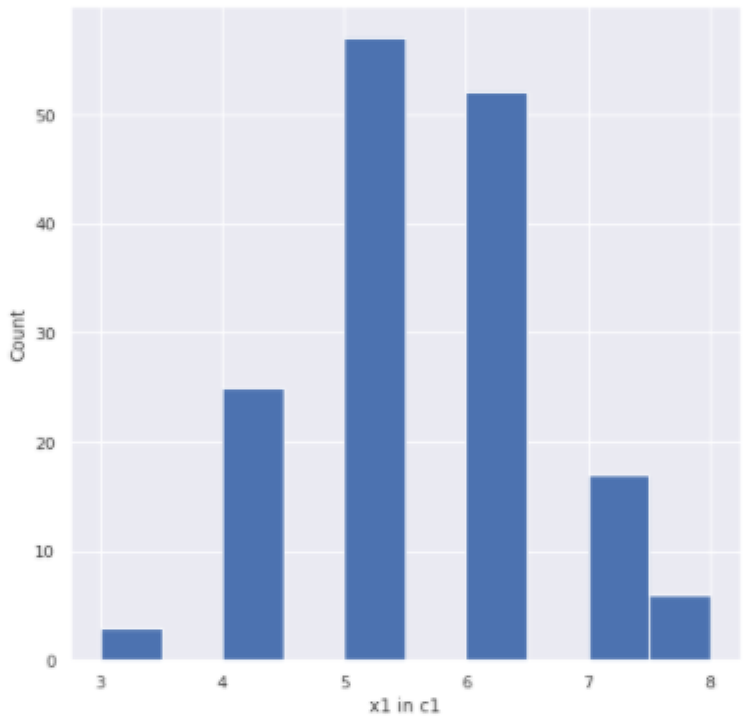
The result dataframe will look like this

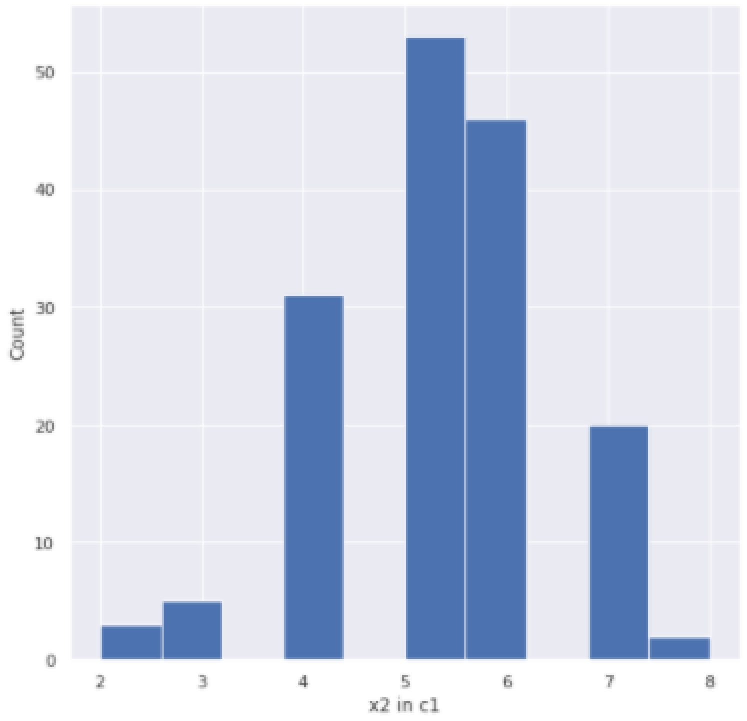


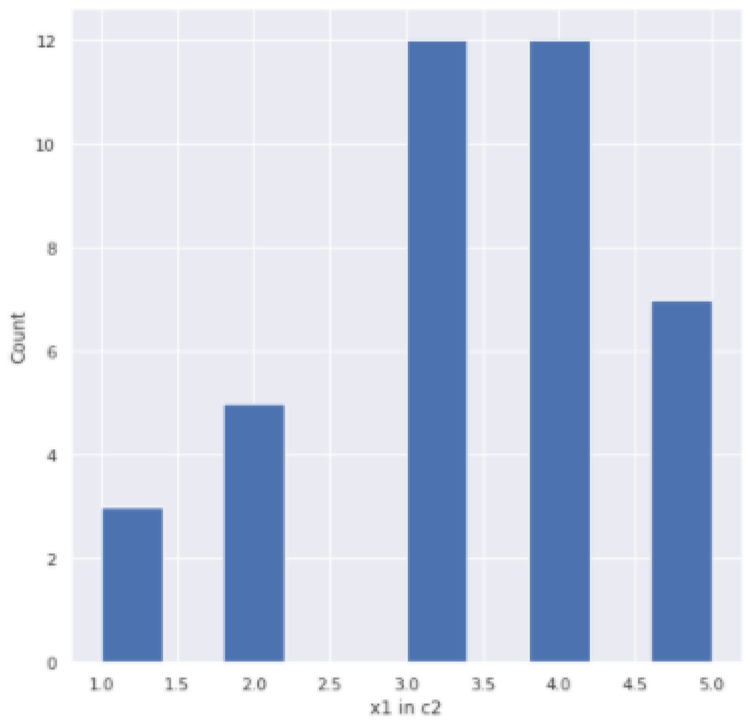
```
result.head()
```

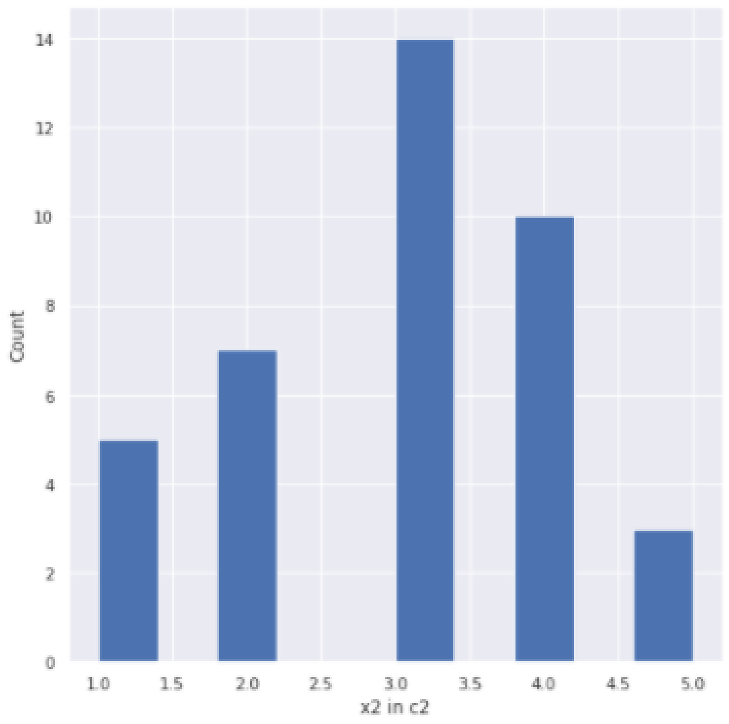
	x1	x2	class	x1_bins	x2_bins
0	0.320478	0.481092	1	5	5
1	0.050691	0.096400	1	4	4
2	3.748470	0.403931	1	8	4
3	2.256374	2.089962	1	7	7
4	1.421454	1.243629	1	6	5

Now the plot of unique values can be plotted easily using bins instead of actual values


Count
x1 in c1


Count
x2 in c1



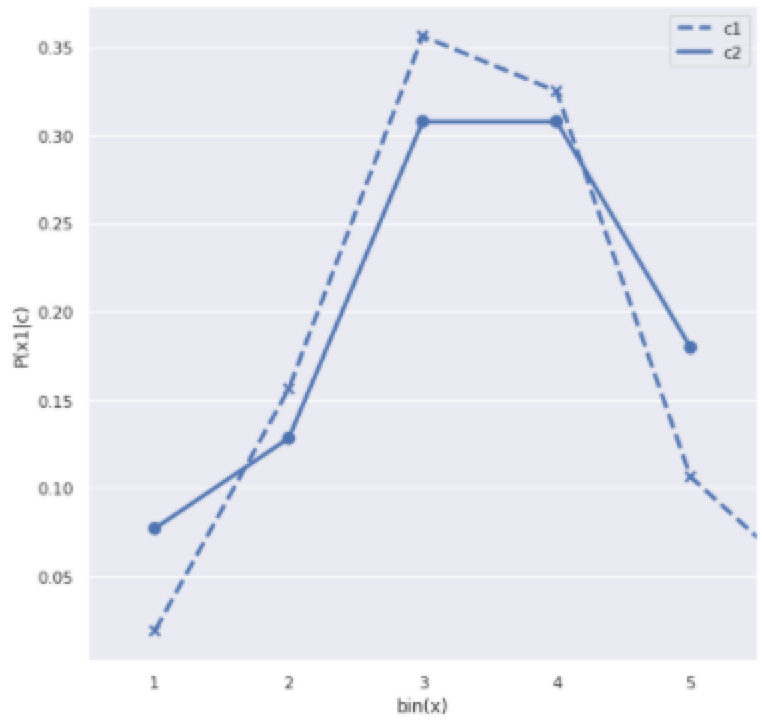


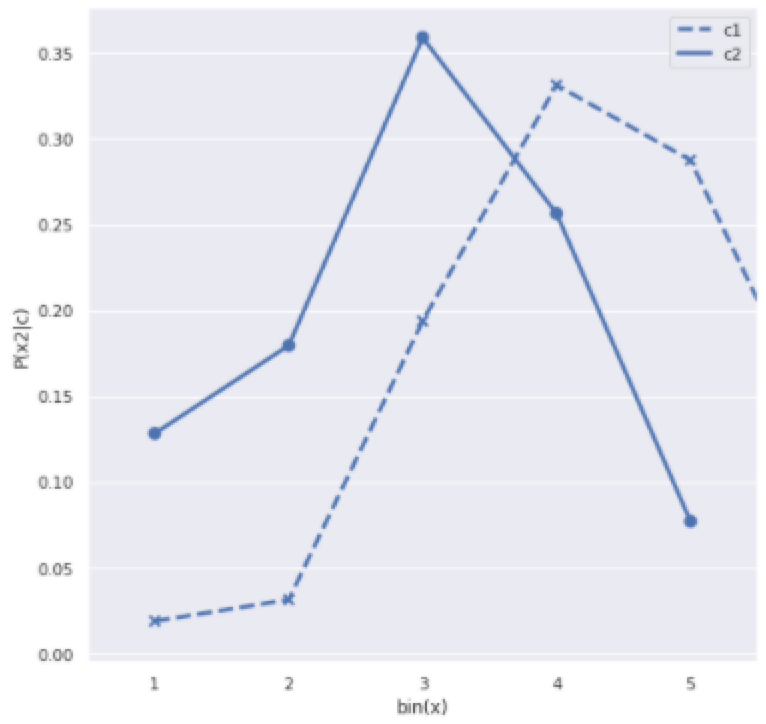
The above graphs represent the count of unique bins for each feature corresponding to each class

The code for posterior probability is similar to the previous question except the fact that now we have two features and we are working with bins instead of values.

```
y1_c1 = np.array(count_unique_x1_c1)/len(x1_c1)

y1_c2 = np.array(count_unique_x1_c2)/len(x1_c2)

y2_c1 = np.array(count_unique_x2_c1)/len(x2_c1)

y2_c2 = np.array(count_unique_x2_c2)/len(x2_c2)
```

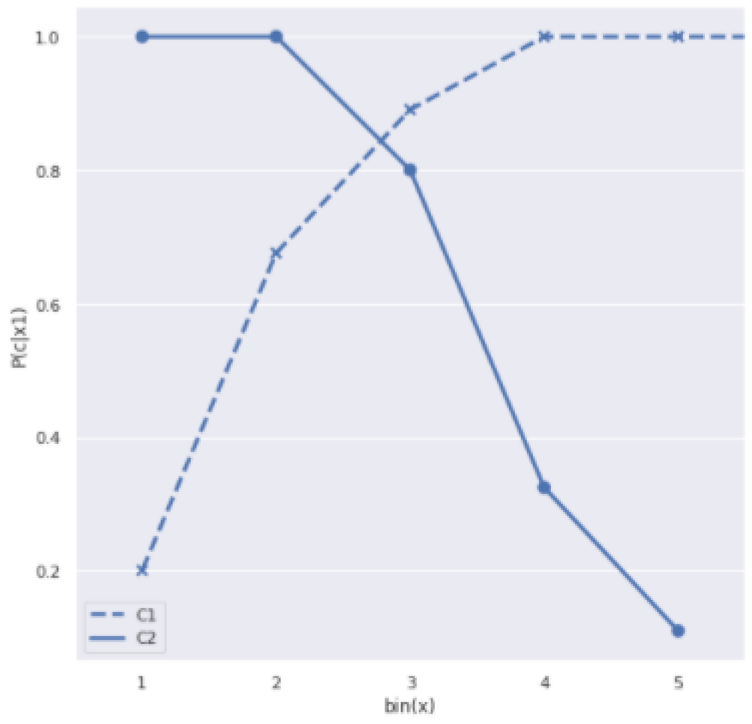
We compute the likelihood using the above code (similar to the previous question)

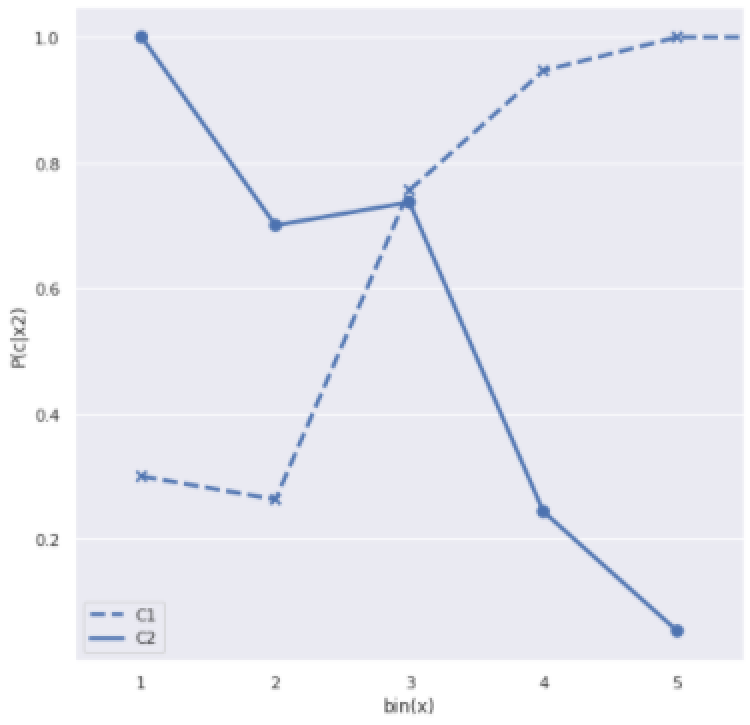




The posterior probability can be calculated easily using the likelihood , prior probabilities and evidence .

The plots of posterior probabilities are as follows





Real Life Dataset: (Notebook-1)

Now we have to repeat all the steps we did for Dataset 1 and apply on Iris dataset. We can choose any one feature to work with . I have chosen “`SepalLengthCm`” as the feature .

a. **Create the labels from the given data**.

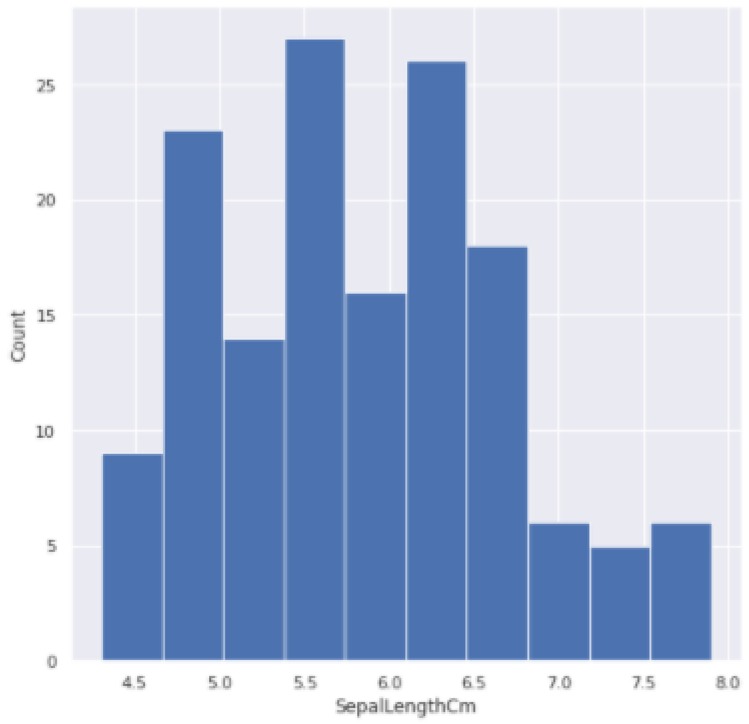
In this dataset , labels are already given i.e data is already labelled

```
x_c1 = []

x_c2 = []

x_c3 = []

for i in range(iris_vis_data.shape[0]):

  if(iris_vis_data[i][1]== 0):

    x_c1.append(iris_vis_data[i][0])

  elif (iris_vis_data[i][1]== 1):

    x_c2.append(iris_vis_data[i][0])

  elif (iris_vis_data[i][1]== 2):

     x_c3.append(iris_vis_data[i][0])
```

x_c1 , x_c2, x_c3 corresponds to the sepalLengthCm for each of the three classes.

b. **Plot the distribution of samples using histogram.**

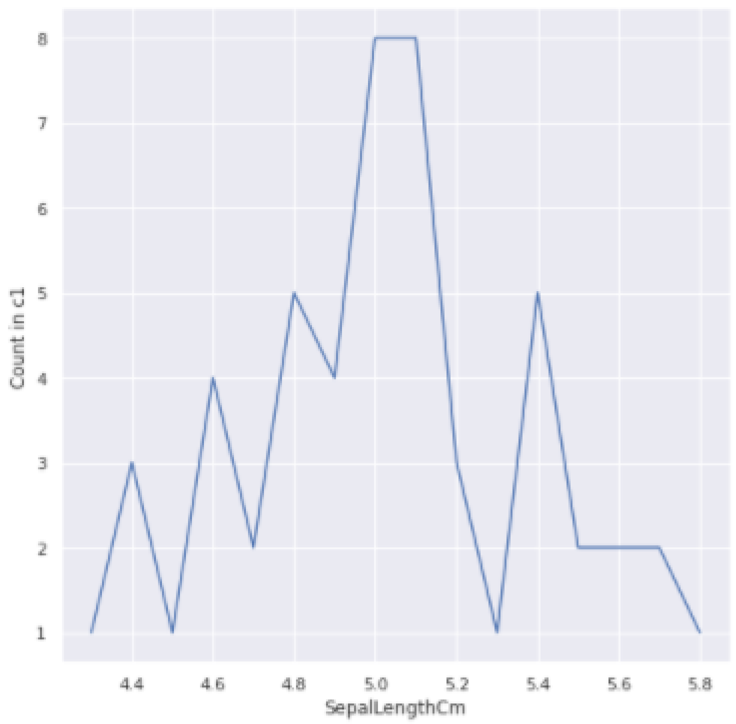
The distribution of samples is plotted as follows . The code used is similar to one used for dataset -2 (can be found in notebook)



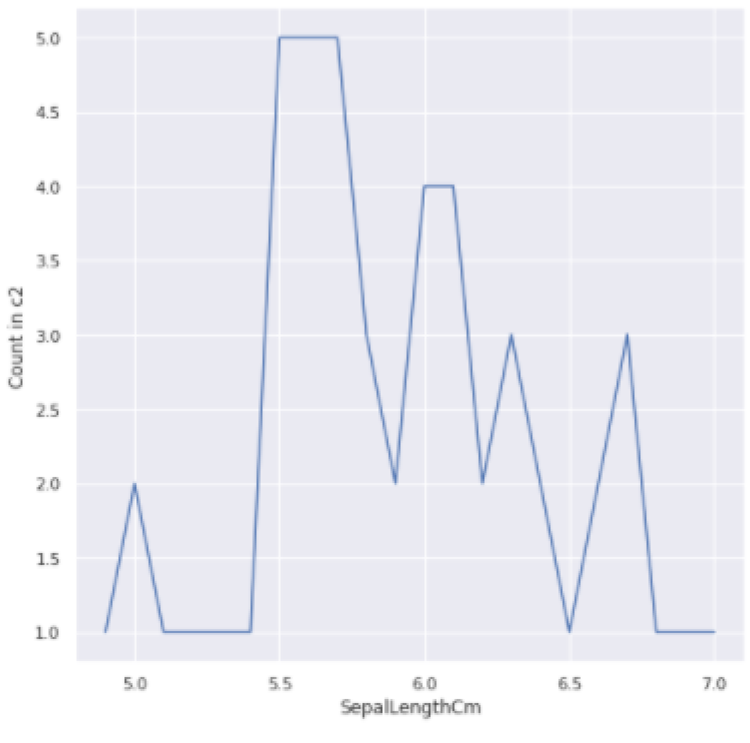
c. **Determine the prior probability for both the classes.**

```
prob_c1 = 50/ len(iris_vis_data)

prob_c2 = 50/ len(iris_vis_data)

prob_c3 = 50/ len(iris_vis_data)
```

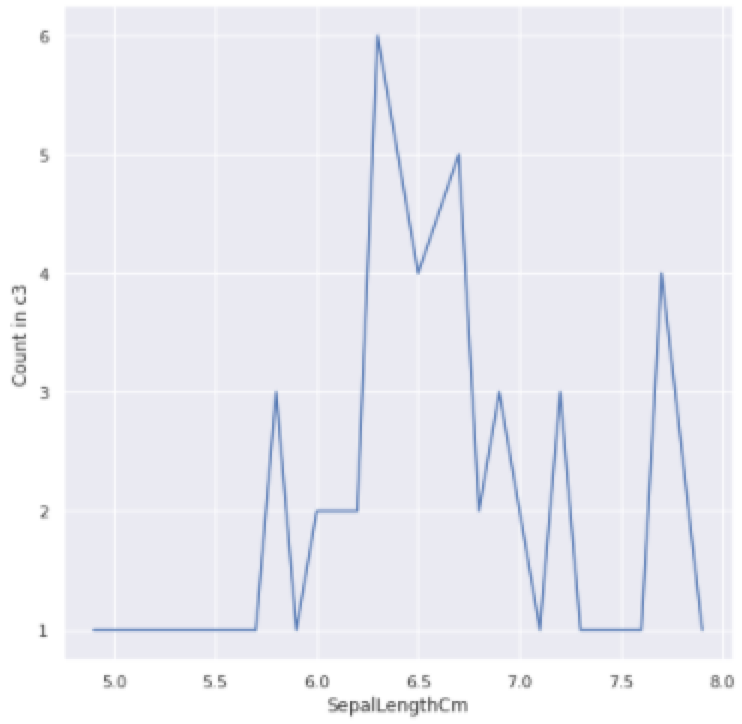
e. **Plot the count of each unique element for each class.**



Count in c1
SepalLengthCm
8
7
6
5
4
3
2
1
4.4
4.6
4.8
5.0
5.2
5.4
5.6
5.8



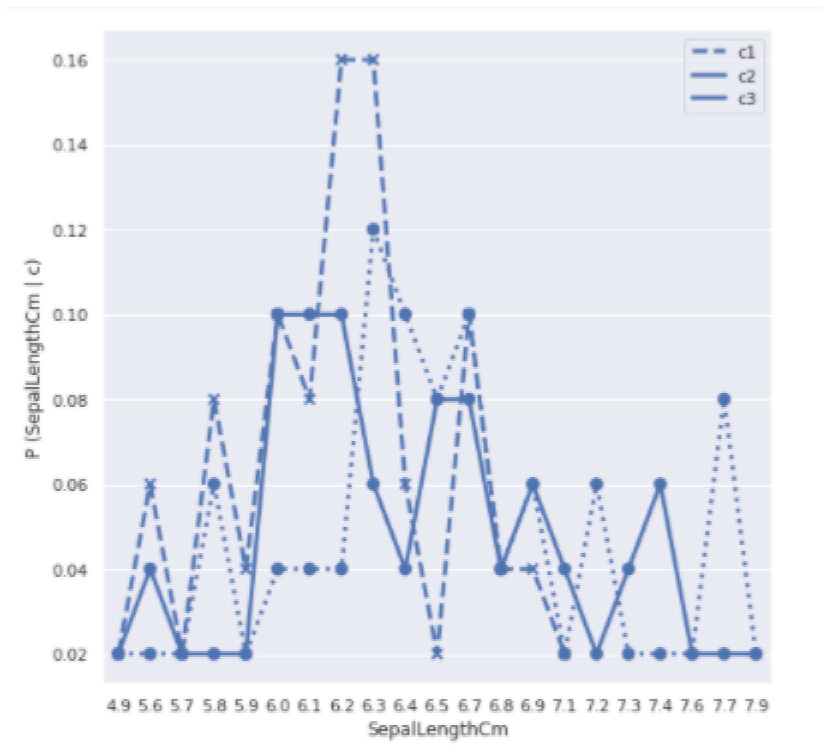
Count in c2
SepalLengthCm
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
5.0
5.5
6.0
6.5
7.0



d. **Determine the likelihood / class conditional probabilities for the classes.**

```
y1 = np.array(count_unique_c1)/len(x_c1)

y2 = np.array(count_unique_c2)/len(x_c2)

y3 = np.array(count_unique_c3)/len(x_c3)
```

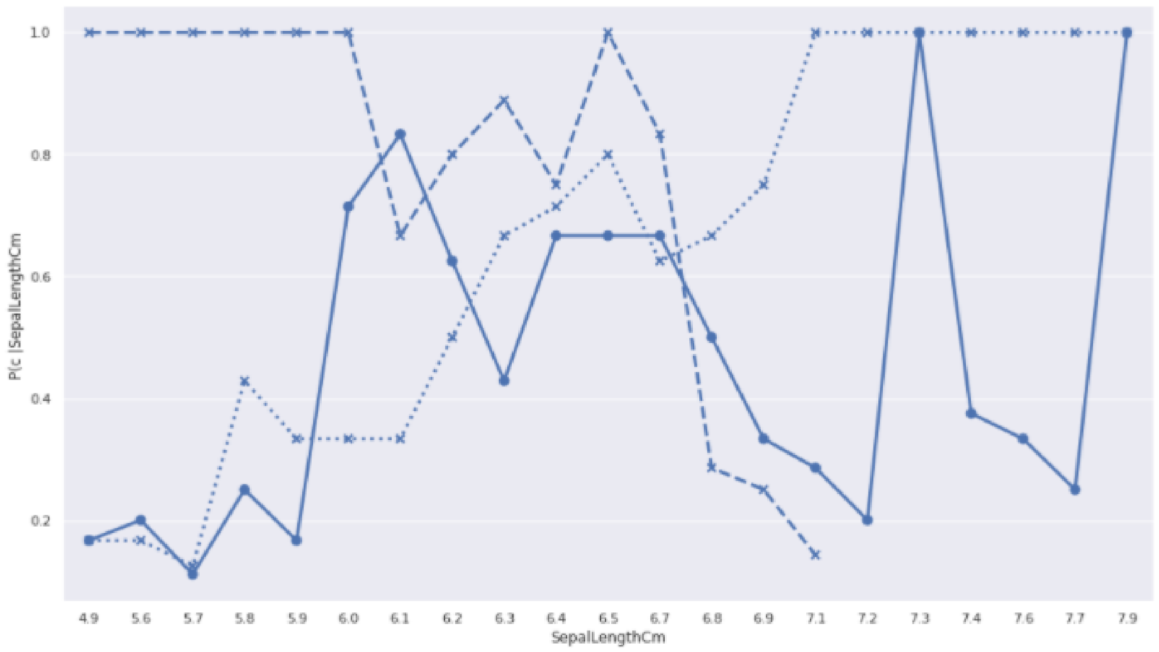
The above code is used to find likelihood for classes .(similar to Dataset-1)



e. **Calculate the P(C1|x) and P(C2|x) i.e posterior probabilities and plot them in a single graph.**

```
fig, ax = plt.subplots(figsize=fig_dims)
sns.pointplot(list(unique_c1),posterior_c1, linestyles="--",markers =['x'] )
sns.pointplot(list(unique_c2),posterior_c2,linestyles='-',markers=['o'])
sns.pointplot(list(unique_c3),posterior_c3, linestyles=":",markers =['x'] )
plt.xlabel("SepalLengthCm")
plt.ylabel("P(c |SepalLengthCm ")
```

```
Text(0, 0.5, 'P(c |SepalLengthCm ')
```


P(c |SepalLengthCm
1.0
0.8
0.6
0.4
0.2
4.9
5.6
5.7
5.8
5.9
6.0
6.1
6.2
6.3
6.4
6.5
6.7
6.8
6.9
7.1
7.2
7.3
7.4
7.6
7.7
7.9
SepalLengthCm