Pattern Recognition & Machine Learning
Lab-3 Assignment

Name: Tanish Pagaria
Roll No.: B21AI040

Question-1
Task-1
Pre-processing of the data
"PassengerId", *"Name"*, and "*Ticket"* did not have any significant role in survival prediction and they had many distinct unique entries hence not useful in classification. So, we decided to drop them.

```
df.isnull().sum()
```

```
PassengerId      0
Name             0
Pclass           0
Sex              0
Age            177
Ticket           0
Fare             0
Cabin          687
Embarked         2
Survived         0
dtype: int64
```

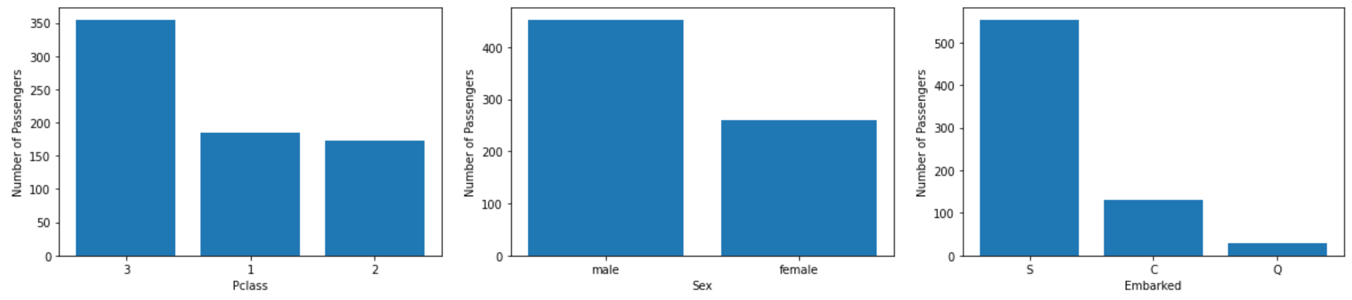
"Cabin" feature had many null values, hence it was also dropped. At the end, we removed all the NaN containing rows which were not very significant in number.
And, then all the categorical variables were encoded. At the end, we were left with the following dataframe:-

	Pclass	Sex	Age	Fare	Embarked	Survived
0	3	1	22.0	7.2500	2	0
1	1	0	38.0	71.2833	0	1
2	3	0	26.0	7.9250	2	1
3	1	0	35.0	53.1000	2	1
4	3	1	35.0	8.0500	2	0
...	...	...	...	...	...	...
885	3	0	39.0	29.1250	1	0
886	2	1	27.0	13.0000	2	0
887	1	0	19.0	30.0000	2	1
889	1	1	26.0	30.0000	0	1
890	3	1	32.0	7.7500	1	0

712 rows × 6 columns

Visualisation of the data

Using bar graphs to show distribution of the data points in the features.



Splitting the data into train and test sets

70:30 splitting performed using `sklearn.model_selection.train_test_split`.

Task-2

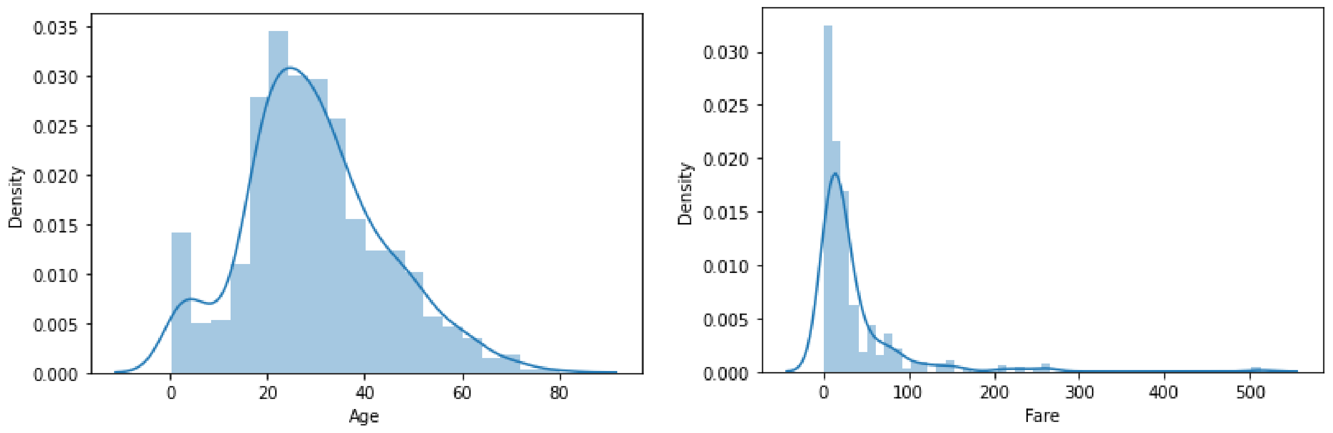
Best possible Naive Bayes variant for the given dataset

→ *Gaussian Naive Bayes*

There are three main variants of the Naive Bayes classifier: *GaussianNB*, *MultinomialNB*, and *BernoulliNB*. BernoulliNB is useful if the feature vectors are binary (i.e., 0's & 1's). Hence, it can't be used here since it the feature vectors given to us don't fulfill this condition.

MultinomialNB mostly deals with number of occurrences of a term or its relative frequency. Hence, it also can't be used here since we mostly have categorical features and continuous data points (in case of *"Age"* and *"Fare"*).

Gaussian Naive Bayes is an effective choice here, as it is evident from the graphs shown below, that the non-categorical features follow a Normal distribution and are continuous data points.



(Graphs indicating Gaussian (Normal) Distribution of the continuous features)

Task-3

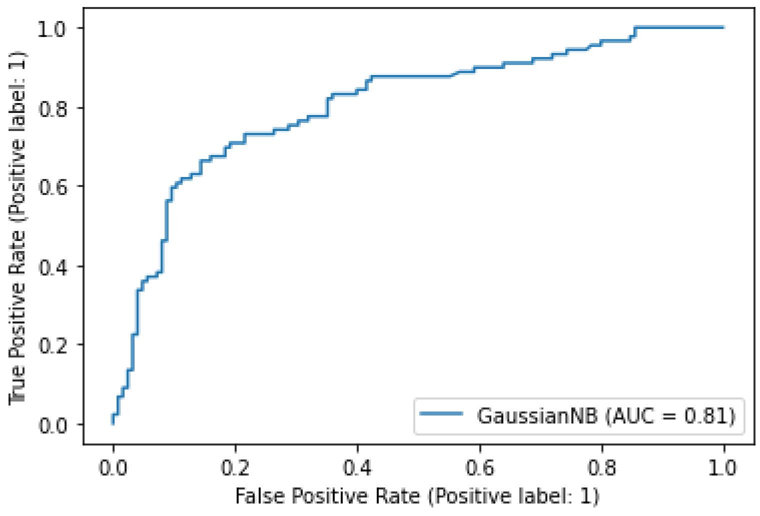
Implementation of Gaussian Naive Bayes Classifier using scikit learn

We use the inbuilt `class sklearn.naive_bayes.GaussianNB` for implementing the classifier.

Using `roc_auc_score, plot_roc_curve` from `sklearn.metrics`:-

```
roc_auc_score(test_y, pred_y)
```

```
0.7426067415730336
```

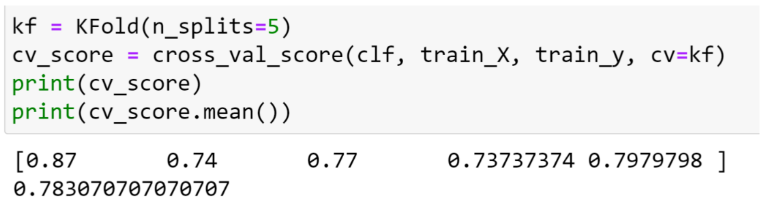


ROC Curve for GaussianNB Classifier

Task-4

<u>5 fold cross validation</u>

Using `sklearn.model_selection.KFold`.



```
kf = KFold(n_splits=5)
cv_score = cross_val_score(clf, train_X, train_y, cv=kf)
print(cv_score)
print(cv_score.mean())
```

```
[0.87        0.74        0.77        0.73737374 0.7979798 ]
0.783070707070707
```

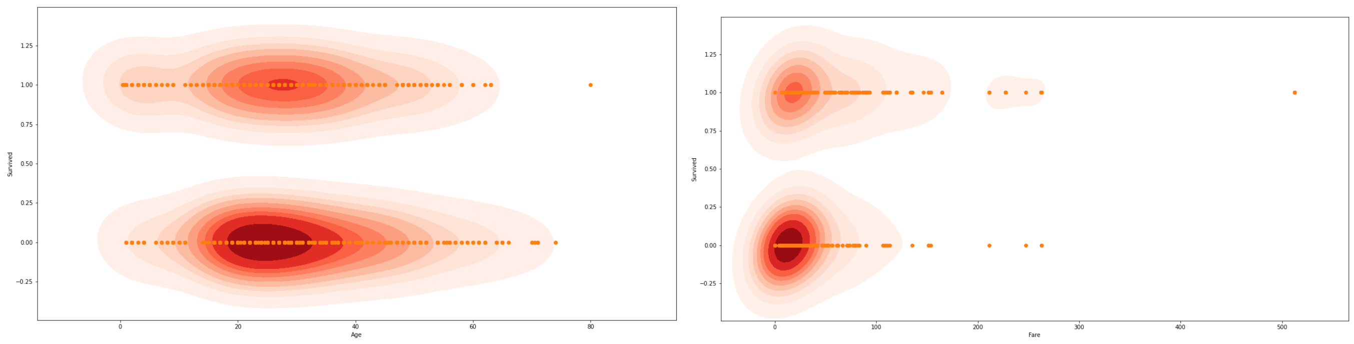
<u>Probability computation of the the top class for each row in the testing dataset</u>

Using `predict_proba()` method, we get an array of arrays showing class probabilities of each data row in the dataframe. We can then then create an empty list and then iterate through the probabilities array and append the maximum probability out of all the given classes for that data row.

Task-5

<u>Contour plots to visualize the class-conditional densities</u>

Using kdeplot() from seaborn library, we plotter the contour plots for the non-categorical features.



Contour plots for "Age" and "Fare" respectively

From the contour plots we can say that, the Naive Bayes model is based on the assumption that features are independent of each other. We can say that because the area of the contour plots do not overlap.

Task-6

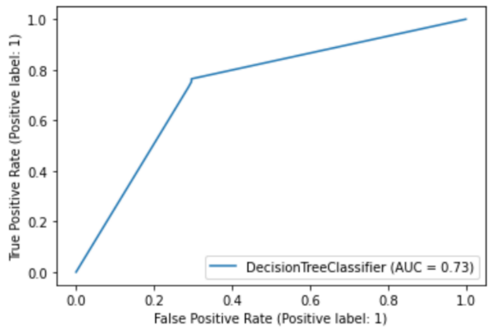
<u>Comparision with Decision Tree classifier</u>

The decision tree classifier was implemented using the inbuilt class `sklearn.tree.DecisionTreeClassifier`.

```
def kfold_cv(train_X, train_y, test_X, test_y, depth=None, split=2, leaf=1):
    dt_clf = DecisionTreeClassifier(random_state=0, max_depth=depth, min_samples_split=split, min_samples_leaf=leaf)
    kf = KFold(n_splits=5)
    cv_score = cross_val_score(dt_clf, train_X, train_y, cv=kf)
    print(cv_score)
    print(cv_score.mean())

    dt_clf.fit(train_X, train_y)
    plot_roc_curve(dt_clf, test_X, test_y)
```

```
kfold_cv(train_X, train_y, test_X, test_y)
```

```
[0.79       0.79       0.76       0.80808081 0.73737374]
0.777090909090909
```



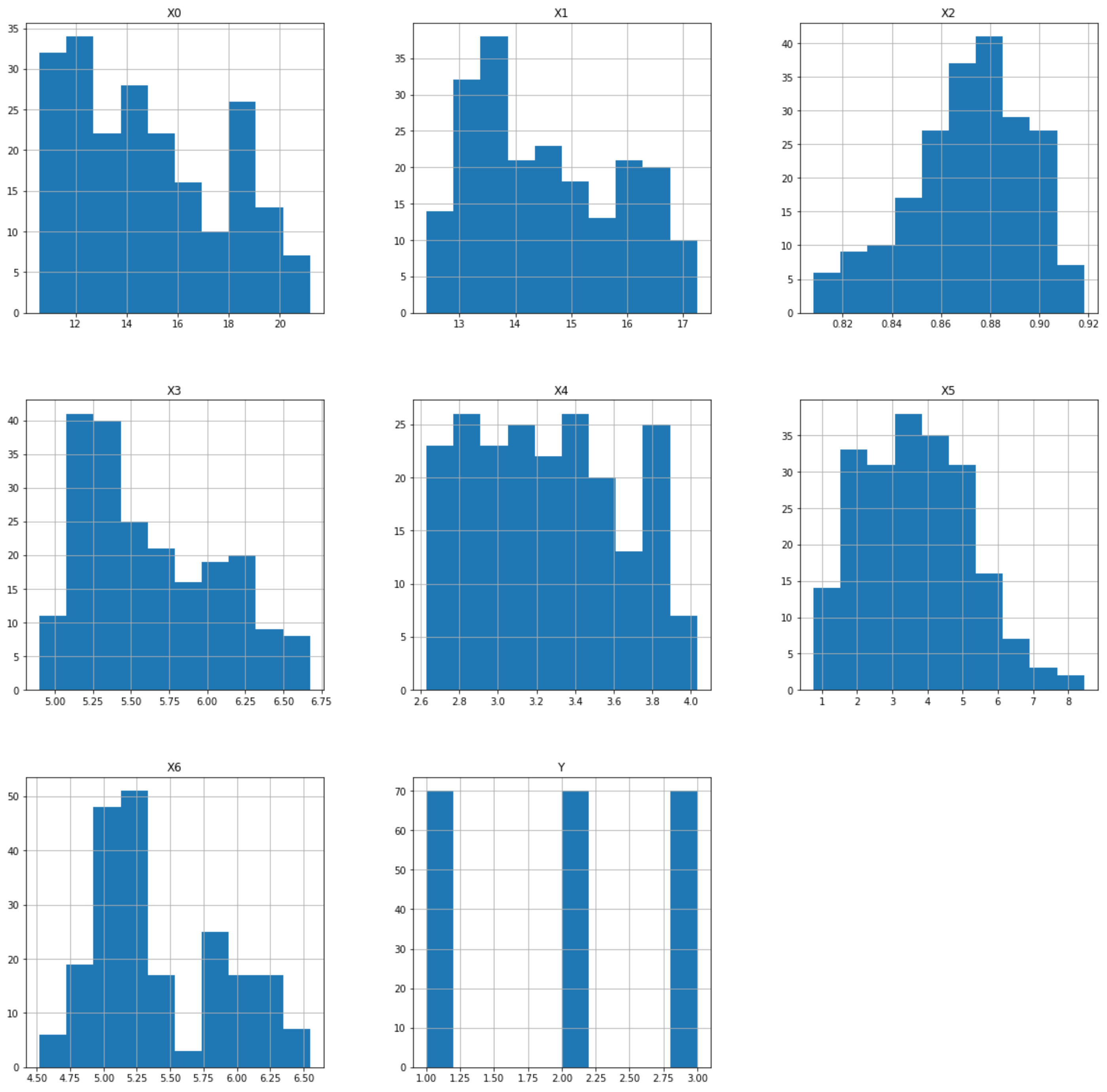
We observe that the area under the curve (AUC) for Gaussian Naive Bayes Classifier is greater than that of the Decision Tree Classifier.

Hence, Gaussian Naive Bayes seems to work better than Decision Tree Classifier on this type of numeric data. Also, Naive Bayes models seem to work better with small data. Also, Naive Bayes works better on categorical input variables (which we have in this case).

Question-2

Histogram plots to show the distribution of samples

(carried out by simply using `hist()` method in `pandas`)



Prior probability of all the classes

Prior Probability = Probability(sample belongs to the given class)

Prior probabilities for each class were calculated by finding the total number of samples of each of the 3 given classes and dividing them by the total number of samples.

```python
def prior(Class):
    return len(df[df['Y']==Class])/len(df)
```

Discretization of features into bin

We used the differences in the data points in a feature in the histograms as the criterion to make bins. For example, in the feature “X0”, we used 2 as the difference, i.e., suppose if a data point lied between 12-14 then it was labelled as ‘12-14’ in a new column created in the dataframe. The changed dataframe is shown below:-

```
features = ['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6']
diff_for_features = [2, 1, 0.02, 0.25, 0.2, 1, 0.25]
```

```
for i in range(len(features)):
    df = binning_using_difference(df, features[i], diff_for_features[i])
```

```
df
```

	X0	X1	X2	X3	X4	X5	X6	Y	bin-X0	bin-X1	bin-X2	bin-X3	bin-X4	bin-X5	bin-X6
0	15.26	14.84	0.8710	5.763	3.312	2.221	5.220	1	14.00-16.00	14.00-15.00	0.86-0.88	5.75-6.00	3.20-3.40	2.00-3.00	5.00-5.25
1	14.88	14.57	0.8811	5.554	3.333	1.018	4.956	1	14.00-16.00	14.00-15.00	0.88-0.90	5.50-5.75	3.20-3.40	1.00-2.00	4.75-5.00
2	14.29	14.09	0.9050	5.291	3.337	2.699	4.825	1	14.00-16.00	14.00-15.00	0.90-0.92	5.25-5.50	3.20-3.40	2.00-3.00	4.75-5.00
3	13.84	13.94	0.8955	5.324	3.379	2.259	4.805	1	12.00-14.00	13.00-14.00	0.88-0.90	5.25-5.50	3.20-3.40	2.00-3.00	4.75-5.00
4	16.14	14.99	0.9034	5.658	3.562	1.355	5.175	1	16.00-18.00	14.00-15.00	0.90-0.92	5.50-5.75	3.40-3.60	1.00-2.00	5.00-5.25
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
205	12.19	13.20	0.8783	5.137	2.981	3.631	4.870	3	12.00-14.00	13.00-14.00	0.86-0.88	5.00-5.25	2.80-3.00	3.00-4.00	4.75-5.00
206	11.23	12.88	0.8511	5.140	2.795	4.325	5.003	3	10.00-12.00	12.00-13.00	0.84-0.86	5.00-5.25	2.60-2.80	4.00-5.00	5.00-5.25
207	13.20	13.66	0.8883	5.236	3.232	8.315	5.056	3	12.00-14.00	13.00-14.00	0.88-0.90	5.00-5.25	3.20-3.40	8.00-9.00	5.00-5.25
208	11.84	13.21	0.8521	5.175	2.836	3.598	5.044	3	10.00-12.00	13.00-14.00	0.84-0.86	5.00-5.25	2.80-3.00	3.00-4.00	5.00-5.25
209	12.30	13.34	0.8684	5.243	2.974	5.637	5.063	3	12.00-14.00	13.00-14.00	0.86-0.88	5.00-5.25	2.80-3.00	5.00-6.00	5.00-5.25

210 rows × 15 columns

Later, we changed the names of these bins to numbers starting from the lowest bin named as 1 and so on. This was done for easy plotting of graphs in further questions.

	X0	X1	X2	X3	X4	X5	X6	Y	bin-X0	bin-X1	bin-X2	bin-X3	bin-X4	bin-X5	bin-X6
0	15.26	14.84	0.8710	5.763	3.312	2.221	5.220	1	3	3	4	5	4	3	3
1	14.88	14.57	0.8811	5.554	3.333	1.018	4.956	1	3	3	5	4	4	2	2
2	14.29	14.09	0.9050	5.291	3.337	2.699	4.825	1	3	3	6	3	4	3	2
3	13.84	13.94	0.8955	5.324	3.379	2.259	4.805	1	2	2	5	3	4	3	2
4	16.14	14.99	0.9034	5.658	3.562	1.355	5.175	1	4	3	6	4	5	2	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
205	12.19	13.20	0.8783	5.137	2.981	3.631	4.870	3	2	2	4	2	2	4	2
206	11.23	12.88	0.8511	5.140	2.795	4.325	5.003	3	1	1	3	2	1	5	3
207	13.20	13.66	0.8883	5.236	3.232	8.315	5.056	3	2	2	5	2	4	9	3
208	11.84	13.21	0.8521	5.175	2.836	3.598	5.044	3	1	2	3	2	2	4	3
209	12.30	13.34	0.8684	5.243	2.974	5.637	5.063	3	2	2	4	2	2	6	3

210 rows × 15 columns

Likelihood / Class-conditional Probabilities of all classes

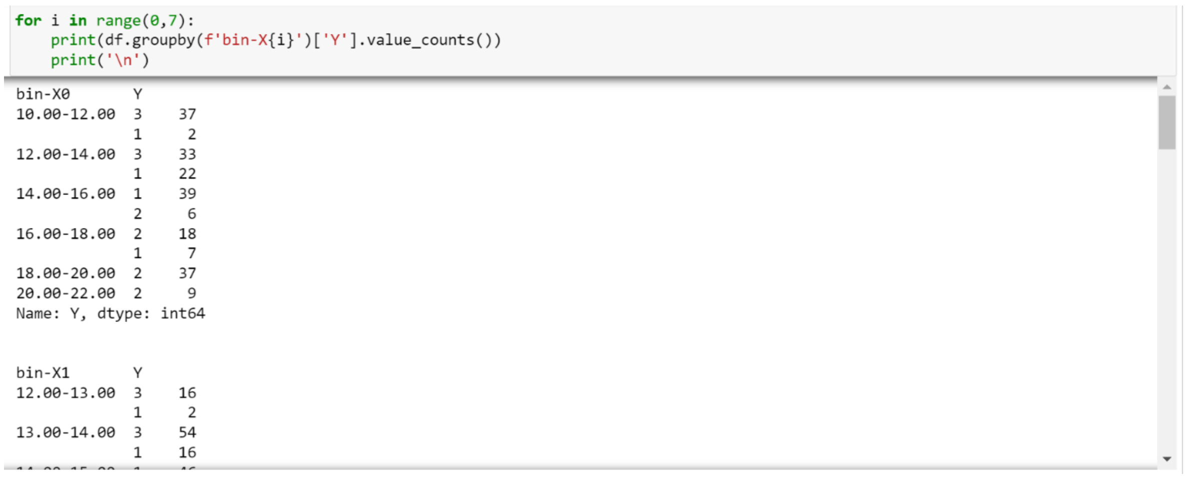
Likelihood = $p(x \mid \omega j) = p(x \cap \omega j) / p(\omega j)$

Using the above formula, likelihood probabilities of all classes for each bin of the features was calculated. The probability was calculated by using `pandas` and finding the number of required data points (measuring the length of that part of the data frame).

```
def likelihood(new_df, Feature, Bin, Class):
    L = len(new_df)

    new_df = new_df[new_df[Feature]==Bin]
    new_df = new_df[new_df['Y']==Class]

    return len(new_df)/(L * prior_dict[Class])
```

Plotting the count of each unique element for each class

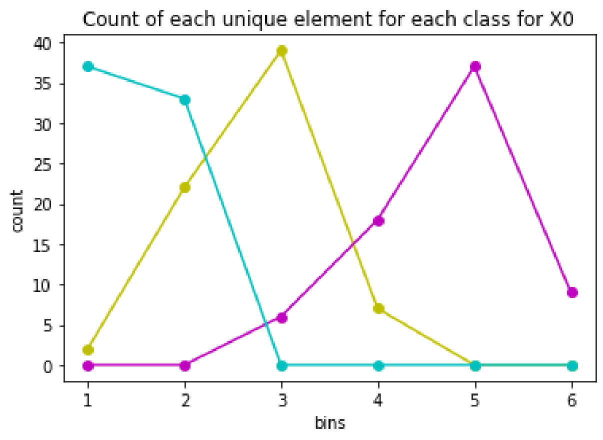
The `pandas` functions `groupby()` and `value_counts()` were used to find the count of each bin in a feature that belonged to a particular class.

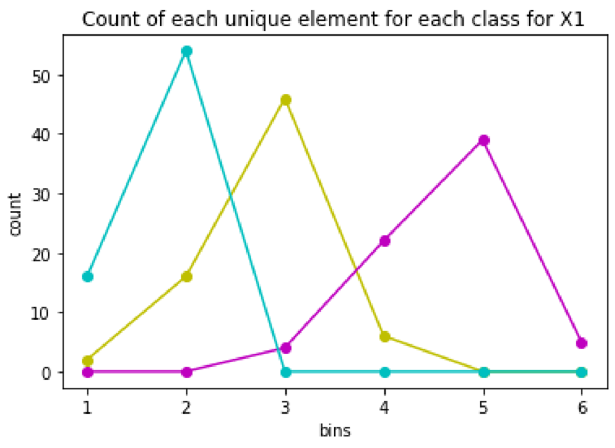


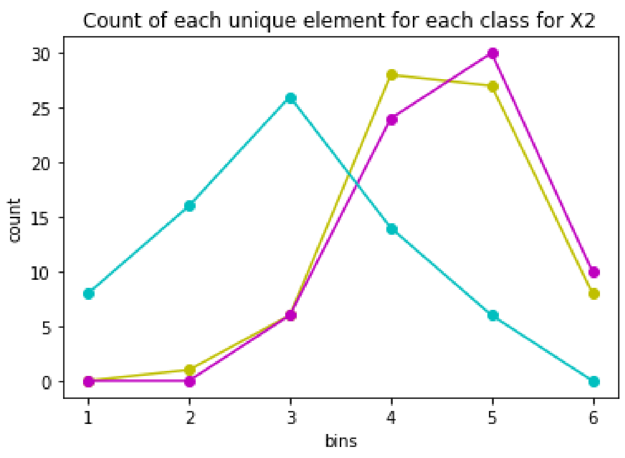
```python
for i in range(0,7):
    print(df.groupby(f'bin-X{i}')['Y'].value_counts())
    print('\n')
```

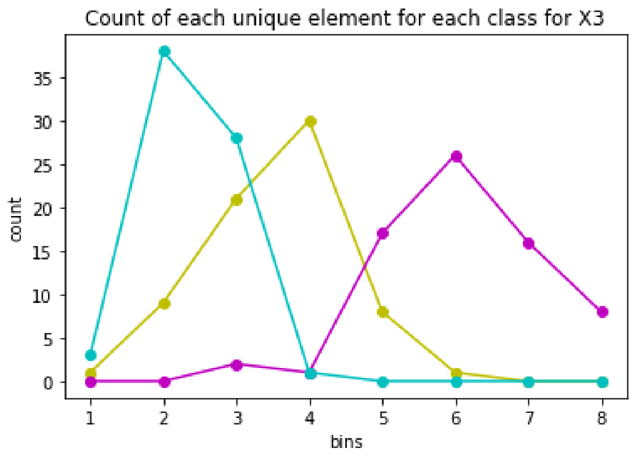
```
bin-X0       Y
10.00-12.00  3    37
             1     2
12.00-14.00  3    33
             1    22
14.00-16.00  1    39
             2     6
16.00-18.00  2    18
             1     7
18.00-20.00  2    37
20.00-22.00  2     9
Name: Y, dtype: int64


bin-X1       Y
12.00-13.00  3    16
             1     2
13.00-14.00  3    54
             1    16
```

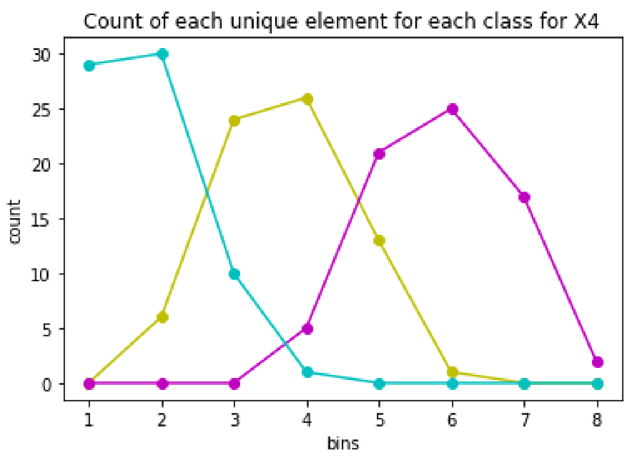
Then, these values were extracted as dictionaries and used in graph plotting.

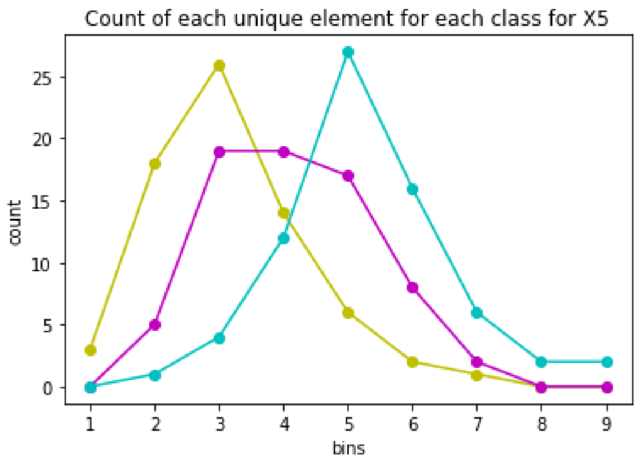


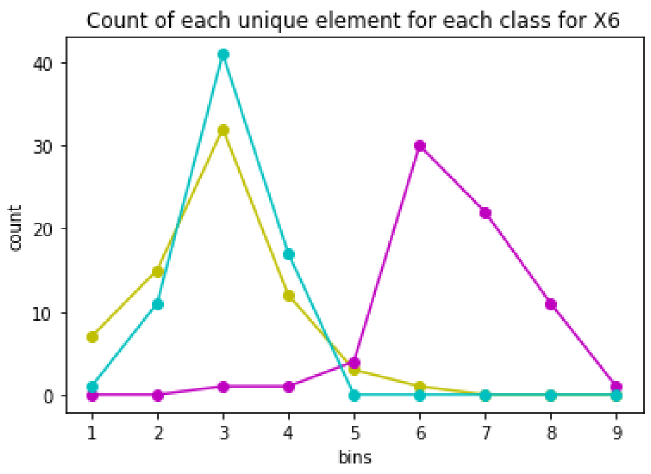












Graphs showing count of each unique element for each class
(Here, classes 1, 2 & 3 are represented using yellow, magenta and cyan colors respectively)

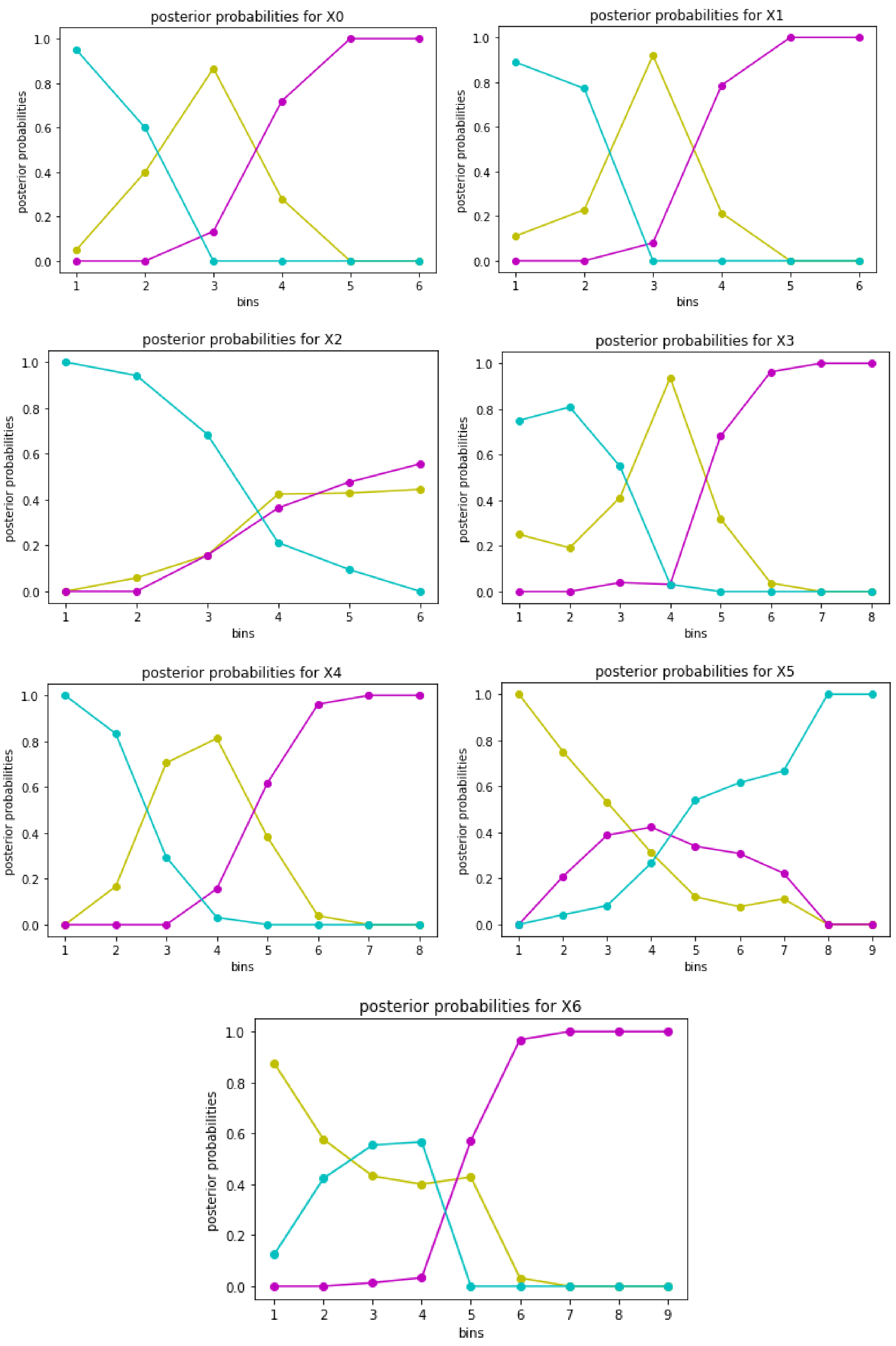
Calculation and Plotting of Posterior probabilities

Posterior Probability = Likelihood * Prior / Evidence

P(ωj | x) = P(x | ωj) * P(ωj) / P(x)

Similar to likelihood calculation, we calculate posterior probabilities and plot graphs accordingly.

```python
def posterior(new_df, Feature, Bin, Class):
    L = len(new_df[new_df[Feature]==Bin])

    new_df = new_df[new_df[Feature]==Bin]
    new_df = new_df[new_df['Y']==Class]

    return len(new_df)/L
```



Graphs showing posterior probabilities of each bin of features for each class
(Here, classes 1, 2 & 3 are represented using yellow, magenta and cyan colors respectively)

Graph analysis:
More area under the curve shows more probability at that point for the given class.
At every point the sum of posteriors is unity.