Machine Learning – Assignment 2

Submitted by Debonil Ghosh (M21AIE225)

Question 1:

Download the dataset, where the first four columns are features, and the last column corresponds to categories (3 labels). Perform the following tasks. #

1. Split the dataset into train and test sets (80:20)
2. Construct the Naive Bayes classifier from scratch and train it on the train set. Assume Gaussian distribution to compute probabilities.
3. Evaluate the performance using the following metric on the test set a. Confusion matrix b. Overall and class-wise accuracy c. ROC curve, AUC
4. Use any library (e.g. scikit-learn) and repeat 1 to 3
5. Compare and comment on the performance of the results of the classifier in 2 and 4
6. Calculate the Bayes risk. Consider,

   λ =
   2 4 6
   1 2 3
   6 4 1
   Where λ is a loss function and rows and columns corresponds to classes (ci) and actions (aj) respectively, e.g. λ(a3 / c2) =4.

In [ ]:
```
import numpy as np
import pandas as pd
```

- Gaussian Naive Bayes Classifier From Scratch

In [ ]:
```
class GaussianNBClassifierFromScratch():
    def calcPrior(self, features: pd.DataFrame, target: pd.DataFrame):
        self.prior = (features.groupby(target).apply(
            lambda x: len(x)) / self.dataCount).to_numpy()
        return self.prior

    def calcStatistics(self, features: pd.DataFrame, target: pd.DataFrame):
        self.mean = features.groupby(target).apply(np.mean).to_numpy()
        self.var = features.groupby(target).apply(np.var).to_numpy()
        return self.mean, self.var

    def gaussianDensity(self, class_idx, x):
        '''
        (1/√2pi*σ) * exp((-1/2)*((x-μ)^2)/(2*σ²))
        '''
        mean = self.mean[class_idx]
        var = self.var[class_idx]
        numerator = np.exp((-1/2)*((x-mean)**2) / (2 * var))
        denominator = np.sqrt(2 * np.pi * var)
        prob = numerator / denominator
```

```
        return prob

    def calcPosterior(self, x):
        posteriors = []

        # calculate posterior probability for each class
        for i in range(self.classCount):
            prior = self.prior[i]
            conditional = np.prod(self.gaussianDensity(i, x))

            #print('posteriors for [{}], prob ==> [{}]'.format(x, self.gaussianDens
            #print('posteriors for [{}], cond ==> [{}]'.format(x, conditional))
            posterior = prior * conditional
            posteriors.append(posterior)

        #print('posteriors for [{}]==> [{}]'.format(x, posteriors))
        return posteriors

    def fit(self, features: pd.DataFrame, target: pd.DataFrame):
        self.classes = np.unique(target)
        self.classCount = len(self.classes)
        self.featureCount = features.shape[1]
        self.dataCount = features.shape[0]

        self.calcStatistics(features, target)
        self.calcPrior(features, target)

        print('GaussianNBClassifier trained :')
        print('\t mean ==> {}'.format(self.mean))
        print('\t var ==> {}'.format(self.var))
        print('\t prior ==> {}'.format(self.prior))

    def predict(self, features):
        classProbabilities = self.predictProbabilities(features)
        # return class with highest posterior probability
        return np.array([self.classes[np.argmax(cp)] for cp in classProbabilities]

    def predictProbabilities(self, features):

        return np.array([self.calcPosterior(f) for f in features.to_numpy()])
```

- Utility function for confusion Matrix And Accuracy Report

In [ ]:

```
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(rc={'axes.facecolor': 'lightblue', 'figure.facecolor': 'lightblue'})


def confusionMatrixAndAccuracyReport(Y_test, Y_pred):
    cm = metrics.confusion_matrix(Y_test, Y_pred)
    overallAccuracy = np.trace(cm)/sum(cm.flatten())

    classwiseAccuracy = np.zeros(len(cm))
    for n in range(len(cm)):
        for i in range(len(cm)):
            for j in range(len(cm)):
                if (i != n and j != n) or (i == n and j == n):
                    classwiseAccuracy[n] += cm[i][j]

    classwiseAccuracy /= sum(cm.flatten())
```

```
    plt.figure(figsize=(6, 6))
    plt.title('Accuracy Score: {0:3.3f}'.format(overallAccuracy), size=12)
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')
    sns.heatmap(data=cm, annot=True, square=True,  cmap='Blues')

    plt.show()
    print('Overall Accuracy Score: {0:3.3f}'.format(overallAccuracy))
    print('Classwise Accuracy Score: {0}'.format(classwiseAccuracy))
```

- Utility Fn for rocCurveAndAucAnalysis

In [ ]:
```
def rocCurveAndAucAnalysis(y_test, y_score):
    n_classes = y_test.shape[1]
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(n_classes):
        fpr[i], tpr[i], _ = metrics.roc_curve(y_test[:, i], y_score[:, i])
        roc_auc[i] = metrics.auc(fpr[i], tpr[i])

    plt.title("ROC Curve")
    plt.plot([0, 1], [0, 1], linestyle='--', lw=2,
             color='r', label='Random', alpha=.8)

    colors = ["aqua", "darkorange", "cornflowerblue"]
    for i, color in zip(range(n_classes), colors):
        plt.plot(
            fpr[i],
            tpr[i],
            color=color,
            lw=2,
            label="ROC curve of class {0} (area = {1:0.2f})".format(
                i, roc_auc[i]),
        )

    plt.xlabel("False Positve Rate")
    plt.ylabel("True Positive Rate")
    plt.legend()
    plt.show()

    for i in range(n_classes):
        print("Area Under the curve for class {} : {}".format(i, roc_auc[i]))
```

- Data load

In [ ]:
```
data = pd.read_csv('data-ques-1/iris_dataset.csv')
data.head()
```

Out[ ]:

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa

1. Split the dataset into train and test sets (80:20)

In [ ]:
```
from sklearn.preprocessing import label_binarize
from sklearn.model_selection import train_test_split

X = data.drop("variety", axis=1)
Y = data["variety"]
X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size=0.2, random_state=23)


print('Given data set==> {}'.format(data.shape))
print('Train data set==> {}'.format(X_train.shape))
print('Test data set==> {}'.format(X_test.shape))
# for use in roc analysis
y_test_bin = label_binarize(Y_test, classes=np.unique(Y_test))
```

```
Given data set==> (150, 5)
Train data set==> (120, 4)
Test data set==> (30, 4)
```

2. Construct the Naive Bayes classifier from scratch and train it on the train set. Assume Gaussian distribution to compute probabilities.

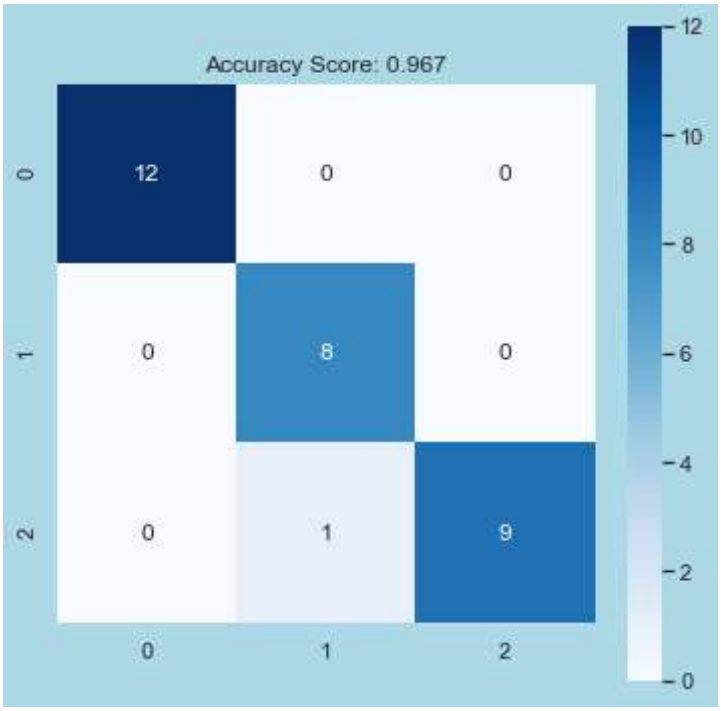
In [ ]:
```
modelFromScratch = GaussianNBClassifierFromScratch()
modelFromScratch.fit(X_train, Y_train)
Y_pred = modelFromScratch.predict(X_test)
y_score = modelFromScratch.predictProbabilities(X_test)
```

```
GaussianNBClassifier trained :
        mean ==> [[4.98157895 3.41578947 1.43684211 0.24473684]
 [5.89761905 2.74761905 4.24285714 1.33095238]
 [6.57       2.9725     5.5725     2.03      ]]
        var ==> [[0.12781856 0.1565928  0.02811634 0.01299861]
 [0.26451814 0.10344671 0.22244898 0.03975624]
 [0.4181     0.11649375 0.32799375 0.0741    ]]
        prior ==> [0.31666667 0.35       0.33333333]
```

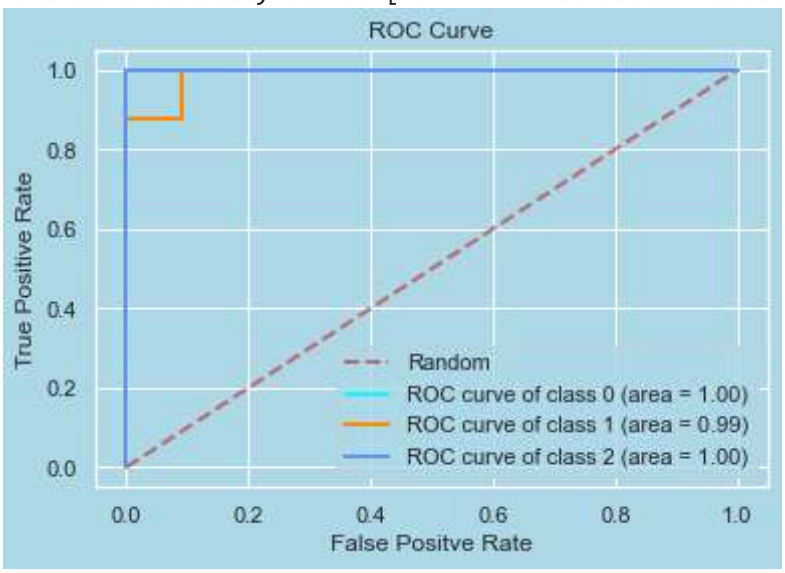
3. Evaluate the performance using the following metric on the test set

a. Confusion matrix
b. Overall and class-wise accuracy
c. ROC curve, AUC

In [ ]:
```
confusionMatrixAndAccuracyReport(Y_test, Y_pred)

rocCurveAndAucAnalysis(y_test_bin, y_score)
```




```
Overall Accuracy Score: 0.967
Classwise Accuracy Score: [1.         0.96666667 0.96666667]
```

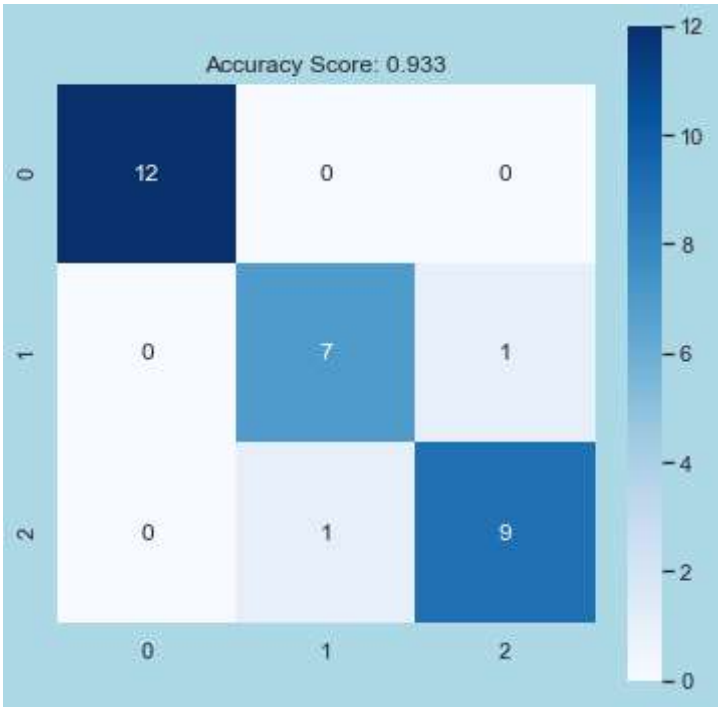



```
Area Under the curve for class 0 : 1.0
Area Under the curve for class 1 : 0.9886363636363636
Area Under the curve for class 2 : 1.0
```

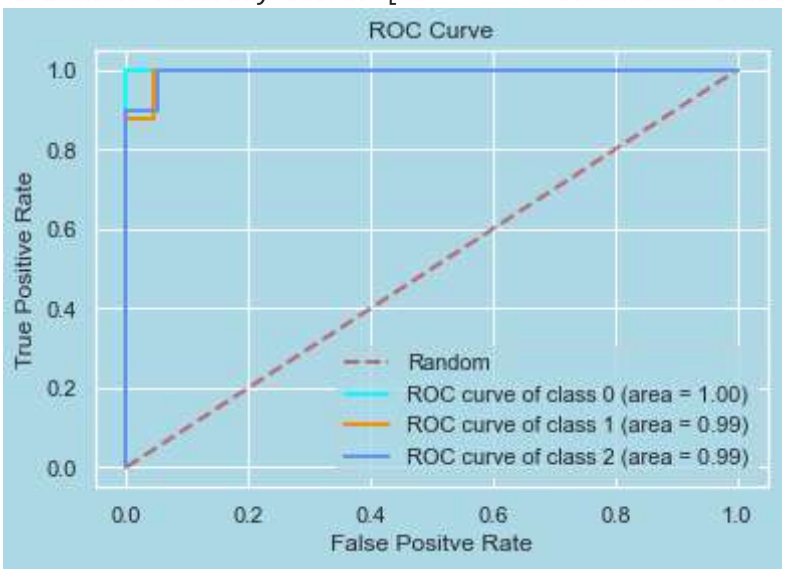
4. Use any library (e.g. scikit-learn) and repeat 1 to 3

In [ ]:

```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(X_train, Y_train)
Y_pred_lib = model.predict(X_test)
y_score_lib = model.predict_proba(X_test)


confusionMatrixAndAccuracyReport(Y_test, Y_pred_lib)

rocCurveAndAucAnalysis(y_test_bin, y_score_lib)
```




```
Overall Accuracy Score: 0.933
Classwise Accuracy Score: [1.         0.93333333 0.93333333]
```




```
Area Under the curve for class 0 : 1.0
Area Under the curve for class 1 : 0.9943181818181819
Area Under the curve for class 2 : 0.995
```

In [ ]:

```python
def calculateBayesRisk(lossMatrix, class_prob):
    no_of_classses = len(class_prob)
    alpha = np.zeros([no_of_classses,no_of_classses])
    for x in range(no_of_classses):
        for r in range(no_of_classses):
            alpha[x][r]=((lossMatrix[x][r] * 1 * class_prob[r])/class_prob[r])

    max_r = np.argmax(alpha,axis=0)

    for x in range(no_of_classses):
        print(
        f"The optimized value of r_x{x+1} is R{max_r[x]+1}_alpha_x{x+1} which is {a

    risk = 0
    for x in range(no_of_classses):
        risk+= class_prob[x]*alpha[x][max_r[x]]

    return risk
```

5. Compare and comment on the performance of the results of the classifier in 2 and 4

Classifier in 2 (from scratch) and Classifier in 4 (from sklearn) both performed very well and similar way.

- *Comparing accuracy* :

Classifier in 2 (from scratch) Overall Accuracy Score: 0.967 Classwise Accuracy Score: [1. 0.96666667 0.96666667]

Classifier in 4 (from sklearn) Overall Accuracy Score: 0.933 Classwise Accuracy Score: [1. 0.93333333 0.93333333]

- *Comparing AUC* :

Classifier in 2 (from scratch) for class 0 : 1.0 for class 1 : 0.9886363636363636 for class 2 : 1.0

Classifier in 4 (from sklearn) for class 0 : 1.0 for class 1 : 0.9943181818181819 for class 2 : 0.995

Classifier in 2 (from scratch) has performed slightly better than the Classifier in 4 (from sklearn) by comparing both matrics.

6. Calculate the Bayes risk. Consider,

λ =
2 4 6
1 2 3
6 4 1

Where λ is a loss function and rows and columns corresponds to classes (ci) and actions (aj) respectively, e.g. λ(a3/ c2) =4.

In [ ]:
```
lambda_ = np.array(
    [[2, 1, 6],
     [4, 2, 4],
     [6, 3, 1]]
)


bayes_risk = calculateBayesRisk(lambda_, modelFromScratch.prior)
print(f"bayes risk {bayes_risk:.4f}")
```

```
The optimized value of r_x1 is R3_alpha_x1 which is 6.0
The optimized value of r_x2 is R3_alpha_x2 which is 4.0
The optimized value of r_x3 is R1_alpha_x3 which is 6.0
bayes risk 5.3000
```

Question 2.

You are requested to use the dataset from here. (Recall the tutorial video shared with you earlier to download the Kaggle datasets). Perform a quick overview of the dataset by using

data wrangling techniques. Now perform the following operations to clean the dataset and perform classification tasks: #

1. Remove links from the dataset.
2. Remove special characters or symbols from the dataset.
3. Remove numbers or alphanumerical characters from the dataset.
4. Check if your dataset contains sufficient features that are required for the operations.
5. Produce the following visualizations:
    a. Spam vs Ham in the dataset
    b. Word clouds for spam and ham
    c. Find dependencies between different features using correlation matrix, pair
    plots, and distributions of different features.
6. Perform the following tasks as part of feature extraction:
    a. Tokenization
    b. Lemmatization
    c. Vectorization
    d. TF-IDF weighting
7. Find top N ham and spam words in messages and visualize them either by plot or
word cloud.
8. Train-Test Split and perform the classification task using the Naive Bayes classification
model.
    a. Find probabilities of the top 30 words.
    b. What problem does smoothing handle? Use the smoothing technique with
    naive Bayes to predict the sentences in the test split.
    c. Draw confusion matrix, ROC, AUC. (You can use the sklearn library)

- Data load

In [ ]:
```
import pandas as pd
dataset1 = pd.read_csv('data-ques-2/lingSpam.csv')
dataset2 = pd.read_csv('data-ques-2/enronSpamSubset.csv')
dataset3 = pd.read_csv('data-ques-2/completeSpamAssassin.csv')

dataset1.head()
```

Out[ ]:

	Unnamed: 0	Body	Label
0	0	Subject: great part-time or summer job !\r\n \...	1
1	1	Subject: auto insurance rates too high ?\r\n \...	1
2	2	Subject: do want the best and economical hunti...	1
3	3	Subject: email 57 million people for $ 99\r\n ...	1
4	4	Subject: do n't miss these !\r\n \r\n attentio...	1

In [ ]:
```
dataset2.head()
```

Out[ ]:

	Unnamed: 0	Unnamed: 0.1	Body	Label
0	2469	2469	Subject: stock promo mover : cwtd\r\n * * * ur...	1
1	5063	5063	Subject: are you listed in major search engine...	1
2	12564	12564	Subject: important information thu , 30 jun 20...	1
3	2796	2796	Subject: = ? utf - 8 ? q ? bask your life with...	1
4	1468	1468	Subject: " bidstogo " is places to go , things...	1

In [ ]:
```
dataset3.head()
```

Out[ ]:

	Unnamed: 0	Body	Label
0	0	\r\nSave up to 70% on Life Insurance.\r\nWhy S...	1
1	1	1) Fight The Risk of Cancer!\r\nhttp://www.adc...	1
2	2	1) Fight The Risk of Cancer!\r\nhttp://www.adc...	1
3	3	##...	1
4	4	I thought you might like these:\r\n1) Slim Dow...	1

- Drop unnecesary data column & Merge 3 data files

In [ ]:
```
#
dataset1.drop("Unnamed: 0",inplace=True,axis=1)
dataset2.drop(["Unnamed: 0","Unnamed: 0.1"],inplace=True,axis=1)
dataset3.drop("Unnamed: 0",inplace=True,axis=1)

# Merged 3 data files
finalDataSet = pd.concat([dataset1,dataset2,dataset3],axis=0)
del dataset1
del dataset2
del dataset3
print(f'Final Data shape after combining 3 file: {finalDataSet.shape}')
finalDataSet.head()
```

```
Final Data shape after combining 3 file: (18651, 2)
```

Out[ ]:

	Body	Label
0	Subject: great part-time or summer job !\r\n \...	1
1	Subject: auto insurance rates too high ?\r\n \...	1
2	Subject: do want the best and economical hunti...	1
3	Subject: email 57 million people for $ 99\r\n ...	1
4	Subject: do n't miss these !\r\n \r\n attentio...	1

- Dropped null values

In [ ]:
```
finalDataSet.dropna(inplace=True)
finalDataSet.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18650 entries, 0 to 6045
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Body    18650 non-null  object
 1   Label   18650 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 437.1+ KB
```

1. Remove links from the dataset.

In [ ]:

```
import re
x_clnd_link = finalDataSet["Body"]
print('Data before link clean =>')
print(finalDataSet[finalDataSet['Body'].str.contains(r'https?://\S+|www\.\S+')])

x_clnd_link = [re.sub(r'https?://\S+|www\.\S+', '', text) for text in x_clnd_link]
x_clnd_link = [re.sub(r'https?:\/\/.*[\r\n]*','',text) for text in x_clnd_link]


print('Data after link clean =>')
print(x_clnd_link[0])
```

```
Data before link clean =>
                                                   Body  Label
1     1) Fight The Risk of Cancer!\r\nhttp://www.adc...      1
2     1) Fight The Risk of Cancer!\r\nhttp://www.adc...      1
3     ##############################################...      1
4     I thought you might like these:\r\n1) Slim Dow...      1
6     Help wanted.  We are a 14 year old fortune 500...      1
...                                                 ...    ...
6033  ----------------------------------------------...      0
6034  EFFector       Vol. 15, No. 35       November ...      0
6039  \r\nWe have extended our Free seat sale until ...      0
6042                      ___           ___           ...      0
6043  IN THIS ISSUE:01. Readers write\r\n02. Extensi...      0

[4178 rows x 2 columns]
Data after link clean =>
Subject: great part-time or summer job !

 * * * * * * * * * * * * * * * * we have display boxes with credit applications that
we need to place in the small owner-operated stores in your area . here is what yo
u do : 1 . introduce yourself to the store owner or manager . 2 . use our 90 % eff
ective script which tells them how this little display box will save their custome
rs hundreds of dollars , be a drawing card for their business , and make them from
$ 5 . 00 to $ 15 . 00 or more for every app sent in . 3 . find a good spot on the
counter , place the box there , and say that nothing more need be done , all you n
eed is his name and address so the company can send him the commission checks . yo
ur compensaation will be $ 10 for every box you place . by becoming a representati
ve you could also earn a commission of $ 10 for each application that came from th
at store . that is of course a much more profitable plan , as it will pay you for
months or years for a very small effort . call 1-888 - 703-5390 code 3 24 hours to
receive the details ! ! * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * to be removed from o
ur mailing list , type : b2998 @ hotmail . com in the ( to : ) area and ( remove )
in the subject area of a new e - mail and send . * * * * * * * * * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

2. Remove special characters or symbols from the dataset.

In [ ]:
```
x_cleaned = [re.sub("[^a-zA-Z0-9]"," ",text) for text in x_clnd_link]
del x_clnd_link
print(x_cleaned[0])
```

```
Subject  great part time or summer job                                    we hav
e display boxes with credit applications that we need to place in the small owner
operated stores in your area   here is what you do   1   introduce yourself to the
store owner or manager   2   use our 90   effective script which tells them how th
is little display box will save their customers hundreds of dollars   be a drawing
card for their business   and make them from   5   00 to   15   00 or more for eve
ry app sent in   3   find a good spot on the counter   place the box there   and s
ay that nothing more need be done   all you need is his name and address so the co
mpany can send him the commission checks   your compensaation will be   10 for eve
ry box you place   by becoming a representative you could also earn a commission o
f   10 for each application that came from that store   that is of course a much m
ore profitable plan   as it will pay you for months or years for a very small effo
rt   call 1 888   703 5390 code 3 24 hours to receive the details
to be removed from our mailing list   type   b2998   hotmail   com in the   to
area and   remove   in the subject area of a new e   mail and send
```

3. Remove numbers or alphanumerical characters from the dataset.

In [ ]:
```
x_cleaned = [text.lower() for text in x_cleaned]
x_cleaned = [re.sub("[^a-z]"," ",text) for text in x_cleaned]
x_cleaned = [" ".join(text.split()) for text in x_cleaned]

print(x_cleaned[0])
```

```
subject great part time or summer job we have display boxes with credit applicatio
ns that we need to place in the small owner operated stores in your area here is w
hat you do introduce yourself to the store owner or manager use our effective scri
pt which tells them how this little display box will save their customers hundreds
of dollars be a drawing card for their business and make them from to or more for
every app sent in find a good spot on the counter place the box there and say that
nothing more need be done all you need is his name and address so the company can
send him the commission checks your compensaation will be for every box you place
by becoming a representative you could also earn a commission of for each applicat
ion that came from that store that is of course a much more profitable plan as it
will pay you for months or years for a very small effort call code hours to receiv
e the details to be removed from our mailing list type b hotmail com in the to are
a and remove in the subject area of a new e mail and send
```

5. Produce the following visualizations:

a. Spam vs Ham in the dataset
b. Word clouds for spam and ham
c. Find dependencies between different features using correlation matrix, pair
plots, and distributions of different features.

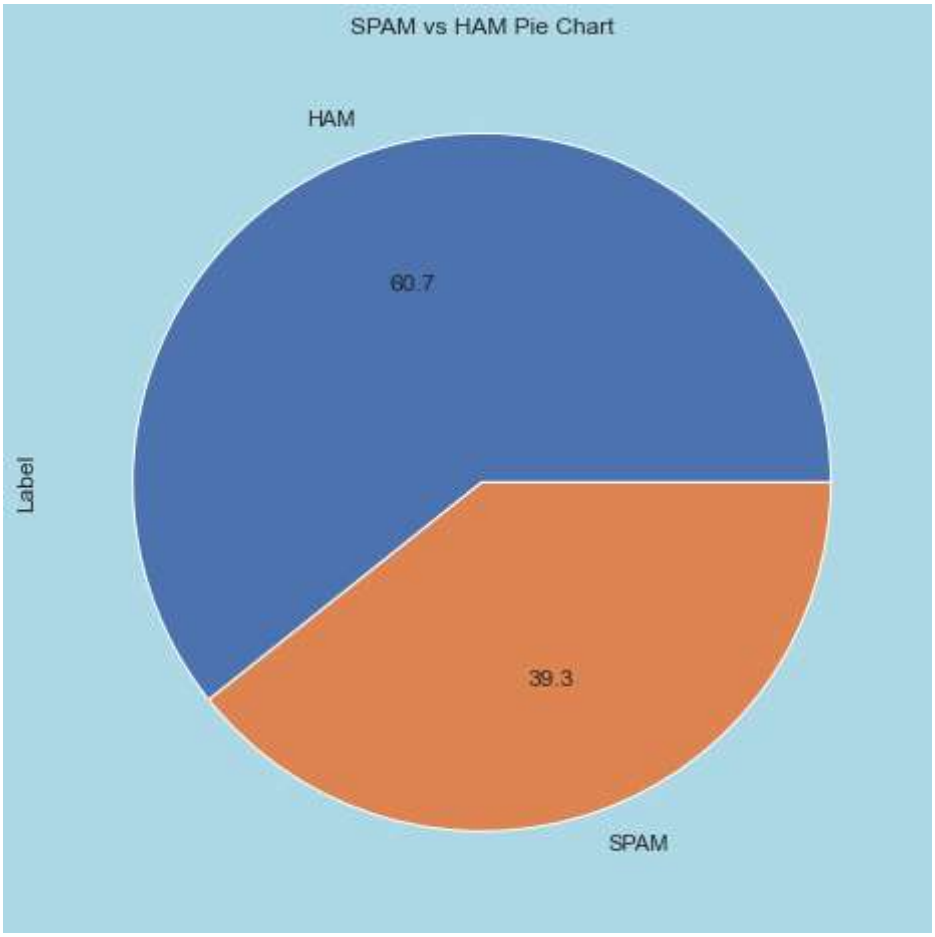
In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(rc={'axes.facecolor': 'lightblue', 'figure.facecolor': 'lightblue'})
```

4. Check if your dataset contains sufficient features that are required for the operations.

With current dataset, accuracy of 96% has been achieved. Though features like message length, usage of stopwords, capital letters, usage of unique word may be used to increase accuracy more.

5.a. Spam vs Ham in the dataset

In [ ]:
```
plot = finalDataSet["Label"].value_counts().plot(kind="pie",labels = ["HAM","SPAM"]
```



5.b. Word clouds for spam and ham

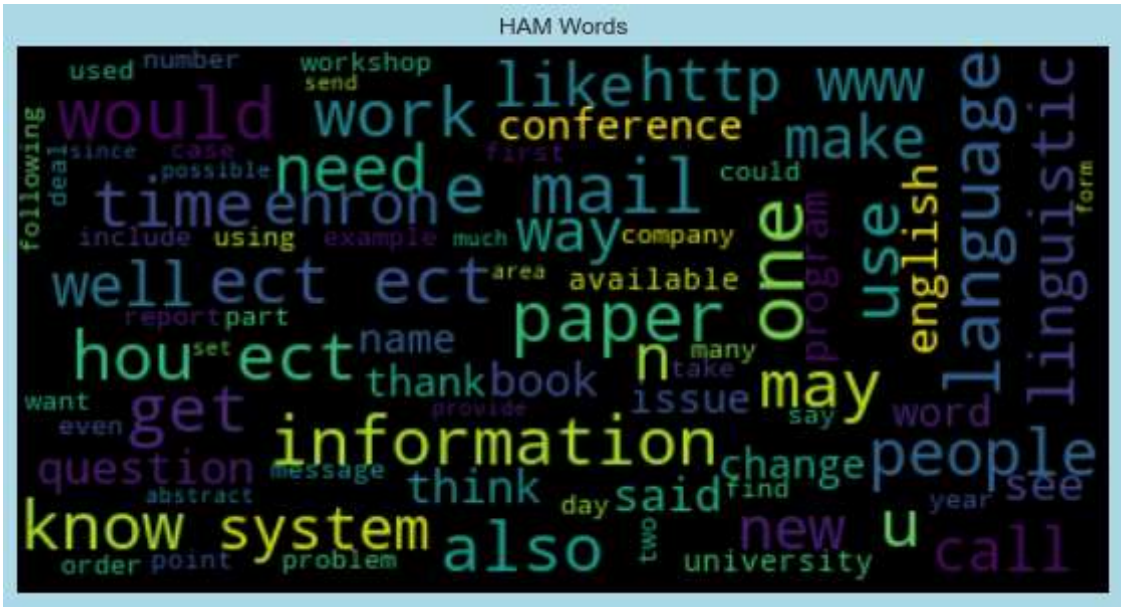
In [ ]:
```
from wordcloud import WordCloud
import nltk
nltk.download('stopwords')
stopwords = nltk.corpus.stopwords.words("english")
stopwords.append('subject')

def showWordCloud(data,title):
    wordcloud = WordCloud(stopwords = set(stopwords),max_font_size=30,max_words=80
    plt.figure(figsize=(12,5))
    plt.imshow(wordcloud,interpolation='bilinear')
    plt.axis("off")
    plt.title(title)
    plt.show()
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\debon\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

- Ham Word Cloud

In [ ]:
```
ham =""

for i in range(len(x_cleaned)) :
    line = x_cleaned[i]
    if finalDataSet["Label"].values[i]==0:
```

```
        ham += line

showWordCloud(ham,"HAM Words")
```



- Spam Word Cloud

In [ ]:
```
ham =""

for i in range(len(x_cleaned)) :
    line = x_cleaned[i]
    if finalDataSet["Label"].values[i]==1:
        ham += line

showWordCloud(ham,"Spam Words")
```

6.a. Tokenization

In [ ]:
```
nltk.download('punkt')
x_tokenized = [nltk.word_tokenize(text) for text in x_cleaned]
del x_cleaned
print(x_tokenized[0])
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\debon\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
['subject', 'great', 'part', 'time', 'or', 'summer', 'job', 'we', 'have', 'displa
y', 'boxes', 'with', 'credit', 'applications', 'that', 'we', 'need', 'to', 'plac
e', 'in', 'the', 'small', 'owner', 'operated', 'stores', 'in', 'your', 'area', 'he
re', 'is', 'what', 'you', 'do', 'introduce', 'yourself', 'to', 'the', 'store', 'ow
ner', 'or', 'manager', 'use', 'our', 'effective', 'script', 'which', 'tells', 'the
m', 'how', 'this', 'little', 'display', 'box', 'will', 'save', 'their', 'customer
s', 'hundreds', 'of', 'dollars', 'be', 'a', 'drawing', 'card', 'for', 'their', 'bu
siness', 'and', 'make', 'them', 'from', 'to', 'or', 'more', 'for', 'every', 'app',
'sent', 'in', 'find', 'a', 'good', 'spot', 'on', 'the', 'counter', 'place', 'the',
'box', 'there', 'and', 'say', 'that', 'nothing', 'more', 'need', 'be', 'done', 'al
l', 'you', 'need', 'is', 'his', 'name', 'and', 'address', 'so', 'the', 'company',
'can', 'send', 'him', 'the', 'commission', 'checks', 'your', 'compensaation', 'wil
l', 'be', 'for', 'every', 'box', 'you', 'place', 'by', 'becoming', 'a', 'represent
ative', 'you', 'could', 'also', 'earn', 'a', 'commission', 'of', 'for', 'each', 'a
pplication', 'that', 'came', 'from', 'that', 'store', 'that', 'is', 'of', 'cours
e', 'a', 'much', 'more', 'profitable', 'plan', 'as', 'it', 'will', 'pay', 'you',
'for', 'months', 'or', 'years', 'for', 'a', 'very', 'small', 'effort', 'call', 'co
de', 'hours', 'to', 'receive', 'the', 'details', 'to', 'be', 'removed', 'from', 'o
ur', 'mailing', 'list', 'type', 'b', 'hotmail', 'com', 'in', 'the', 'to', 'area',
'and', 'remove', 'in', 'the', 'subject', 'area', 'of', 'a', 'new', 'e', 'mail', 'a
nd', 'send']
```

6.b. Lemmatization

In [ ]:
```
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer
lemma = WordNetLemmatizer()

x_lemmatized = [[lemma.lemmatize(word) for word in text] for text in x_tokenized]
del x_tokenized
print(x_lemmatized[0])
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\debon\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
['subject', 'great', 'part', 'time', 'or', 'summer', 'job', 'we', 'have', 'displa
y', 'box', 'with', 'credit', 'application', 'that', 'we', 'need', 'to', 'place',
'in', 'the', 'small', 'owner', 'operated', 'store', 'in', 'your', 'area', 'here',
'is', 'what', 'you', 'do', 'introduce', 'yourself', 'to', 'the', 'store', 'owner',
'or', 'manager', 'use', 'our', 'effective', 'script', 'which', 'tell', 'them', 'ho
w', 'this', 'little', 'display', 'box', 'will', 'save', 'their', 'customer', 'hund
red', 'of', 'dollar', 'be', 'a', 'drawing', 'card', 'for', 'their', 'business', 'a
nd', 'make', 'them', 'from', 'to', 'or', 'more', 'for', 'every', 'app', 'sent', 'i
n', 'find', 'a', 'good', 'spot', 'on', 'the', 'counter', 'place', 'the', 'box', 't
here', 'and', 'say', 'that', 'nothing', 'more', 'need', 'be', 'done', 'all', 'yo
u', 'need', 'is', 'his', 'name', 'and', 'address', 'so', 'the', 'company', 'can',
'send', 'him', 'the', 'commission', 'check', 'your', 'compensaation', 'will', 'b
e', 'for', 'every', 'box', 'you', 'place', 'by', 'becoming', 'a', 'representativ
e', 'you', 'could', 'also', 'earn', 'a', 'commission', 'of', 'for', 'each', 'appli
cation', 'that', 'came', 'from', 'that', 'store', 'that', 'is', 'of', 'course',
'a', 'much', 'more', 'profitable', 'plan', 'a', 'it', 'will', 'pay', 'you', 'for',
'month', 'or', 'year', 'for', 'a', 'very', 'small', 'effort', 'call', 'code', 'hou
r', 'to', 'receive', 'the', 'detail', 'to', 'be', 'removed', 'from', 'our', 'maili
ng', 'list', 'type', 'b', 'hotmail', 'com', 'in', 'the', 'to', 'area', 'and', 'rem
ove', 'in', 'the', 'subject', 'area', 'of', 'a', 'new', 'e', 'mail', 'and', 'sen
d']
```

- Removing Stopwords

In [ ]:
```
x_prepared = [[word for word in text if word not in stopwords] for text in x_lemma
del x_lemmatized
clubbed_text = [' '.join(x) for x in x_prepared]
del x_prepared
print(clubbed_text[0])
finalDataSet['Body'] = clubbed_text
```

```
great part time summer job display box credit application need place small owner o
perated store area introduce store owner manager use effective script tell little
display box save customer hundred dollar drawing card business make every app sent
find good spot counter place box say nothing need done need name address company s
end commission check compensaation every box place becoming representative could a
lso earn commission application came store course much profitable plan pay month y
ear small effort call code hour receive detail removed mailing list type b hotmail
com area remove area new e mail send
```

6.c. Vectorization

In [ ]:
```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(max_features=20000)
x_vectorized = vectorizer.fit_transform(clubbed_text).toarray()


print(x_vectorized.shape)
print(x_vectorized[:2])
```

```
(18650, 20000)
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

6.d. TF-IDF weighting

In [ ]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
count_vect = TfidfVectorizer(analyzer='word', ngram_range=(1, 3),
                             stop_words="english", min_df = 10, max_df = 0.95,
                             max_features = 20000)
## fitting and transforming data
count_vect.fit(clubbed_text)

## getting train and test features
x_vectorized = count_vect.transform(clubbed_text)

print(x_vectorized.shape)
print(x_vectorized[:2])
```

```
(18650, 20000)
  (0, 19888)    0.05777154647312691
  (0, 18753)    0.05538551886331337
  (0, 18299)    0.07420379186004081
  (0, 17867)    0.04696855195201727
  (0, 17576)    0.15237101914977116
  (0, 17574)    0.08170050556293512
  (0, 17230)    0.09609541754382064
  (0, 16979)    0.28201980871032367
  (0, 16702)    0.10609704635410663
  (0, 16354)    0.16414788497785293
  (0, 15900)    0.05948920730987377
  (0, 15818)    0.1202759056872621
  (0, 15648)    0.10983899750703363
  (0, 15544)    0.06937257149758379
  (0, 15525)    0.07667818019080268
  (0, 14904)    0.09666738617923042
  (0, 14762)    0.11696149960582213
  (0, 14761)    0.11553482180145995
  (0, 14756)    0.07767755499472914
  (0, 14742)    0.07415330991150865
  (0, 14406)    0.06957128946487373
  (0, 13830)    0.10893287418574395
  (0, 13217)    0.07518840573427858
  (0, 13191)    0.208355454168776
  (0, 12818)    0.07673558286294009
  :     :
  (1, 3736)     0.11204390814378991
  (1, 3342)     0.0738753371414525
  (1, 3105)     0.146118243506943
  (1, 3079)     0.0712947298336163
  (1, 2702)     0.06749922867102492
  (1, 2553)     0.05408089781078473
  (1, 2550)     0.09625031410971238
  (1, 2453)     0.06665753350428187
  (1, 2332)     0.06392169854117093
  (1, 2159)     0.06443928361738549
  (1, 2131)     0.0655517941224384
  (1, 1799)     0.05811040389406535
  (1, 1797)     0.08185540939596538
  (1, 1313)     0.0656829820096574
  (1, 1282)     0.08976917676153913
  (1, 1258)     0.04838309281495975
  (1, 1230)     0.17342544595590126
  (1, 752)      0.10355118053501962
  (1, 751)      0.08816488755624474
  (1, 665)      0.11341489087587361
  (1, 511)      0.07468322179355041
  (1, 332)      0.12262296660429385
  (1, 331)      0.11733758799322636
  (1, 330)      0.08263162713392094
  (1, 256)      0.04528160288231057
```

7. Find top N ham and spam words in messages and visualize them either by plot or word cloud.

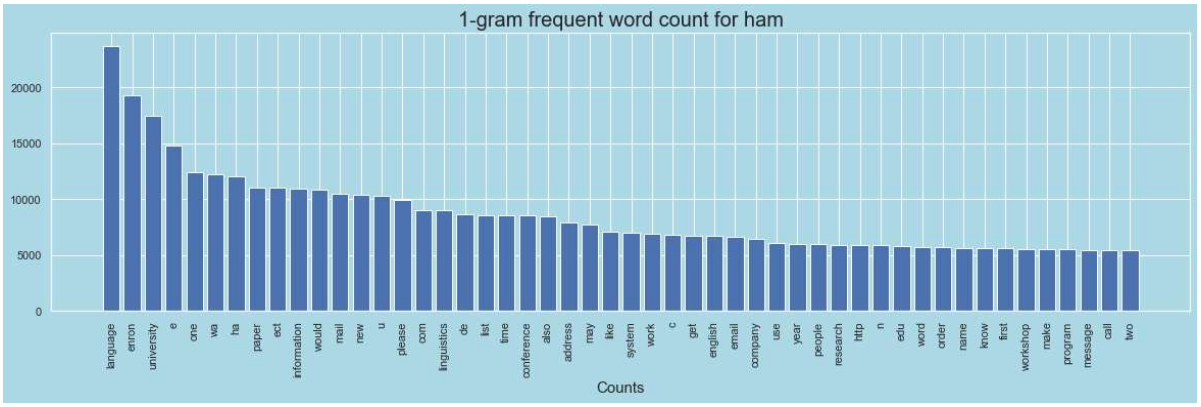
- utility to count words

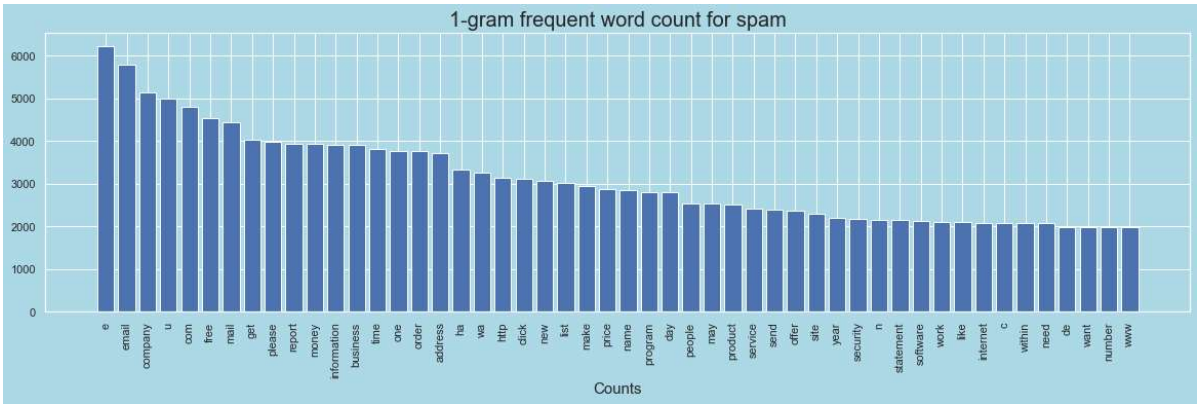
In [ ]:
```
from collections import defaultdict,Counter
def generate_ngrams(text,n_gram):
    word = [token for token in str(text).lower().split(" ") if token !=""]
```

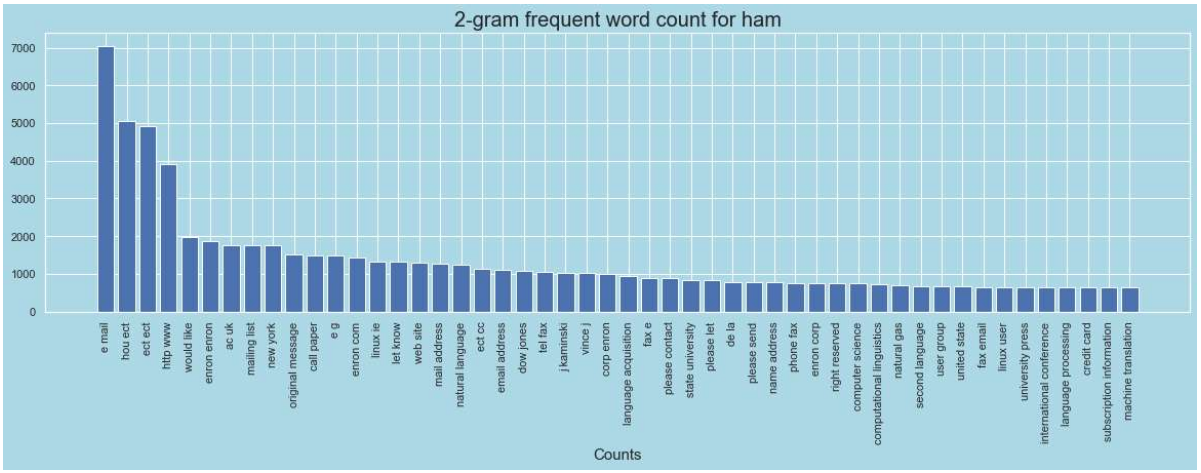
```
    ngrams = zip(*[word[i:] for i in range(n_gram)])
    return [" ".join(ngram) for ngram in ngrams]
def freq_ngrams(text,n_gram):
    freq_dict = defaultdict(int)
    for sent in text:
        for w in generate_ngrams(sent,n_gram):
            freq_dict[w] += 1
    freq_dict = dict(freq_dict)
    qstn_txt = pd.DataFrame(sorted(freq_dict.items(),key=lambda x: x[1],reverse=Tru
    qstn_txt.columns = ["word","wordcount"]
    return qstn_txt
def plot_freq_ngrams(text,n_gram,range,title):
    qstn_txt = freq_ngrams(text,n_gram)
    fig = plt.figure(figsize=(20,5))
    plt.bar(qstn_txt.word[0:range],qstn_txt.wordcount[0:range])
    plt.xticks(rotation=90)
    plt.xlabel("Words",fontsize=15)
    plt.xlabel("Counts",fontsize=15)
    plt.title(title,fontsize=20)
    return plt.show()
```

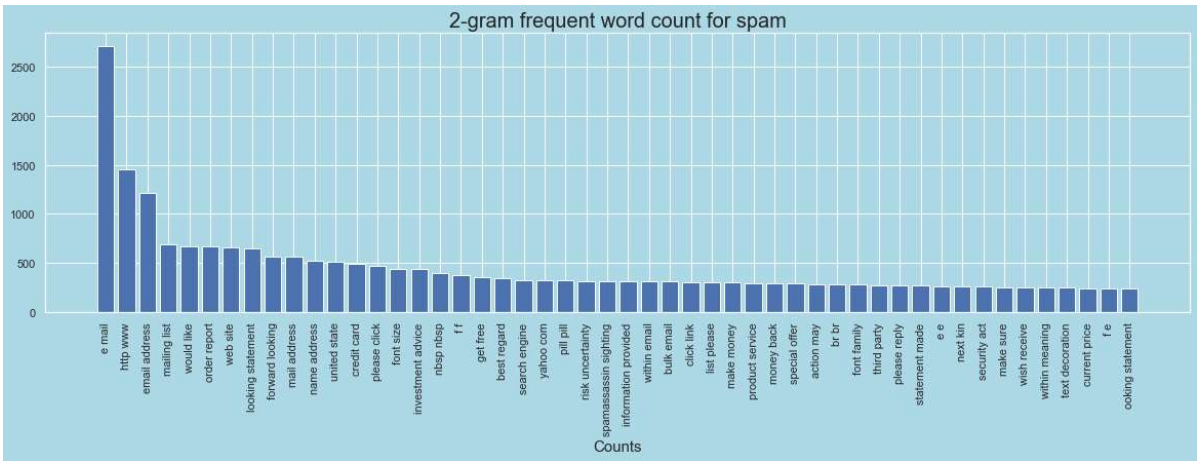
In [ ]:

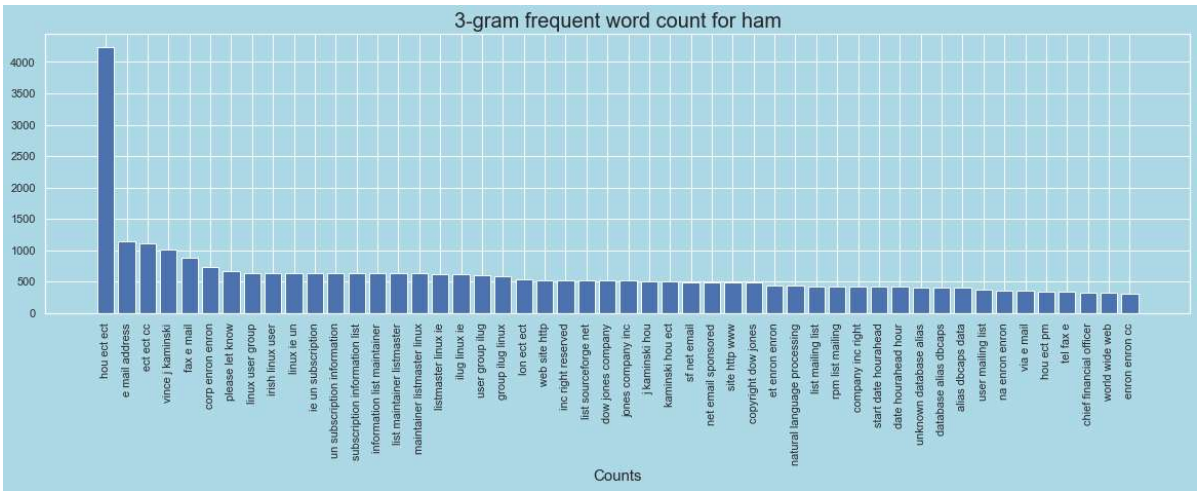
```
for n_g in range(3):
    plot_freq_ngrams(finalDataSet[finalDataSet['Label'] == 0]["Body"], n_gram=n_g+1
            title=f"{n_g+1}-gram frequent word count for ham")
    plot_freq_ngrams(finalDataSet[finalDataSet['Label'] == 1]["Body"], n_gram=n_g+1
            title=f"{n_g+1}-gram frequent word count for spam")
```

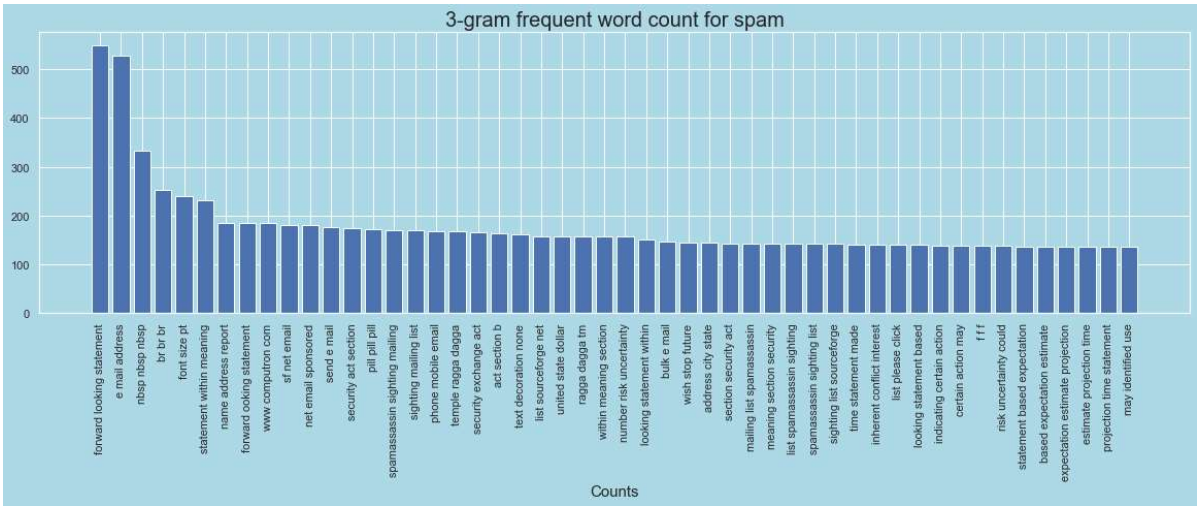





2-gram frequent word count for ham
Counts


2-gram frequent word count for spam
Counts


3-gram frequent word count for ham
Counts


3-gram frequent word count for spam
Counts

- Top 30 Ham

In [ ]:
```
hamCounts = freq_ngrams(finalDataSet[finalDataSet['Label'] == 0]["Body"], n_gram=1
spamCounts = freq_ngrams(finalDataSet[finalDataSet['Label'] == 1]["Body"], n_gram=

print('Top 30 Ham:')
print(hamCounts[:30])
```

```
Top 30 Ham:
           word  wordcount
0      language      23714
1         enron      19273
2    university      17416
3             e      14774
4           one      12421
5            wa      12236
6            ha      12023
7         paper      11028
8           ect      11007
9   information      10909
10        would      10853
11         mail      10471
12          new      10407
13            u      10317
14       please       9973
15          com       9007
16  linguistics       8971
17           de       8681
18         list       8586
19         time       8538
20   conference       8535
21         also       8486
22      address       7876
23          may       7729
24         like       7083
25       system       7010
26         work       6882
27            c       6794
28          get       6730
29      english       6720
```

- Top 30 Spam:

In [ ]:
```
print(spamCounts[:30])
```

	word	wordcount
0	e	6234
1	email	5798
2	company	5145
3	u	4984
4	com	4814
5	free	4530
6	mail	4435
7	get	4026
8	please	3974
9	report	3936
10	money	3929
11	information	3923
12	business	3902
13	time	3816
14	one	3776
15	order	3768
16	address	3726
17	ha	3322
18	wa	3269
19	http	3135
20	click	3110
21	new	3077
22	list	3019
23	make	2952
24	price	2873
25	name	2862
26	program	2805
27	day	2798
28	people	2541
29	may	2535

8. Train-Test Split and perform the classification task using the Naive Bayes classification model.

#

1. Find probabilities of the top 30 words.
2. What problem does smoothing handle? Use the smoothing technique with naive Bayes to predict the sentences in the test split.
3. Draw confusion matrix, ROC, AUC. (You can use the sklearn library)

- Test Train split

In [ ]:
```
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(
    x_vectorized,finalDataSet["Label"],test_size=0.20,random_state=42,stratify=fina
xtrain.shape, xtest.shape, ytrain.shape, ytest.shape
```

Out[ ]:
```
((14920, 20000), (3730, 20000), (14920,), (3730,))
```

- Perform the classification task using the Naive Bayes classification model.

In [ ]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import RandomizedSearchCV
clf_nb = MultinomialNB()
param_grid = {'alpha':[10** x for x in range(-3,3)]}
clf = RandomizedSearchCV(clf_nb,param_grid,cv=5,scoring="accuracy",return_train_sc
```

```
clf.fit(xtrain,ytrain)
print("Best cross-validation score: {:.2f}".format(clf.best_score_))
print("Best parameters: ", clf.best_params_)
```

```
C:\Users\debon\AppData\Roaming\Python\Python39\site-packages\sklearn\model_selecti
on\_search.py:285: UserWarning: The total space of parameters 6 is smaller than n_
iter=10. Running 6 iterations. For exhaustive searches, use GridSearchCV.
  warnings.warn(
Fitting 5 folds for each of 6 candidates, totalling 30 fits
Best cross-validation score: 0.95
Best parameters:  {'alpha': 0.1}
```

- Algo fitting

In [ ]:
```
clf_nb_final = MultinomialNB(alpha=clf.best_params_['alpha'])
clf_nb_final.fit(xtrain,ytrain)


print(f"train accuracy score {clf_nb_final.score(xtrain, ytrain) * 100 : .2f}%")
```

```
train accuracy score  96.41%
```

8.a. Find probabilities of the top 30 words.

In [ ]:
```
freq_ngrams(finalDataSet[finalDataSet['Label'] == 1]["Body"], n_gram=5)
```

Out[ ]:

	word	wordcount
0	br br br br br	234
1	nbsp nbsp nbsp nbsp nbsp	232
2	within meaning section security act	142
3	spamassassin sighting mailing list spamassassin	142
4	sighting mailing list spamassassin sighting	142
...	...	...
592518	send blank email mailto adsub	1
592519	blank email mailto adsub btamail	1
592520	email mailto adsub btamail net	1
592521	mailto adsub btamail net cn	1
592522	adsub btamail net cn remove	1

592523 rows × 2 columns

In [ ]:
```
spam_ngrams = freq_ngrams(finalDataSet[finalDataSet['Label'] == 1]["Body"], n_gram=
spam_ngrams_vector= count_vect.transform(spam_ngrams["word"])

spam_ngrams['proba'] = clf_nb_final.predict_proba(spam_ngrams_vector)[:,1]

print(spam_ngrams)
```

	word	wordcount	proba
0	br br br br br	234	0.861205
1	nbsp nbsp nbsp nbsp nbsp	232	0.960296
2	within meaning section security act	142	0.999434
3	spamassassin sighting mailing list spamassassin	142	0.997451
4	sighting mailing list spamassassin sighting	142	0.998871
5	mailing list spamassassin sighting list	142	0.998471
6	list spamassassin sighting list sourceforge	142	0.999345
7	spamassassin sighting list sourceforge net	142	0.997257
8	forward looking statement within meaning	136	0.999763
9	statement based expectation estimate projection	136	0.999879
10	based expectation estimate projection time	136	0.999844
11	expectation estimate projection time statement	136	0.999885
12	estimate projection time statement made	136	0.999452
13	statement within meaning section security	134	0.999494
14	statement indicating certain action may	134	0.998906
15	security must understood information provided	132	0.998862
16	must understood information provided investment	132	0.998978
17	understood information provided investment advice	132	0.999863
18	deciding trade stock featured within	131	0.999515
19	action may identified use word	130	0.997101
20	statement action may identified use	129	0.999279
21	may identified use word project	128	0.995904
22	identified use word project foresee	128	0.999614
23	use word project foresee expects	128	0.999612
24	meaning section security act section	127	0.999833
25	pertaining investing stock security must	126	0.999828
26	investing stock security must understood	126	0.999675
27	stock security must understood information	126	0.999368
28	security representative deciding trade stock	126	0.999852
29	representative deciding trade stock featured	126	0.999868

c. Draw confusion matrix, ROC, AUC. (You can use the sklearn library)

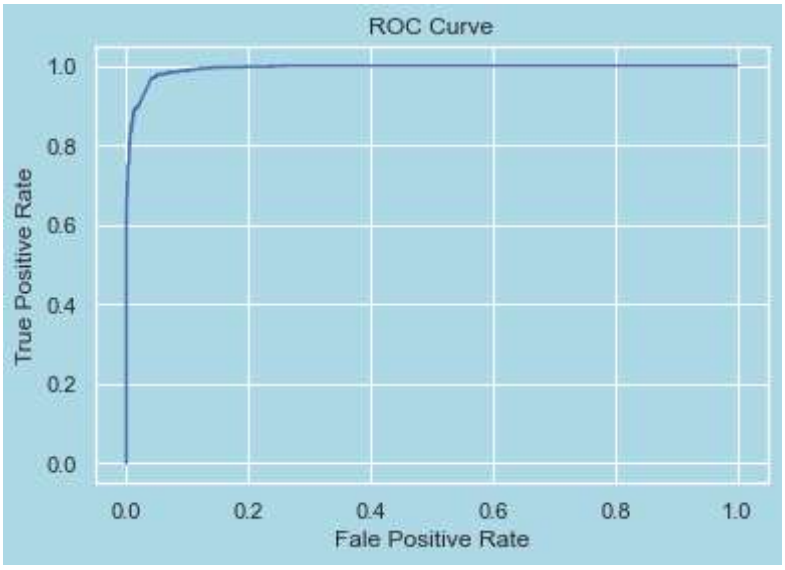
- Accuracy Analysis

In [ ]:
```
from sklearn.metrics import accuracy_score, classification_report,f1_score,confusi
ypred = clf_nb_final.predict(xtest)
ypred_proba = clf_nb_final.predict_proba(xtest)

print(f"test accuracy score: {accuracy_score(ytest, ypred)*100:.2f}%")
print(f"test classification report:\n {classification_report(ytest, ypred)}")
print(f"confusion matrix:\n{confusion_matrix(ytest,ypred)}")
```

```
test accuracy score: 94.83%
test classification report:
              precision    recall  f1-score   support

           0       0.93      0.98      0.96      2264
           1       0.97      0.89      0.93      1466

    accuracy                           0.95      3730
   macro avg       0.95      0.94      0.94      3730
weighted avg       0.95      0.95      0.95      3730

confusion matrix:
[[2229   35]
 [ 158 1308]]
```

- ROC Graph

In [ ]:
```
fpr,tpr,thrs = roc_curve(ytest,ypred_proba[:,1])
plt.plot(fpr,tpr)
plt.xlabel("Fale Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.show()
```



- AUC Calculation

In [ ]:
```
auc_score = auc(fpr, tpr)
print(f"the auc score is {auc_score :.4f}")
```

```
the auc score is 0.9936
```

Question 3:

Download the dataset containing two columns X and Y. Fix an appropriate threshold and consider Y > threshold as 1 otherwise 0. Perform the following tasks. #

1. Create the labels from the given data.
2. Plot the distribution of samples using the histogram.
3. Determine the prior probability for both classes.
4. Determine the class conditional probabilities (likelihood) for the classes.
5. Plot the count of each unique element for each class.
6. Calculate the posterior probability of both classes and plot them.
7. Consider the dataset of question 1 and select any one feature and class label. Repeat
1 to 6.

- Data load

In [ ]:
```
import numpy as np
import pandas as pd
data = pd.read_csv('data-ques-3/Data.csv')
data.head()
```

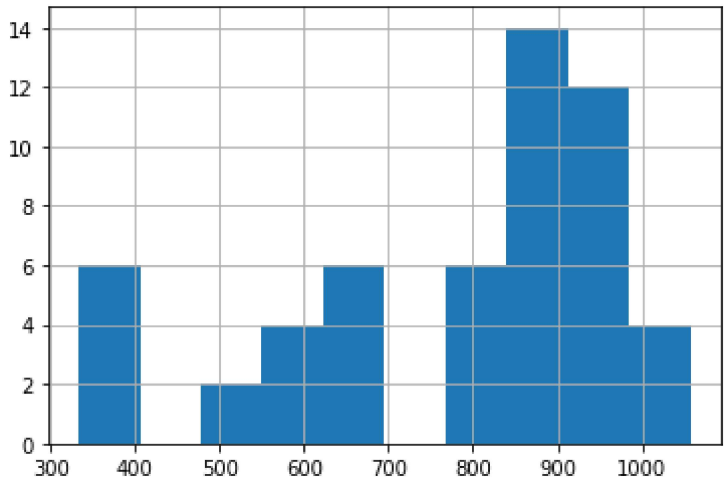
Out[ ]:

	X	Y
0	1.0	550
1	3.0	566
2	4.0	558
3	3.0	784
4	5.0	333

1. Create the labels from the given data.

In [ ]:
```
data['Y'].hist()
```

Out[ ]:
```
<AxesSubplot:>
```



Choosen thresold 700 as there were clear division in histogram at y = 700

In [ ]:
```
data["label"] = data.Y.apply(lambda x: 0 if x <= 700 else 1)
data.head()
```

Out[ ]:

	X	Y	label
0	1.0	550	0
1	3.0	566	0
2	4.0	558	0
3	3.0	784	1
4	5.0	333	0

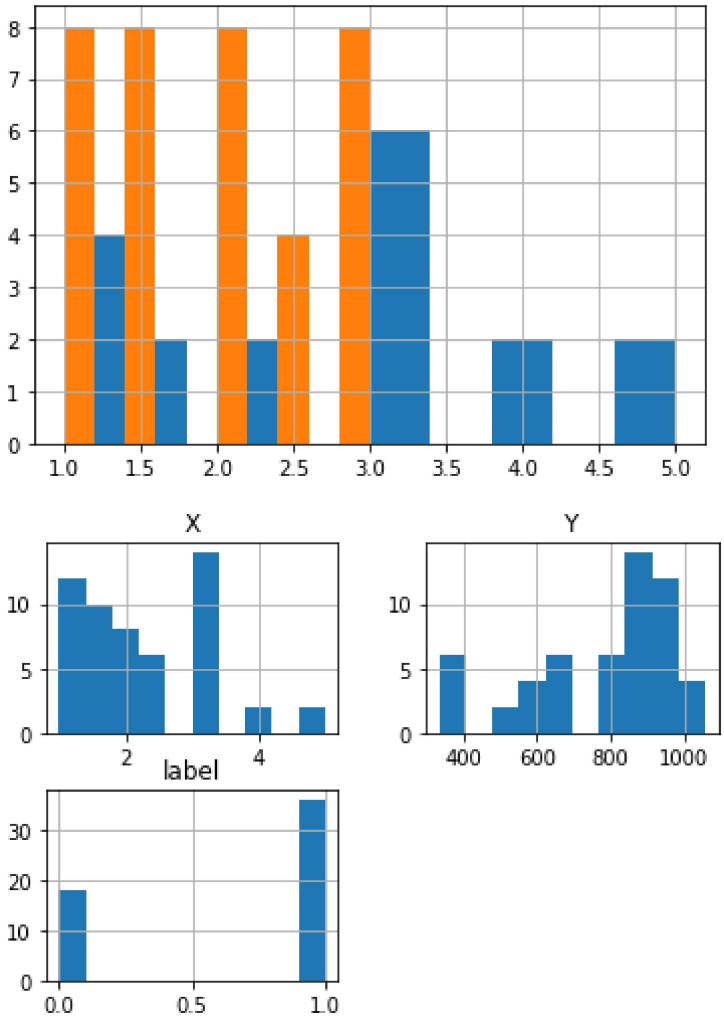
1. Plot the distribution of samples using the histogram.

In [ ]:
```
data[data['label'] == 0]['X'].hist()
data[data['label'] == 1]['X'].hist()
data.hist()
```

```
C:\Users\debon\AppData\Roaming\Python\Python39\site-packages\pandas\plotting\_matplotlib\tools.py:400: MatplotlibDeprecationWarning:
The is_first_col function was deprecated in Matplotlib 3.4 and will be removed two minor releases later. Use ax.get_subplotspec().is_first_col() instead.
  if ax.is_first_col():
```

Out[ ]:

```
array([[<AxesSubplot:title={'center':'X'}>,
        <AxesSubplot:title={'center':'Y'}>],
       [<AxesSubplot:title={'center':'label'}>, <AxesSubplot:>]],
      dtype=object)
```



1. Determine the prior probability for both classes.

In [ ]:

```
classes = data['label'].unique()
classCounts = data['label'].value_counts()
print(f'Class count :\n{classCounts.sort_index()}')

priors = classCounts/data['label'].size
for c in classes:
    print(f'Priors for class {c} : {priors[c]}')
```

```
Class count :
0    18
1    36
Name: label, dtype: int64
Priors for class 0 : 0.3333333333333333
Priors for class 1 : 0.6666666666666666
```

1. Determine the class conditional probabilities (likelihood) for the classes.

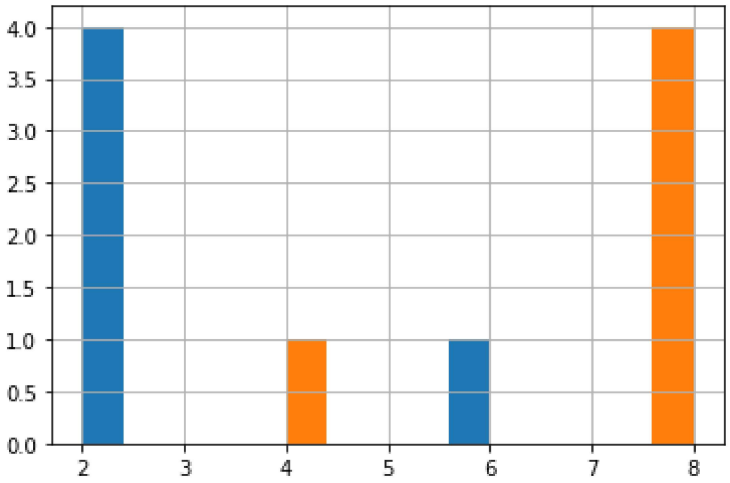
In [ ]:
```
likelihood = data.groupby(data['X'].values).apply(
    lambda x: x['label'].value_counts()/len(x))

print(likelihood)
```

```
1.0  1    0.666667
     0    0.333333
1.5  1    0.800000
     0    0.200000
2.0  1    1.000000
2.5  1    0.666667
     0    0.333333
3.0  1    0.571429
     0    0.428571
4.0  0    1.000000
5.0  0    1.000000
Name: label, dtype: float64
```

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

In [ ]:
```
uniqueCntPerClass = data.groupby(data['label'].values).apply(
    lambda x: x['X'].value_counts())

uniqueCntPerClass[0].hist()
uniqueCntPerClass[1].hist()

print(uniqueCntPerClass)
```

```
0  3.0    6
   1.0    4
   5.0    2
   1.5    2
   4.0    2
   2.5    2
1  3.0    8
   2.0    8
   1.5    8
   1.0    8
   2.5    4
Name: X, dtype: int64
```



1. Calculate the posterior probability of both classes and plot them.

In [ ]:
```
uniqeFeatures = data['X'].unique()
uniqeFeatures.sort()
evidenceProb = data['X'].value_counts()/len(data)
```

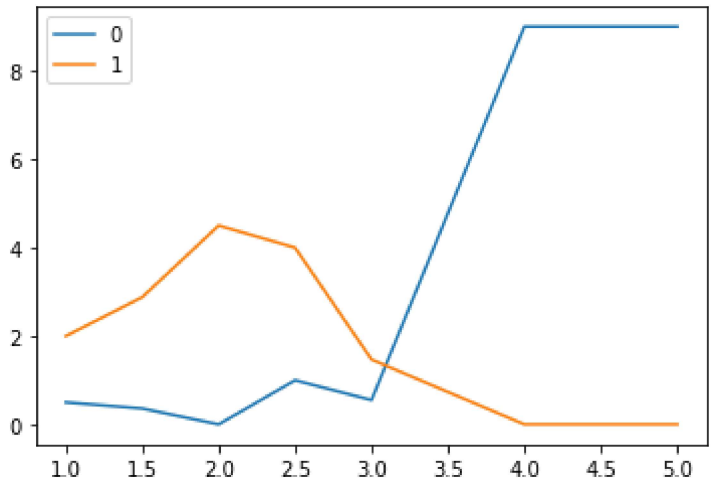
```
posterior = []


for c in classes:
    classPost = []
    for f in uniqeFeatures:
        classPost.append((likelihood[f].get(c, 0)*priors[c])/evidenceProb[f])
    posterior.append(classPost)

print(posterior)
```

```
[[0.5, 0.36000000000000004, 0.0, 1.0, 0.5510204081632653, 9.0, 9.0], [2.0, 2.88000
00000000003, 4.5, 4.0, 1.469387755102041, 0.0, 0.0]]
```

In [ ]:
```
postDf = pd.DataFrame(posterior, columns=uniqeFeatures).T
print(postDf)
postDf.plot()
```

```
           0         1
1.0  0.50000  2.000000
1.5  0.36000  2.880000
2.0  0.00000  4.500000
2.5  1.00000  4.000000
3.0  0.55102  1.469388
4.0  9.00000  0.000000
5.0  9.00000  0.000000
<AxesSubplot:>
```

Out[ ]:



1. Consider the dataset of question 1 and select any one feature and class label. Repeat 1 to 6.

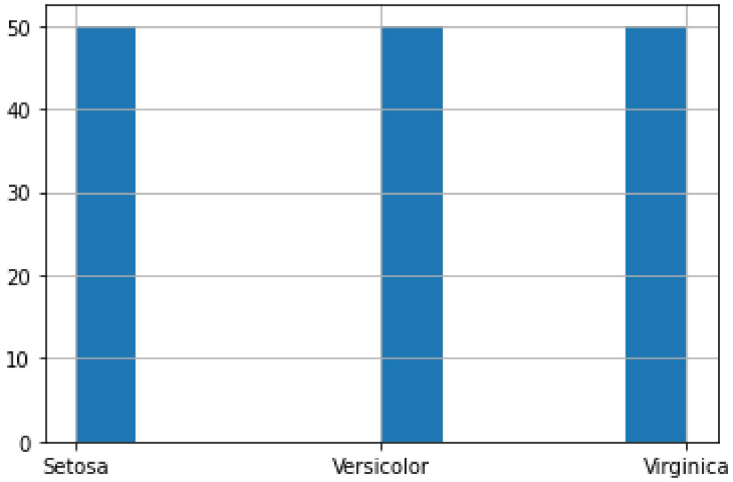
In [ ]:
```
dataQ1 = pd.read_csv('data-ques-1/iris_dataset.csv')
dataQ1.head()
```

Out[ ]:

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa

In [ ]:
```
dataQ1['variety'].hist()
```

Out[ ]:
```
<AxesSubplot:>
```



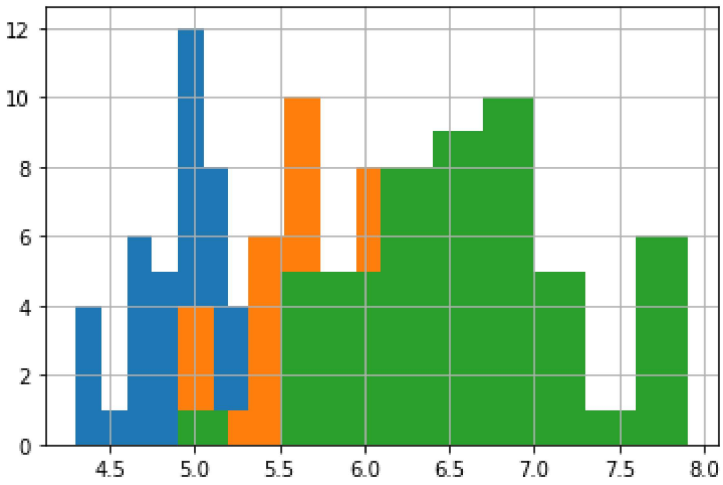
- Selected sepal.length as seceted feature

In [ ]:
```
x = 'sepal.length'
label = 'variety'
```

- Plot the distribution of samples using the histogram.

In [ ]:
```
classes = dataQ1[label].unique()
classCounts = dataQ1[label].value_counts()
print(f'Class count :\n{classCounts.sort_index()}')

for cls in classes:
    dataQ1[dataQ1[label]==cls][x].hist()
```

```
Class count :
Setosa        50
Versicolor    50
Virginica     50
Name: variety, dtype: int64
```



- Determine the prior probability for both classes

In [ ]:
```
priors = classCounts/dataQ1[label].size
for c in classes:
    print(f'Priors for class {c} : {priors[c]}')
```

```
Priors for class Setosa : 0.3333333333333333
Priors for class Versicolor : 0.3333333333333333
Priors for class Virginica : 0.3333333333333333
```

- Determine the class conditional probabilities (likelihood) for the classes.

In [ ]:
```
likelihood = dataQ1.groupby(dataQ1[x].values).apply(
    lambda x: x[label].value_counts()/len(x))

print(likelihood)
```

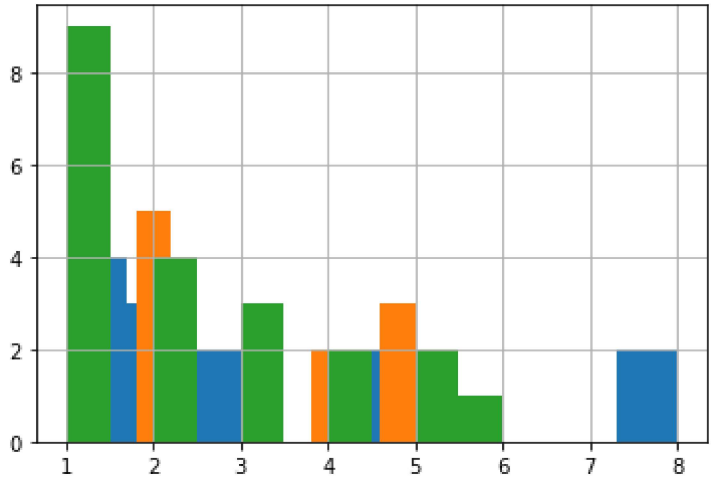
```
4.3  Setosa        1.000000
4.4  Setosa        1.000000
4.5  Setosa        1.000000
4.6  Setosa        1.000000
4.7  Setosa        1.000000
4.8  Setosa        1.000000
4.9  Setosa        0.666667
     Versicolor    0.166667
     Virginica     0.166667
5.0  Setosa        0.800000
     Versicolor    0.200000
5.1  Setosa        0.888889
     Versicolor    0.111111
5.2  Setosa        0.750000
     Versicolor    0.250000
5.3  Setosa        1.000000
5.4  Setosa        0.833333
     Versicolor    0.166667
5.5  Versicolor    0.714286
     Setosa        0.285714
5.6  Versicolor    0.833333
     Virginica     0.166667
5.7  Versicolor    0.625000
     Setosa        0.250000
     Virginica     0.125000
5.8  Versicolor    0.428571
     Virginica     0.428571
     Setosa        0.142857
5.9  Versicolor    0.666667
     Virginica     0.333333
6.0  Versicolor    0.666667
     Virginica     0.333333
6.1  Versicolor    0.666667
     Virginica     0.333333
6.2  Versicolor    0.500000
     Virginica     0.500000
6.3  Virginica     0.666667
     Versicolor    0.333333
6.4  Virginica     0.714286
     Versicolor    0.285714
6.5  Virginica     0.800000
     Versicolor    0.200000
6.6  Versicolor    1.000000
6.7  Virginica     0.625000
     Versicolor    0.375000
6.8  Virginica     0.666667
     Versicolor    0.333333
6.9  Virginica     0.750000
     Versicolor    0.250000
7.0  Versicolor    1.000000
7.1  Virginica     1.000000
7.2  Virginica     1.000000
7.3  Virginica     1.000000
7.4  Virginica     1.000000
7.6  Virginica     1.000000
7.7  Virginica     1.000000
7.9  Virginica     1.000000
Name: variety, dtype: float64
```

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

In [ ]:
```
uniqueCntPerClass = dataQ1.groupby(dataQ1[label].values).apply(
    lambda a: a[x].value_counts())
```

```
for c in classes:
    uniqueCntPerClass[c].hist()

print(uniqueCntPerClass)
```

```
Setosa      5.0    8
            5.1    8
            4.8    5
            5.4    5
            4.9    4
            4.6    4
            5.2    3
            4.4    3
            4.7    2
            5.7    2
            5.5    2
            4.5    1
            5.8    1
            4.3    1
            5.3    1
Versicolor  5.6    5
            5.7    5
            5.5    5
            6.0    4
            6.1    4
            6.3    3
            6.7    3
            5.8    3
            6.2    2
            5.0    2
            6.4    2
            6.6    2
            5.9    2
            5.2    1
            5.4    1
            5.1    1
            7.0    1
            6.8    1
            4.9    1
            6.5    1
            6.9    1
Virginica   6.3    6
            6.7    5
            6.4    5
            7.7    4
            6.5    4
            6.9    3
            7.2    3
            5.8    3
            6.1    2
            6.8    2
            6.2    2
            6.0    2
            5.6    1
            7.3    1
            7.6    1
            7.9    1
            7.1    1
            5.7    1
            4.9    1
            5.9    1
            7.4    1
Name: sepal.length, dtype: int64
```



- Calculate the posterior probability of both classes and plot them.

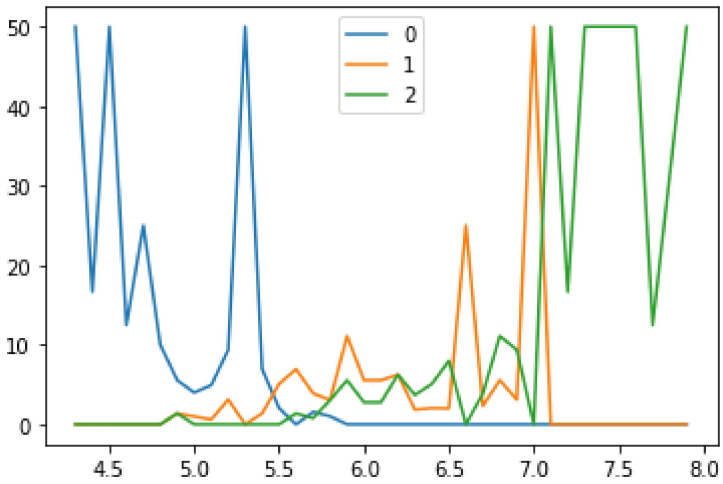
In [ ]:
```
uniqeFeatures = dataQ1[x].unique()
uniqeFeatures.sort()
evidenceProb = dataQ1[x].value_counts()/len(dataQ1)
posterior = []


for c in classes:
    classPost = []
    for f in uniqeFeatures:
        classPost.append((likelihood[f].get(c, 0)*priors[c])/evidenceProb[f])
    posterior.append(classPost)

print(posterior)
```

```
[[49.99999999999999, 16.666666666666664, 49.99999999999999, 12.499999999999998, 2
4.999999999999996, 10.0, 5.555555555555555, 4.0, 4.938271604938271, 9.375, 49.9999
9999999999, 6.944444444444445, 2.040816326530612, 0.0, 1.5624999999999998, 1.02040
8163265306, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.3888888888888888, 1.0,
0.6172839506172839, 3.1249999999999996, 0.0, 1.3888888888888888, 5.102040816326530
5, 6.944444444444445, 3.9062499999999996, 3.061224489795918, 11.11111111111111, 5.
555555555555555, 5.555555555555555, 6.249999999999999, 1.8518518518518519, 2.04081
6326530612, 2.0, 24.999999999999996, 2.34375, 5.555555555555555, 3.124999999999999
6, 49.99999999999999, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 1.3888888888888888, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.3888888888888888, 0.78
12499999999999, 3.061224489795918, 5.555555555555555, 2.7777777777777777, 2.777777
7777777777, 6.249999999999999, 3.7037037037037037, 5.1020408163265305, 8.0, 0.0,
3.9062499999999996, 11.11111111111111, 9.375, 0.0, 49.99999999999999, 16.666666666
666664, 49.99999999999999, 49.99999999999999, 49.99999999999999, 12.49999999999999
8, 49.99999999999999]]
```

In [ ]:
```
postDf = pd.DataFrame(posterior, columns=uniqeFeatures).T
print(postDf)
postDf.plot()
```

```
              0          1          2
4.3   50.000000   0.000000   0.000000
4.4   16.666667   0.000000   0.000000
4.5   50.000000   0.000000   0.000000
4.6   12.500000   0.000000   0.000000
4.7   25.000000   0.000000   0.000000
4.8   10.000000   0.000000   0.000000
4.9    5.555556   1.388889   1.388889
5.0    4.000000   1.000000   0.000000
5.1    4.938272   0.617284   0.000000
5.2    9.375000   3.125000   0.000000
5.3   50.000000   0.000000   0.000000
5.4    6.944444   1.388889   0.000000
5.5    2.040816   5.102041   0.000000
5.6    0.000000   6.944444   1.388889
5.7    1.562500   3.906250   0.781250
5.8    1.020408   3.061224   3.061224
5.9    0.000000  11.111111   5.555556
6.0    0.000000   5.555556   2.777778
6.1    0.000000   5.555556   2.777778
6.2    0.000000   6.250000   6.250000
6.3    0.000000   1.851852   3.703704
6.4    0.000000   2.040816   5.102041
6.5    0.000000   2.000000   8.000000
6.6    0.000000  25.000000   0.000000
6.7    0.000000   2.343750   3.906250
6.8    0.000000   5.555556  11.111111
6.9    0.000000   3.125000   9.375000
7.0    0.000000  50.000000   0.000000
7.1    0.000000   0.000000  50.000000
7.2    0.000000   0.000000  16.666667
7.3    0.000000   0.000000  50.000000
7.4    0.000000   0.000000  50.000000
7.6    0.000000   0.000000  50.000000
7.7    0.000000   0.000000  12.500000
7.9    0.000000   0.000000  50.000000
```

Out[ ]: <AxesSubplot:>



Question 4:

1. Use Gaussian NB from sklearn on UCI Dataset. Plot the decision boundary. Draw analysis on the results by calculating the Roc curve, and classification report.

2. Now consider all features as correlated and construct a classifier using Mahalanobis distance. Draw analysis on the results by calculating the Roc curve, and classification report. Compare the results with part 1. You can refer to this article or book for Mahalanobis distance.

Q1. Use Gaussian NB from sklearn on UCI Dataset. Plot the decision boundary. Draw analysis on the results by calculating the Roc curve, and classification report.

- Utility functions for performance analysis after classification

In [ ]:
```
import numpy as np
import pandas as pd
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(rc={'axes.facecolor': 'lightyellow', 'figure.facecolor': 'lightblue'})


def confusionMatrixAndAccuracyReport(Y_test, Y_pred):
    cm = metrics.confusion_matrix(Y_test, Y_pred)
    overallAccuracy = np.trace(cm)/sum(cm.flatten())

    classwiseAccuracy = np.zeros(len(cm))
    for n in range(len(cm)):
        for i in range(len(cm)):
            for j in range(len(cm)):
                if (i != n and j != n) or (i == n and j == n):
                    classwiseAccuracy[n] += cm[i][j]

    classwiseAccuracy /= sum(cm.flatten())

    plt.figure(figsize=(6, 6))
    plt.title('Accuracy Score: {0:3.3f}'.format(overallAccuracy), size=12)
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')
    sns.heatmap(data=cm, annot=True, square=True,  cmap='Blues')

    plt.show()
    print('Overall Accuracy Score: {0:3.3f}'.format(overallAccuracy))
    print('Classwise Accuracy Score: {0}'.format(classwiseAccuracy))
```

In [ ]:
```
def rocCurveAndAucAnalysis(y_test, y_score):
    n_classes = y_test.shape[1]
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(n_classes):
        fpr[i], tpr[i], _ = metrics.roc_curve(y_test[:, i], y_score[:, i])
        roc_auc[i] = metrics.auc(fpr[i], tpr[i])

    plt.title("ROC Curve")
    plt.plot([0, 1], [0, 1], linestyle='--', lw=2,
             color='r', label='Random', alpha=.8)

    colors = ["aqua", "darkorange", "cornflowerblue"]
    for i, color in zip(range(n_classes), colors):
        plt.plot(
            fpr[i],
```

```
            tpr[i],
            color=color,
            lw=2,
            label="ROC curve of class {0} (area = {1:0.2f})".format(
                i, roc_auc[i]),
        )

    plt.xlabel("False Positve Rate")
    plt.ylabel("True Positive Rate")
    plt.legend()
    plt.show()

    for i in range(n_classes):
        print("Area Under the curve for class {} : {}".format(i, roc_auc[i]))
```

- Data Load

In [ ]:
```
# Data Load

data = pd.read_csv('data-ques-4/breast-cancer.data', names=[
    "Class",
    "age",
    "menopause",
    "tumor-size",
    "inv-nodes",
    "node-caps",
    "deg-malig",
    "breast",
    "breast-quad",
    "irradiat"
], header=None)
data.head()
```

Out[ ]:

	Class	age	menopause	tumor-size	inv-nodes	node-caps	deg-malig	breast	breast-quad	irradiat
0	no-recurrence-events	30-39	premeno	30-34	0-2	no	3	left	left_low	no
1	no-recurrence-events	40-49	premeno	20-24	0-2	no	2	right	right_up	no
2	no-recurrence-events	40-49	premeno	20-24	0-2	no	2	left	left_low	no
3	no-recurrence-events	60-69	ge40	15-19	0-2	no	2	right	left_up	no
4	no-recurrence-events	40-49	premeno	0-4	0-2	no	2	right	right_low	no

- Unique values of different columns

In [ ]:
```
[np.unique(data[col]) for col in data.columns]
```

Out[ ]:
```
[array(['no-recurrence-events', 'recurrence-events'], dtype=object),
 array(['20-29', '30-39', '40-49', '50-59', '60-69', '70-79'], dtype=object),
 array(['ge40', 'lt40', 'premeno'], dtype=object),
 array(['0-4', '10-14', '15-19', '20-24', '25-29', '30-34', '35-39',
        '40-44', '45-49', '5-9', '50-54'], dtype=object),
 array(['0-2', '12-14', '15-17', '24-26', '3-5', '6-8', '9-11'],
       dtype=object),
 array(['?', 'no', 'yes'], dtype=object),
 array([1, 2, 3], dtype=int64),
 array(['left', 'right'], dtype=object),
 array(['?', 'central', 'left_low', 'left_up', 'right_low', 'right_up'],
       dtype=object),
 array(['no', 'yes'], dtype=object)]
```

- Data pre process

In [ ]:
```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

data = data.apply(lambda col: le.fit_transform(col))
data
```

Out[ ]:

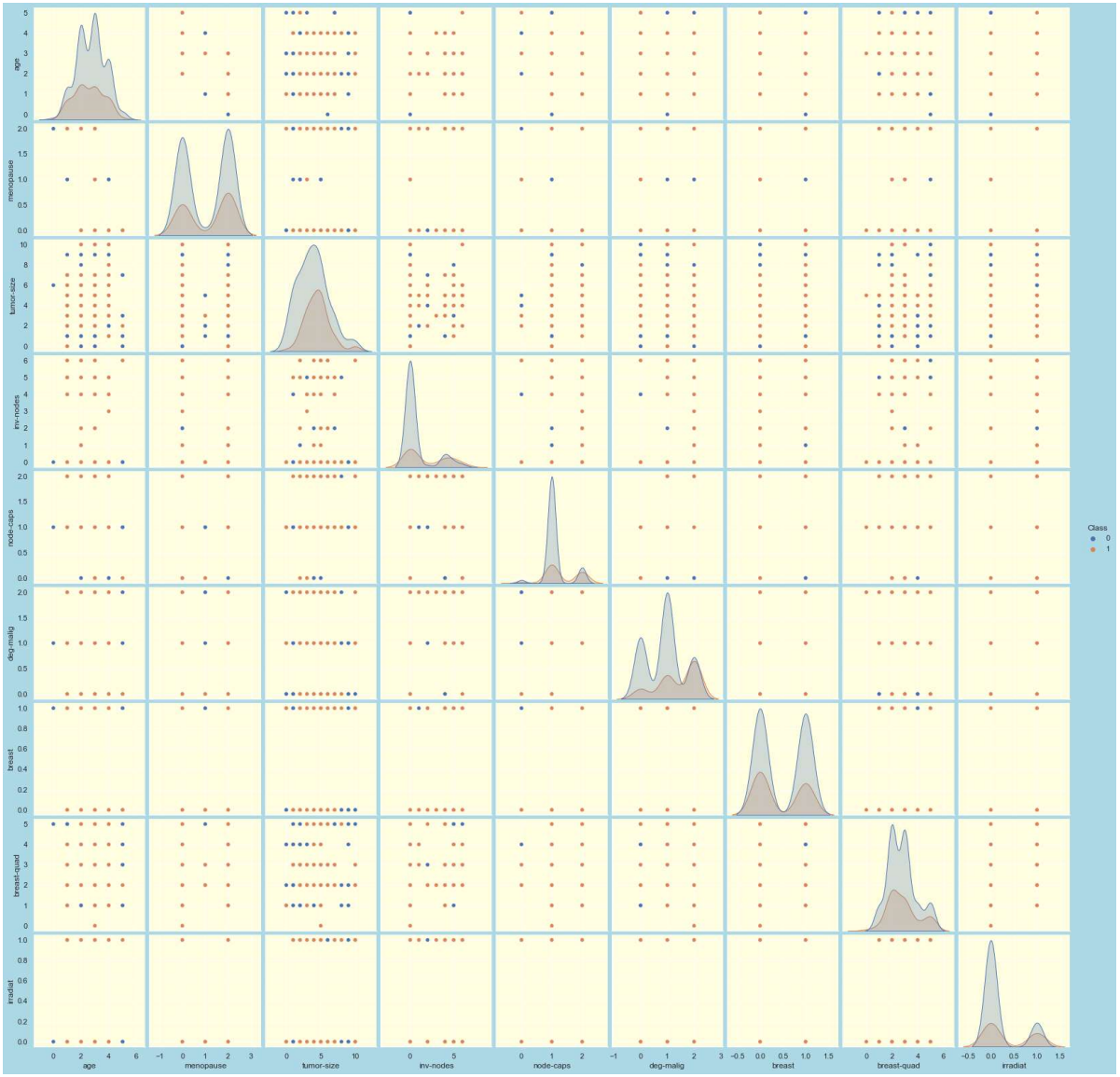
	Class	age	menopause	tumor-size	inv-nodes	node-caps	deg-malig	breast	breast-quad	irradiat
0	0	1	2	5	0	1	2	0	2	0
1	0	2	2	3	0	1	1	1	5	0
2	0	2	2	3	0	1	1	0	2	0
3	0	4	0	2	0	1	1	1	3	0
4	0	2	2	0	0	1	1	1	4	0
...	...	...	...	...	...	...	...	...	...	...
281	1	1	2	5	0	1	1	0	3	0
282	1	1	2	3	0	1	2	0	3	1
283	1	4	0	3	0	1	0	1	3	0
284	1	2	0	5	4	1	2	0	2	0
285	1	3	0	5	4	1	2	0	2	0

286 rows × 10 columns

- Data Exploration

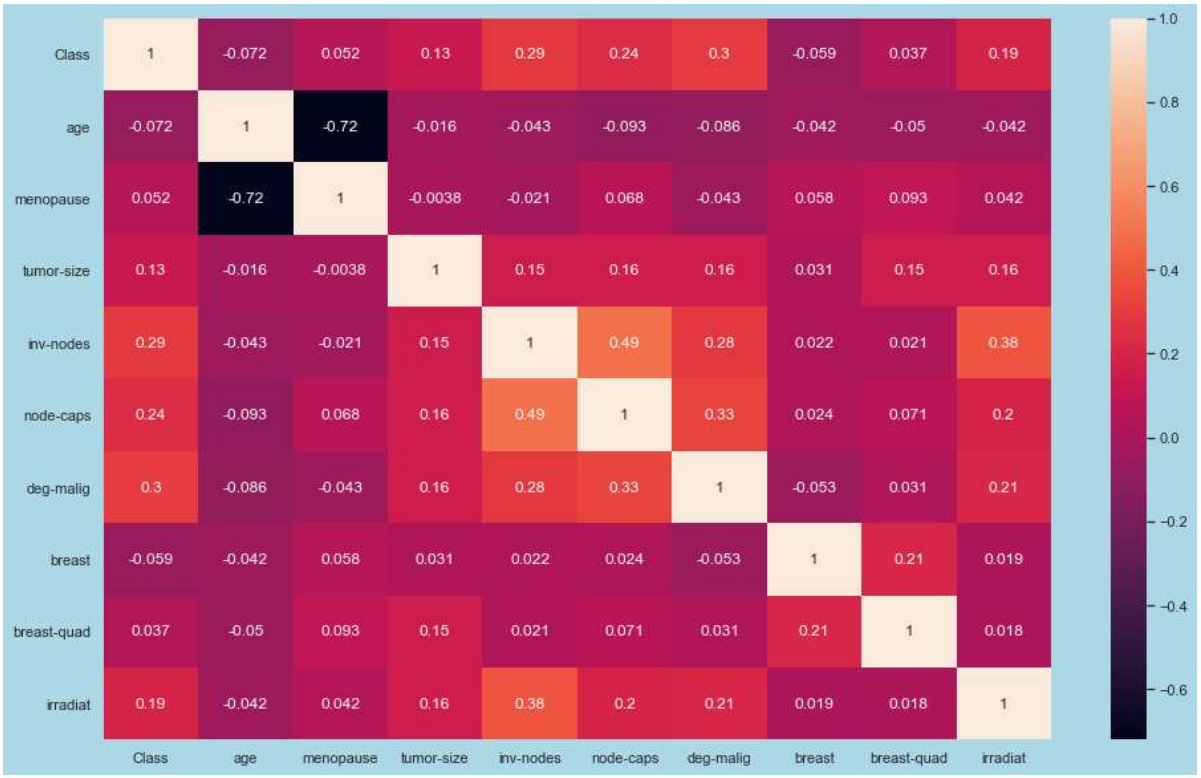
In [ ]:
```
import seaborn as sns
sns.pairplot(data=data,hue='Class')
```

Out[ ]:
```
<seaborn.axisgrid.PairGrid at 0x2a7e34fab50>
```



In [ ]:
```
plt.figure(figsize=(16,10))
sns.heatmap(data.corr(),annot=True)
```

Out[ ]:
```
<AxesSubplot:>
```



In [ ]:
```
# Feature Selection

""" from sklearn.feature_selection import SelectKBest, chi2

chi2Features = SelectKBest(chi2,k=6)

X = data.drop("Class", axis=1).astype(int)
Y = data["Class"]
X =  chi2Features.fit_transform(X,Y)

print(X) """
```

Out[ ]:
```
' from sklearn.feature_selection import SelectKBest, chi2\n\nchi2Features = Select
KBest(chi2,k=6)\n\nX = data.drop("Class", axis=1).astype(int)\nY = data["Class"]\n
X =  chi2Features.fit_transform(X,Y)\n\nprint(X) '
```

Split the dataset into train and test sets (80:20)

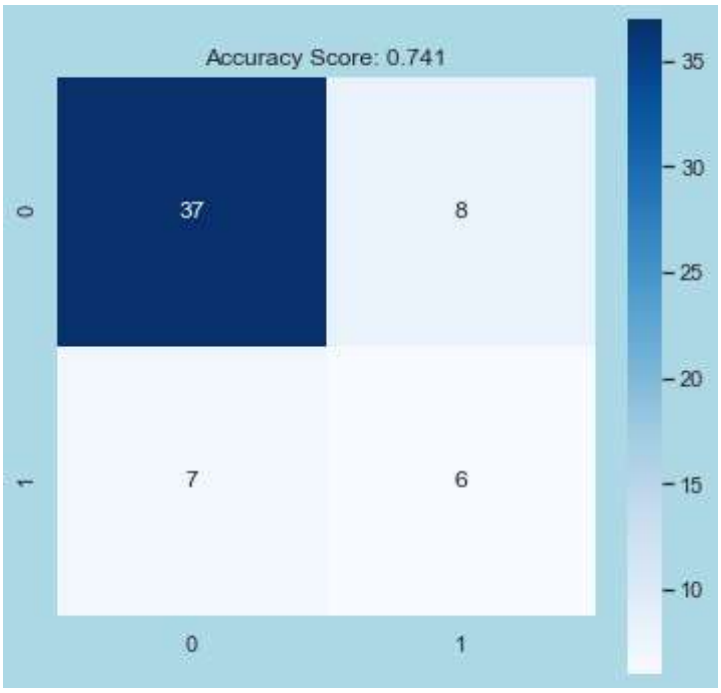
In [ ]:
```
from sklearn.preprocessing import label_binarize
from sklearn.model_selection import train_test_split

X = data.drop("Class", axis=1)
Y = data["Class"]
X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size=0.2, random_state=23)


print('Given data set==> {}'.format(data.shape))
print('Train data set==> {}'.format(X_train.shape))
print('Test data set==> {}'.format(X_test.shape))
# for use in roc analysis
y_test_bin = label_binarize(Y_test, classes=np.unique(Y_test))
```

```
Given data set==> (286, 10)
Train data set==> (228, 9)
Test data set==> (58, 9)
```

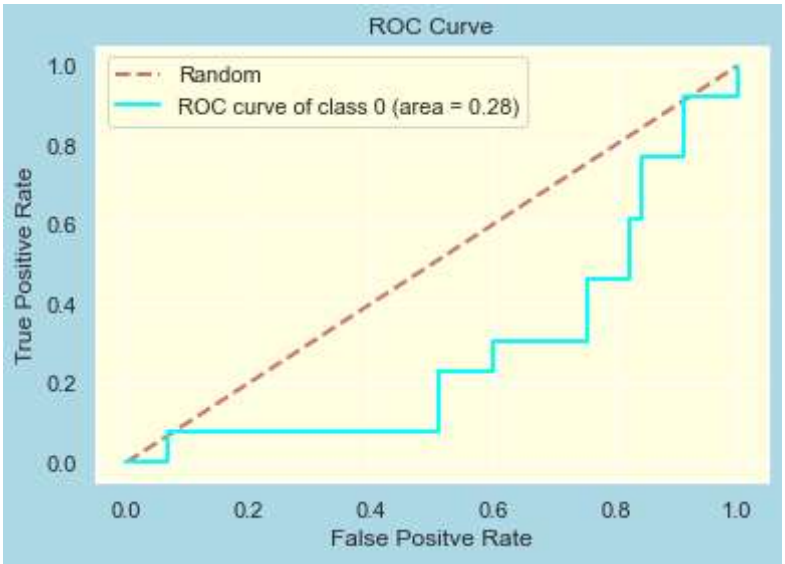
- Fitting data in sklearn GaussianNB and analyse performance

In [ ]:
```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(X_train, Y_train)
Y_pred_lib = model.predict(X_test)
y_score_lib = model.predict_proba(X_test)


print(metrics.classification_report(Y_test, Y_pred_lib))

confusionMatrixAndAccuracyReport(Y_test, Y_pred_lib)

rocCurveAndAucAnalysis(y_test_bin, y_score_lib)
```

```
              precision    recall  f1-score   support

           0       0.84      0.82      0.83        45
           1       0.43      0.46      0.44        13

    accuracy                           0.74        58
   macro avg       0.63      0.64      0.64        58
weighted avg       0.75      0.74      0.74        58
```




```
Overall Accuracy Score: 0.741
Classwise Accuracy Score: [0.74137931 0.74137931]
```



```
Area Under the curve for class 0 : 0.28034188034188035
```

Q2. Now consider all features as correlated and construct a classifier using Mahalanobis distance. Draw analysis on the results by calculating the Roc curve, and classification report. Compare the results with part 1. You can refer to this article or book for Mahalanobis distance.

- Mahalanobis distance function

In [ ]:
```
import scipy as sp
def mahalanobis(x=None, data=None, cov=None):
    """Compute the Mahalanobis Distance between each row of x and the data
    x    : vector or matrix of data with, say, p columns.
    data : ndarray of the distribution from which Mahalanobis distance of each obse
    cov  : covariance matrix (p x p) of the distribution. If None, will be computed
    """
    x_minus_mu = x - np.mean(data)
    if not cov:
        cov = np.cov(data.values.T)
    inv_covmat = sp.linalg.inv(cov)
    left_term = np.dot(x_minus_mu, inv_covmat)
    mahal = np.dot(left_term, x_minus_mu.T)
    return mahal.diagonal()
```

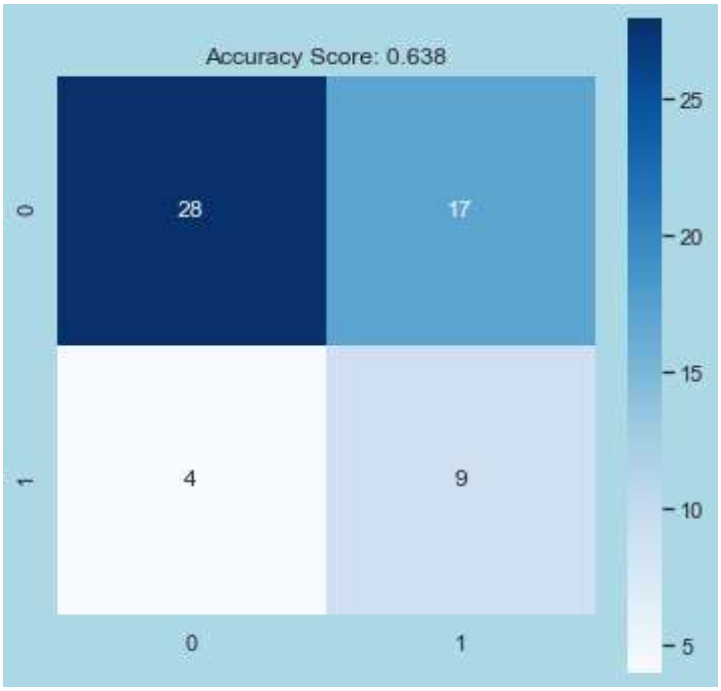
- Mahalanobis binary classifier

In [ ]:
```
class MahalanobisBinaryClassifier():
    def __init__(self, xtrain, ytrain):
        self.xtrain_pos = xtrain.loc[ytrain == 1, :]
        self.xtrain_neg = xtrain.loc[ytrain == 0, :]

    def predict_proba(self, xtest):
        pos_neg_dists = [(p,n) for p, n in zip(mahalanobis(xtest, self.xtrain_pos),
        return np.array([(1-n/(p+n), 1-p/(p+n)) for p,n in pos_neg_dists])

    def predict(self, xtest):
        return np.array([np.argmax(row) for row in self.predict_proba(xtest)])
```

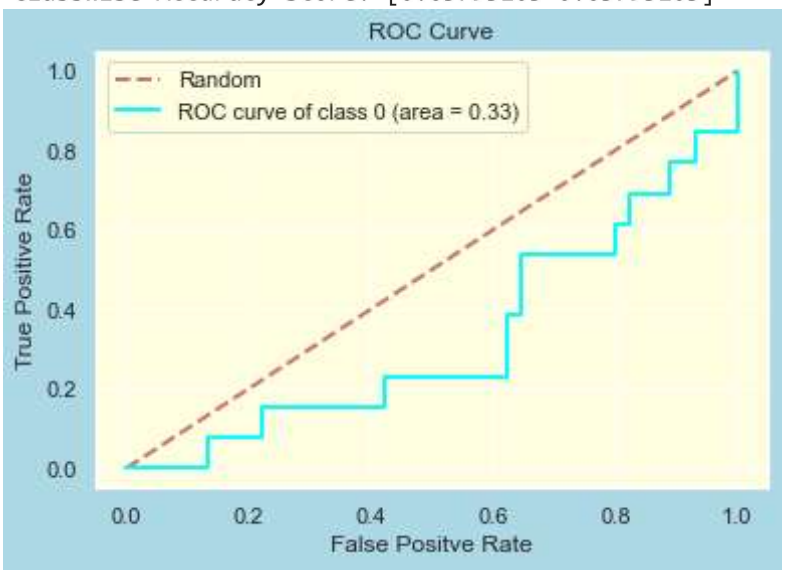
- Applying Mahalanobis Classifier on the given dataset and performance analysis

In [ ]:
```
clf = MahalanobisBinaryClassifier(X_train, Y_train)
y_score_lib = clf.predict_proba(X_test)
Y_pred_lib = clf.predict(X_test)

print(metrics.classification_report(Y_test, Y_pred_lib))

confusionMatrixAndAccuracyReport(Y_test, Y_pred_lib)

rocCurveAndAucAnalysis(y_test_bin, y_score_lib)
```

	precision	recall	f1-score	support
0	0.88	0.62	0.73	45
1	0.35	0.69	0.46	13
accuracy			0.64	58
macro avg	0.61	0.66	0.59	58
weighted avg	0.76	0.64	0.67	58



Overall Accuracy Score: 0.638
Classwise Accuracy Score: [0.63793103 0.63793103]



Area Under the curve for class 0 : 0.3264957264957265