Question 1

In [70]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt
from sklearn.preprocessing import StandardScaler
```

In [71]:

```
dataset = pd.read_csv('Iris.csv', header=0)
dataset.drop('Id', axis=1, inplace=True)
dataset.head()
```

Out[71]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

In [72]:

```
x = dataset.iloc[:, :-1]
y = dataset.iloc[:, -1]
x.shape, y.shape
```

Out[72]:

```
((150, 4), (150,))
```

In [73]:

```
# group the training data into its respective classes

d = {}
for s, rows in dataset.groupby(['Species']):
    d[s] = np.array(rows)[:, :-1]
print(d.keys())
```

```
dict_keys(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'])
```

In [74]:

```
#claculate the mean vector of given training data of K-dimensions excluding the target clas


featureMean = x.mean().values
classMean = dataset.groupby(['Species']).mean().values
print(featureMean)
print("\n")
print(classMean)
```

```
[5.84333333 3.054      3.75866667 1.19866667]


[[5.006 3.418 1.464 0.244]
 [5.936 2.77  4.26  1.326]
 [6.588 2.974 5.552 2.026]]
```

In [75]:

```
#and calculate class-wise mean vector for the given training data
# print(x.mean())
print("Class-wise means:")
print(dataset.groupby(['Species']).mean())
```

```
Class-wise means:
                 SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
Species
Iris-setosa              5.006         3.418          1.464         0.244
Iris-versicolor          5.936         2.770          4.260         1.326
Iris-virginica           6.588         2.974          5.552         2.026
```

In [76]:

```
#claculate the scatter matrices needed to maximize the difference between means of given cl
#and minimize the variance of given classes

#scatter within classes
SWC = np.zeros((x.shape[1], x.shape[1]))
i = 0
for key in d.keys():
    classwiseMean = classMean[i].reshape(-1, 1)
    s = np.zeros(SWC.shape)
    for a in d[key]:
        a = a.reshape(-1, 1)
        s = s + np.dot((a-classwiseMean), ((a-classwiseMean).T))
    SWC = SWC + s
    i += 1
print("Scatter within Classes: ")
print(SWC)
```

```
Scatter within Classes:
[[38.95620000000002 13.683 24.614000000000004 5.6556000000000015]
 [13.683 17.035000000000001 8.12 4.913200000000002]
 [24.614000000000004 8.12 27.220000000000006 6.2536000000000005]
 [5.6556000000000015 4.913200000000002 6.2536000000000005
  6.175599999999999]]
```

In [77]:

```
n = dataset['Species'].value_counts().values
```

In [78]:

```
#scatter between classes for each classes
SBC = np.zeros(SWC.shape)
for i in range(3):
    classwise_mean = classMean[i].reshape(-1,1)
    featureswise_mean = featureMean.reshape(-1,1)
    SBC += n[i]*np.dot((classwise_mean-featureswise_mean), (classwise_mean-featureswise_mea

print(SBC)
```

```
[[ 63.21213333 -19.534      165.16466667  71.36306667]
 [-19.534       10.9776     -56.0552     -22.4924    ]
 [165.16466667 -56.0552     436.64373333 186.90813333]
 [ 71.36306667 -22.4924     186.90813333  80.60413333]]
```

In [79]:

```
#claculate the eigen values of M and get the eigen vector pairs for the first n needed dime
SWC = SWC.astype('float64')
eigen_values, eigen_vectors = np.linalg.eig(np.linalg.inv(SWC).dot(SBC))

print(eigen_values)
print(eigen_vectors)
```

```
[3.22719578e+01 2.77566864e-01 2.86057914e-15 7.28263021e-15]
[[-0.20490976 -0.00898234 -0.83287318 -0.4915145 ]
 [-0.38714331 -0.58899857  0.38982272 -0.11747774]
 [ 0.54648218  0.25428655  0.38892165 -0.18951674]
 [ 0.71378517 -0.76703217 -0.05568185  0.84184077]]
```

In [80]:

```
#sorting eigen vectors by decreasing eigen values
print(eigen_values)
print(eigen_values[eigen_values.argsort()])
print(eigen_values[(-eigen_values).argsort()[:]])
```

```
[3.22719578e+01 2.77566864e-01 2.86057914e-15 7.28263021e-15]
[2.86057914e-15 7.28263021e-15 2.77566864e-01 3.22719578e+01]
[3.22719578e+01 2.77566864e-01 7.28263021e-15 2.86057914e-15]
```

In [81]:

```
#if we negate the values then we get the descending order sorted indices
sorted_indices=np.argsort(-eigen_values)
print("BEFORE SORTING:")
print(eigen_values)
sorted_eigenvalues=eigen_values[sorted_indices]
print("AFTER SORTING:")
print(sorted_eigenvalues)
#applied to all columns
sorted_eigenvectors=eigen_vectors[:,sorted_indices]
print("SORTED EIGEN VECTORS")
print(sorted_eigenvectors)
```

```
BEFORE SORTING:
[3.22719578e+01 2.77566864e-01 2.86057914e-15 7.28263021e-15]
AFTER SORTING:
[3.22719578e+01 2.77566864e-01 7.28263021e-15 2.86057914e-15]
SORTED EIGEN VECTORS
[[-0.20490976 -0.00898234 -0.4915145  -0.83287318]
 [-0.38714331 -0.58899857 -0.11747774  0.38982272]
 [ 0.54648218  0.25428655 -0.18951674  0.38892165]
 [ 0.71378517 -0.76703217  0.84184077 -0.05568185]]
```

In [82]:

```
# choosing k eigen values with the largest eigen values
#lets say k=2
k=2
weights_matrix = sorted_eigenvectors[:, 0:k]

#first column in our rearranges eigen vector matrix will be a linear discriminant component
#that captures the higest variability
print(weights_matrix)
```

```
[[-0.20490976 -0.00898234]
 [-0.38714331 -0.58899857]
 [ 0.54648218  0.25428655]
 [ 0.71378517 -0.76703217]]
```

In [83]:

```
# transforming the samples onto the new subspace
x_lda = np.array(x.dot(weights_matrix))
x_lda.shape
```

Out[83]:

```
(150, 2)
```

In [ ]:

```
```

In [ ]:

```
```

Question 2

In [84]:

```
data = pd.read_csv('Wine.csv')
print(data.head())
```

```
   Alcohol  Malic_Acid   Ash  Ash_Alcanity  Magnesium  Total_Phenols  \
0    14.23        1.71  2.43          15.6        127           2.80
1    13.20        1.78  2.14          11.2        100           2.65
2    13.16        2.36  2.67          18.6        101           2.80
3    14.37        1.95  2.50          16.8        113           3.85
4    13.24        2.59  2.87          21.0        118           2.80

   Flavanoids  Nonflavanoid_Phenols  Proanthocyanins  Color_Intensity   Hue
\
0        3.06                  0.28             2.29             5.64  1.04
1        2.76                  0.26             1.28             4.38  1.05
2        3.24                  0.30             2.81             5.68  1.03
3        3.49                  0.24             2.18             7.80  0.86
4        2.69                  0.39             1.82             4.32  1.04

   OD280  Proline  Customer_Segment
0   3.92     1065                 1
1   3.40     1050                 1
2   3.17     1185                 1
3   3.45     1480                 1
4   2.93      735                 1
```

In [85]:

```
X = data.iloc[:, :-1]
Y = data.iloc[:, -1]
X.shape, Y.shape
```

Out[85]:

```
((178, 13), (178,))
```

In [86]:

```
# feature scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

In [87]:

```
X = scaler.fit_transform(X)
```

In [88]:

```
#split the dataset
from sklearn.model_selection import train_test_split as split_data
```

In [89]:

```
train_x, test_x, train_y, test_y = split_data(X, Y, test_size=0.3, shuffle=True)
train_x.shape, test_x.shape, train_y.shape, test_y.shape
```

Out[89]:

```
((124, 13), (54, 13), (124,), (54,))
```

In [90]:

```
#apply lda
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

lda = LDA(n_components = 1)
train_x = lda.fit_transform(train_x, train_y)
test_x = lda.transform(test_x)
```

In [91]:

```
#train the model with logistic regression
from sklearn.linear_model import LogisticRegression as LR
clf = LR()
clf.fit(train_x, train_y)
pred_y = clf.predict(test_x)
```

In [92]:

```
#compute the confusion matrix
from sklearn.metrics import accuracy_score, confusion_matrix

print("Confusion matrix")
print(confusion_matrix(test_y, pred_y))
print(f"Accuracy: {str(round(accuracy_score(test_y, pred_y), 2))}")
```

```
Confusion matrix
[[19  0  0]
 [ 1 21  1]
 [ 0  0 12]]
Accuracy: 0.96
```

In [ ]:

```

```