In [1]:

```
import os import pandas as pd from sklearn.linear_model
import LogisticRegression import matplotlib.pyplot as plt
import numpy as np from IPython.display import display
from matplotlib.colors import ListedColormap from
sklearn.model_selection import train_test_split from
sklearn.preprocessing import StandardScaler path=
r"C:\Users\prasad\Downloads\seeds_dataset.txt" features =
['Area',             'Perimeter',
            'Compactness',
            'Length ',
            'Width',
            'Asymmetry coefficient',
            ' groove.']
df = pd.read_csv(path, delimiter=r'\t+', header=None, names=features +
['target' display(df) from sklearn.model_selection import train_test_split
X,y = df.iloc[1:,[0,1,2,3,4,5,6]].values, df.iloc[1:,
7].values X=X.astype('float64') y=y.astype('int64')
X_train,    X_test,    y_train,    y_test    =    train_test_split(X,    y,
test_size=0.3,stratify
# standardize the features
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
display(X)
X_test_std =
sc.transform(X_test)
np.set_printoptions(precision=4)
mean_vecs = [] for label in
range(1,4):
 mean_vecs.append(np.mean(X_train_std[y_train==label],
axis=0))  print('MV %s: %s\n' %(label, mean_vecs[label-1])) d
= 7 # number of features S_W = np.zeros((d, d)) for label, mv
in zip(range(1, 4), mean_vecs):
 class_scatter = np.zeros((d, d)) for
row in X_train_std[y_train == label]:
 row, mv = row.reshape(d, 1), mv.reshape(d, 1)
class_scatter += (row - mv).dot((row - mv).T)
S_W += class_scatter
print('Within-class scatter matrix: %sx%s' % (S_W.shape[0],
S_W.shape[1])) print('Class label distribution: %s'%
np.bincount(y_train)[1:]) d = 7 # number of features S_W = np.zeros((d,
d)) for label,mv in zip(range(1, 4), mean_vecs):
 class_scatter = np.cov(X_train_std[y_train==label].T)
S_W += class_scatter
print('Scaled within-class scatter matrix: %sx%s'% (S_W.shape[0],
S_W.shape[1])) mean_overall = np.mean(X_train_std, axis=0) d = 7 # number of
features S_B = np.zeros((d, d)) for i, mean_vec in enumerate(mean_vecs):
 n = X_train_std[y_train == i + 1, :].shape[0]
mean_vec = mean_vec.reshape(d, 1) # make column
vector  mean_overall = mean_overall.reshape(d, 1)
```

```
 S_B += n * (mean_vec - mean_overall).dot((mean_vec - mean_overall).T)
print('Between-class scatter matrix: %sx%s' % (S_B.shape[0], S_B.shape[1]))
eigen_vals, eigen_vecs =np.linalg.eig(np.linalg.inv(S_W).dot(S_B)) eigen_pairs
= [(np.abs(eigen_vals[i]), eigen_vecs[:,i])for i in range(len(eigen_
eigen_pairs = sorted(eigen_pairs,key=lambda k: k[0], reverse=True)
print('Eigenvalues in descending order:\n') for eigen_val in eigen_pairs:
print(eigen_val[0])  tot = sum(eigen_vals.real) discr = [(i / tot) for i in
sorted(eigen_vals.real, reverse=True)] cum_discr = np.cumsum(discr)
plt.bar(range(1, 8), discr, alpha=0.5, align='center',label='Individual
"discrim plt.step(range(1, 8), cum_discr, where='mid',label='Cumulative
"discriminability plt.ylabel('"Discriminability" ratio') plt.xlabel('Linear
Discriminants') plt.ylim([-0.1, 1.1]) plt.legend(loc='best')
plt.tight_layout() plt.show() w = np.hstack((eigen_pairs[0][1][:,
np.newaxis].real,eigen_pairs[1][1][:, np.new print('Matrix W:\n', w)
X_train_lda = X_train_std.dot(w) colors = ['r', 'b',
'g'] markers = ['s', 'x', 'o'] for l, c, m in
zip(np.unique(y_train), colors, markers):
 plt.scatter(X_train_lda[y_train==l, 0],  X_train_lda[y_train==l, 1] * (-
1),  c=c, label=l, marker=m) plt.xlabel('LDA1') plt.ylabel('LDA2')
plt.legend(loc='lower right') plt.tight_layout() plt.show() from
sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
lda = LDA(n_components=2)
X_train_lda = lda.fit_transform(X_train_std, y_train) from
matplotlib.colors import ListedColormap def
plot_decision_regions(X, y, classifier, resolution=0.02):
    # setup marker generator and color map     markers =
('s', 'x', 'o', '^', 'v')     colors = ('red', 'blue',
'lightgreen', 'gray', 'cyan')     cmap =
ListedColormap(colors[:len(np.unique(y))])

    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1     xx1,
xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
np.arange(x2_min, x2_max, resolution))     z =
classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)     z
= z.reshape(xx1.shape)     plt.contourf(xx1, xx2, z, alpha=0.4,
cmap=cmap)     plt.xlim(xx1.min(), xx1.max())
plt.ylim(xx2.min(), xx2.max())

    # plot class samples     for idx, c1
in enumerate(np.unique(y)):
        plt.scatter(x=X[y == c1, 0], y=X[y == c1, 1],
alpha=0.6, edgecolor='black',                    marker=markers[idx],
label=c1) lr = LogisticRegression(multi_class='ovr',
random_state=1,solver='lbfgs')
```

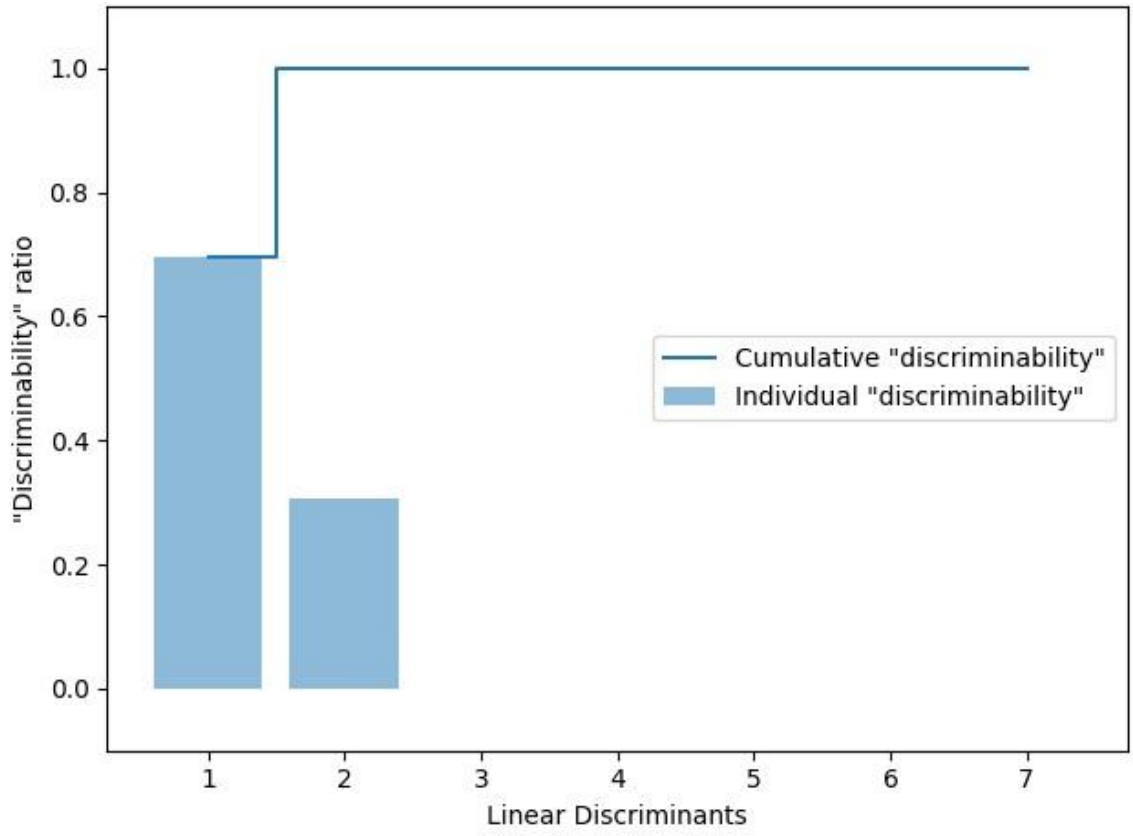
```
lr = lr.fit(X_train_lda, y_train)
plot_decision_regions(X_train_lda, y_train,
classifier=lr) plt.xlabel('LD 1') plt.ylabel('LD 2')
plt.legend(loc='lower left') plt.tight_layout()
plt.show()
X_test_lda = lda.transform(X_test_std)
plot_decision_regions(X_test_lda, y_test,
classifier=lr) plt.xlabel('LD 1') plt.ylabel('LD 2')
plt.legend(loc='lower left') plt.tight_layout()
plt.show()
```

	Area					Asymmetry coefficient	Compactness	Perimeter Length Width groove. target
0	15.26	14.84	0.8710	5.763	3.312	2.221	5.220	1
1	14.88	14.57	0.8811	5.554	3.333	1.018	4.956	1
2	14.29	14.09	0.9050	5.291	3.337	2.699	4.825	1
3	13.84	13.94	0.8955	5.324	3.379	2.259	4.805	1
4	16.14	14.99	0.9034	5.658	3.562	1.355	5.175	1
...	...	...	...	...	...	...	...	...
205	12.19	13.20	0.8783	5.137	2.981	3.631	4.870	3
206	11.23	12.88	0.8511	5.140	2.795	4.325	5.003	3
207	13.20	13.66	0.8883	5.236	3.232	8.315	5.056	3
208	11.84	13.21	0.8521	5.175	2.836	3.598	5.044	3
209	12.30	13.34	0.8684	5.243	2.974	5.637	5.063	3

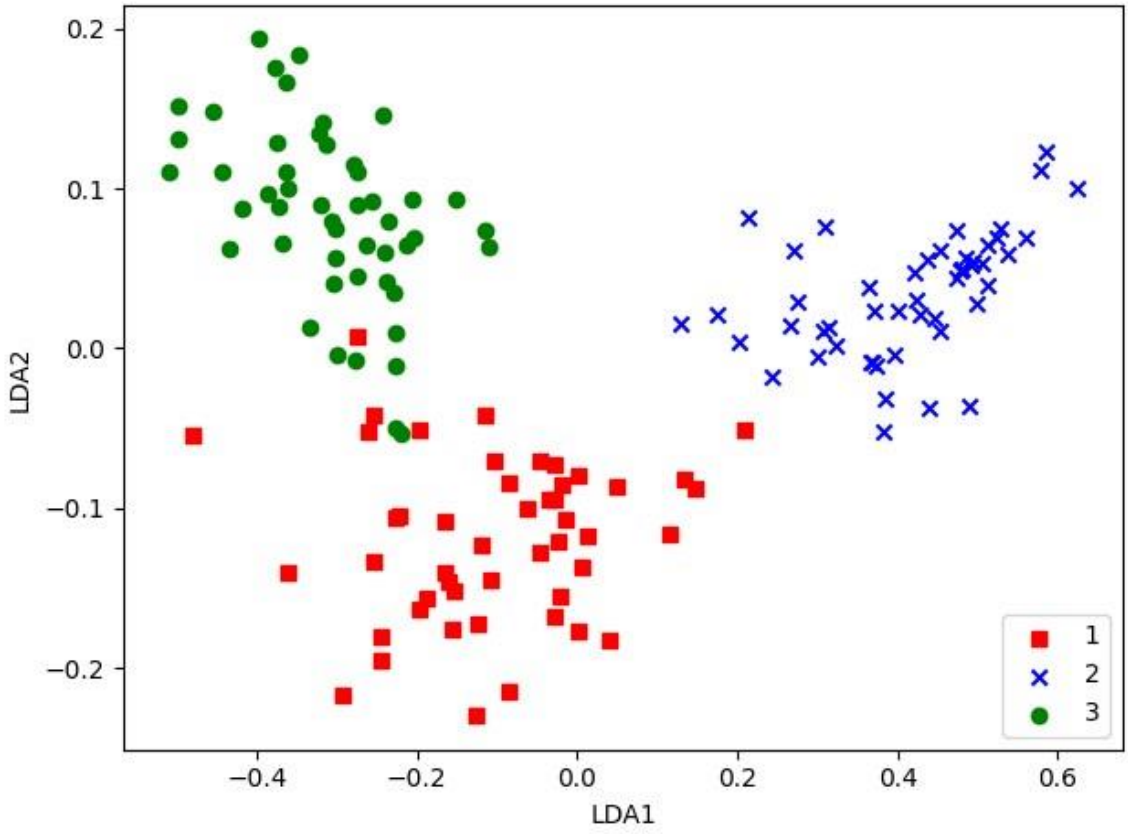
210 rows × 8 columns

```
array([[14.88  , 14.57  ,  0.8811, ...,  3.333 ,  1.018 ,  4.956 ],
[14.29  , 14.09  ,  0.905 , ...,  3.337 ,  2.699 ,  4.825 ],
       [13.84  , 13.94  ,  0.8955, ...,  3.379 ,  2.259 ,  4.805 ],
...,
       [13.2   , 13.66  ,  0.8883, ...,  3.232 ,  8.315 ,  5.056 ],
       [11.84  , 13.21  ,  0.8521, ...,  2.836 ,  3.598 ,  5.044 ],
       [12.3   , 13.34  ,  0.8684, ...,  2.974 ,  5.637 ,  5.063 ]])
MV 1: [-0.155  -0.1764  0.3855 -0.2423 -0.0122 -0.7162 -0.6113]

MV 2: [1.1928 1.1997 0.5433 1.1663 1.1026 0.1007 1.2519] MV

3: [-1.041  -1.0268 -0.9209 -0.9289 -1.0907  0.6008 -0.6531]

Within-class scatter matrix: 7x7
```

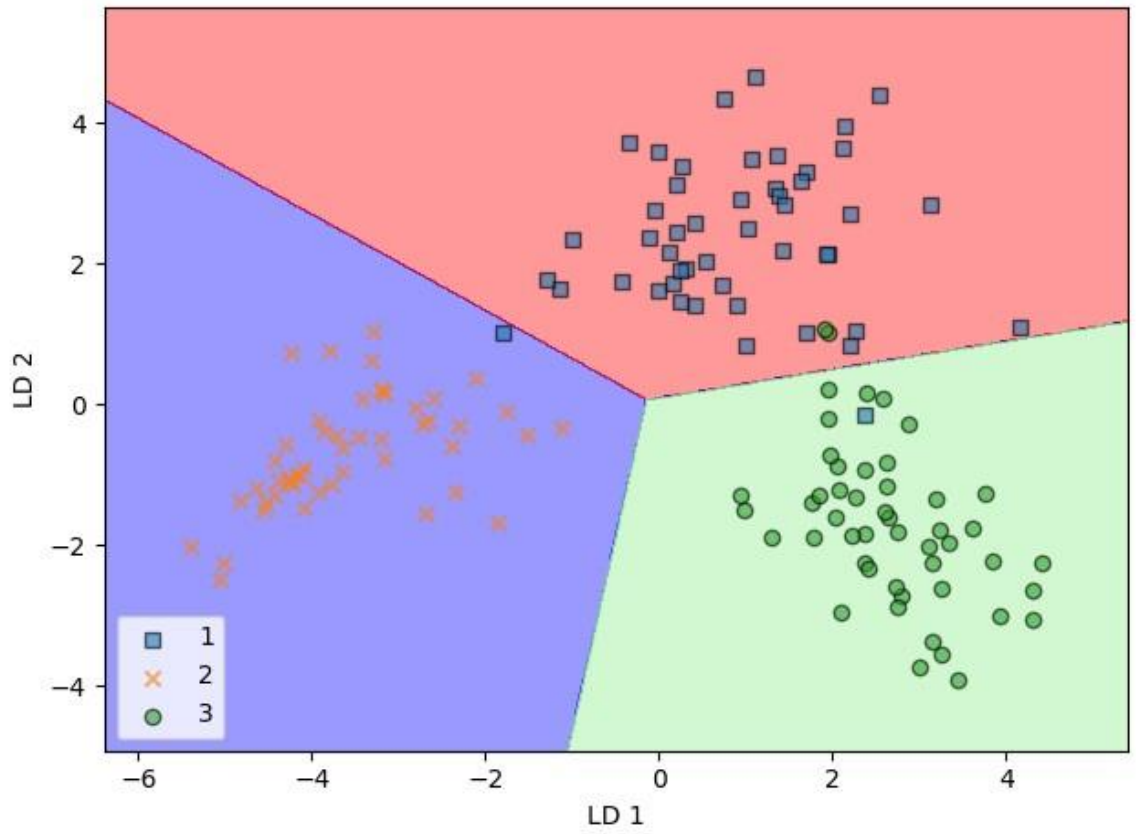
```
Class label distribution: [48 49 49]
Scaled within-class scatter matrix: 7x7
Between-class scatter matrix: 7x7
Eigenvalues in descending order:

332.9709998913104
146.56070410516344
3.0086439656230857e-12 1.6634055202958446e-
12 1.6634055202958446e-12
4.1041428393878205e-13
3.028902817669226e-13
```



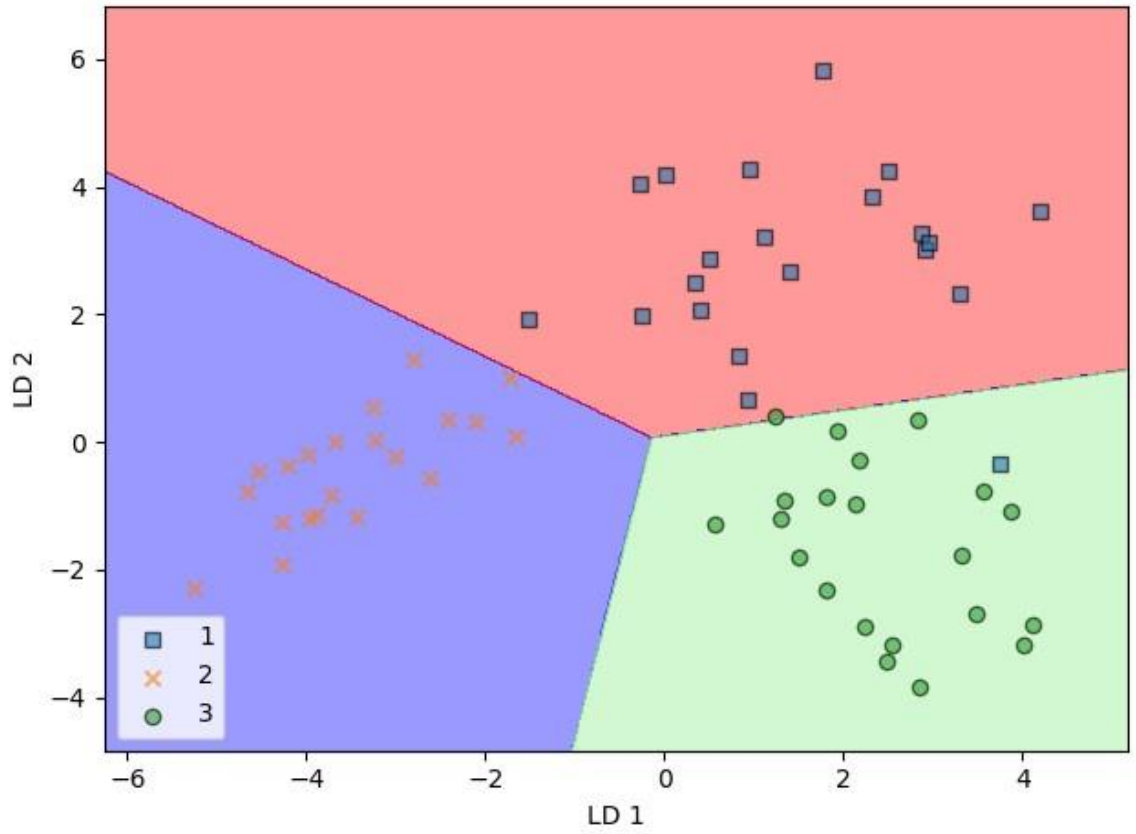

```
Matrix W:
 [[-0.3843 -0.7089]
 [ 0.8384  0.6507]
 [ 0.0965  0.1006]
 [-0.3037  0.1738]
 [-0.0816  0.0045]
 [ 0.0051 -0.0222]
 [ 0.2027 -0.1823]]
```




```
C:\Users\prasad\AppData\Local\Temp\ipykernel_21624\2709391357.py:115: Us
erWarning: You passed a edgecolor/edgecolors ('black') for an unfilled marker
('x').  Matplotlib is ignoring the edgecolor in favor of the facecolor.  This
b ehavior may change in the future.
  plt.scatter(x=X[y == c1, 0], y=X[y == c1, 1],
```




```
C:\Users\prasad\AppData\Local\Temp\ipykernel_21624\2709391357.py:115: Us
erWarning: You passed a edgecolor/edgecolors ('black') for an unfilled marker
('x').  Matplotlib is ignoring the edgecolor in favor of the facecolor.  This
b ehavior may change in the future.   plt.scatter(x=X[y == c1, 0], y=X[y ==
c1, 1],
```



In [ ]:

In [ ]:

In [11]

```
]
```

```
import os import pandas as pd from sklearn.linear_model
import LogisticRegression import matplotlib.pyplot as plt
import numpy as np from matplotlib.colors import
ListedColormap from sklearn.model_selection import
train_test_split from sklearn.preprocessing import
StandardScaler path=
(r"C:\Users\prasad\Downloads\seeds_dataset.txt") features =
['Area',             'Perimeter',
            'Compactness',
            'Length of kernel',
            'Width of kernel',
            'Asymmetry coefficient',
            ' groove.']
df = pd.read_csv(path,delimiter='[\t]+',header=None,names=features +
['target']) display(df) from sklearn.model_selection import train_test_split
X, y = df.iloc[1:,[0,1,2,3,4,5,6]].values, df.iloc[1:, 7].values
X_train,    X_test,    y_train,    y_test    =    train_test_split(X,    y,
test_size=0.3,stratify
# standardize the features
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
display(X)
X_test_std = sc.transform(X_test) import numpy as np cov_mat =
np.cov(X_train_std.T) eigen_vals, eigen_vecs = np.linalg.eig(cov_mat)
print('\nEigenvalues \n%s' % eigen_vals) import numpy as np cov_mat =
np.cov(X_train_std.T) eigen_vals, eigen_vecs = np.linalg.eig(cov_mat)
print('\nEigenvalues \n%s' % eigen_vals) tot = sum(eigen_vals) var_exp = [(i /
tot) for i in sorted(eigen_vals, reverse=True)] cum_var_exp =
np.cumsum(var_exp) import matplotlib.pyplot as plt plt.bar(range(1,8),
var_exp, alpha=0.5, align='center',label='Individual explain
plt.step(range(1,8), cum_var_exp, where='mid',label='Cumulative explained
varian plt.ylabel('Explained variance ratio') plt.xlabel('Principal component
index') plt.legend(loc='center right' ) plt.tight_layout() plt.show()
# Make a list of (eigenvalue, eigenvector) tuples
eigen_pairs = [(np.abs(eigen_vals[i]), eigen_vecs[:, i])
for i in range(len(eigen_vals))]
# Sort the (eigenvalue, eigenvector) tuples from high to low
eigen_pairs.sort(key=lambda k: k[0], reverse=True) display(eigen_pairs) w =
np.hstack((eigen_pairs[0][1][:, np.newaxis],eigen_pairs[1][1][:, np.newaxis
print('Matrix W:\n', w)
X_train_pca = X_train_std.dot(w) colors = ['r', 'b',
'g'] markers = ['s', 'x', 'o'] for l, c, m in
zip(np.unique(y_train), colors, markers):
```

```
    plt.scatter(X_train_pca[y_train==l, 0],X_train_pca[y_train==l, 1],c=c,
label plt.xlabel('PC 1') plt.ylabel('PC 2') plt.legend(loc='lower left')
plt.tight_layout() plt.show()         from matplotlib.colors import
ListedColormap import matplotlib.pyplot as plt import numpy as np from
sklearn.linear_model import LogisticRegression from sklearn.decomposition
import PCA from sklearn.model_selection import train_test_split from
sklearn.preprocessing import StandardScaler
# Load Wheat Seeds dataset
path= np.genfromtxt(r"C:\Users\prasad\Downloads\seeds_dataset.txt") X =
path[:, :-1] y = path[:, -1]
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_ sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)
# Perform PCA
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_std)
X_test_pca = pca.transform(X_test_std)
# Train logistic regression model
lr = LogisticRegression(multi_class='ovr', random_state=1, solver='lbfgs')
lr.fit(X_train_pca, y_train)
_test_pca = pca.transform(X_test_std) def
plot_decision_regions(X, y, classifier, resolution=0.02):
    #setup marker generator and color map     markers =
('s', 'x', 'o', '^', 'v')     colors = ('red', 'blue',
'lightgreen', 'gray', 'cyan')     cmap =
ListedColormap(colors[:len(np.unique(y))])
    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1     xx1,
xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
np.arange(x2_min, x2_max, resolution))
    Z =lr.predict(np.array([xx1.ravel(),
xx2.ravel()]).T)     Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.3,
cmap=cmap)     plt.xlim(xx1.min(), xx1.max())
plt.ylim(xx2.min(), xx2.max())     # plot
examples by class     for idx, c1 in
enumerate(np.unique(y)):
        mask=np.where(y == c1)[0]
if len(mask) > 0:
            plt.scatter(x=X[mask, 0],
y=X[mask, 1],
alpha=0.8,
color=cmap(idx),
edgecolor='black',
marker=markers[idx],
label=c1)
    # plot decision regions for training data
plot_decision_regions(X_train_pca, y_train,
classifier=lr) plt.xlabel('PC 1') plt.ylabel('PC 2')
```

```
plt.legend(loc='lower left')
plt.tight_layout()
plt.show()
# plot decision regions for test data
plot_decision_regions(X_test_pca, y_test,
classifier=lr) plt.xlabel('PC1') plt.ylabel('PC2')
plt.legend(loc='lower left') plt.tight_layout()
plt.show()
```

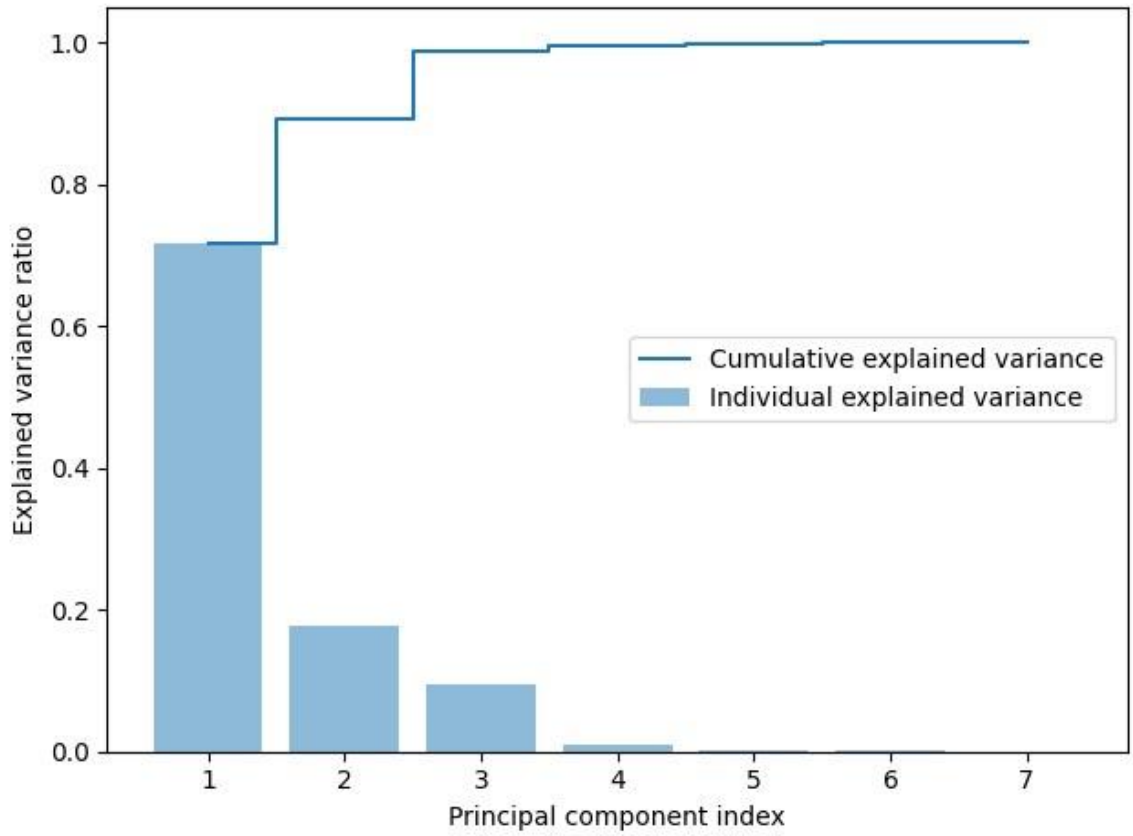
```
C:\Users\prasad\AppData\Local\Temp\ipykernel_21004\701045780.py:17: Pars
erWarning: Falling back to the 'python' engine because the 'c' engine does
not support regex separators (separators > 1 char and different from '\s+'
are inte rpreted as regex); you can avoid this warning by specifying
engine='python'.   df =
pd.read_csv(path,delimiter='[\t]+',header=None,names=features + ['targe t'])
```

	Area			Length of Perimeter kernel	Width of Compactness coefficient	Asymmetry groove.	target kernel	
0	15.26	14.84	0.8710	5.763	3.312	2.221	5.220	1
1	14.88	14.57	0.8811	5.554	3.333	1.018	4.956	1
2	14.29	14.09	0.9050	5.291	3.337	2.699	4.825	1
3	13.84	13.94	0.8955	5.324	3.379	2.259	4.805	1
4	16.14	14.99	0.9034	5.658	3.562	1.355	5.175	1
...	...	...	...	...	...	...	...	...
205	12.19	13.20	0.8783	5.137	2.981	3.631	4.870	3
206	11.23	12.88	0.8511	5.140	2.795	4.325	5.003	3
207	13.20	13.66	0.8883	5.236	3.232	8.315	5.056	3
208	11.84	13.21	0.8521	5.175	2.836	3.598	5.044	3
209	12.30	13.34	0.8684	5.243	2.974	5.637	5.063	3

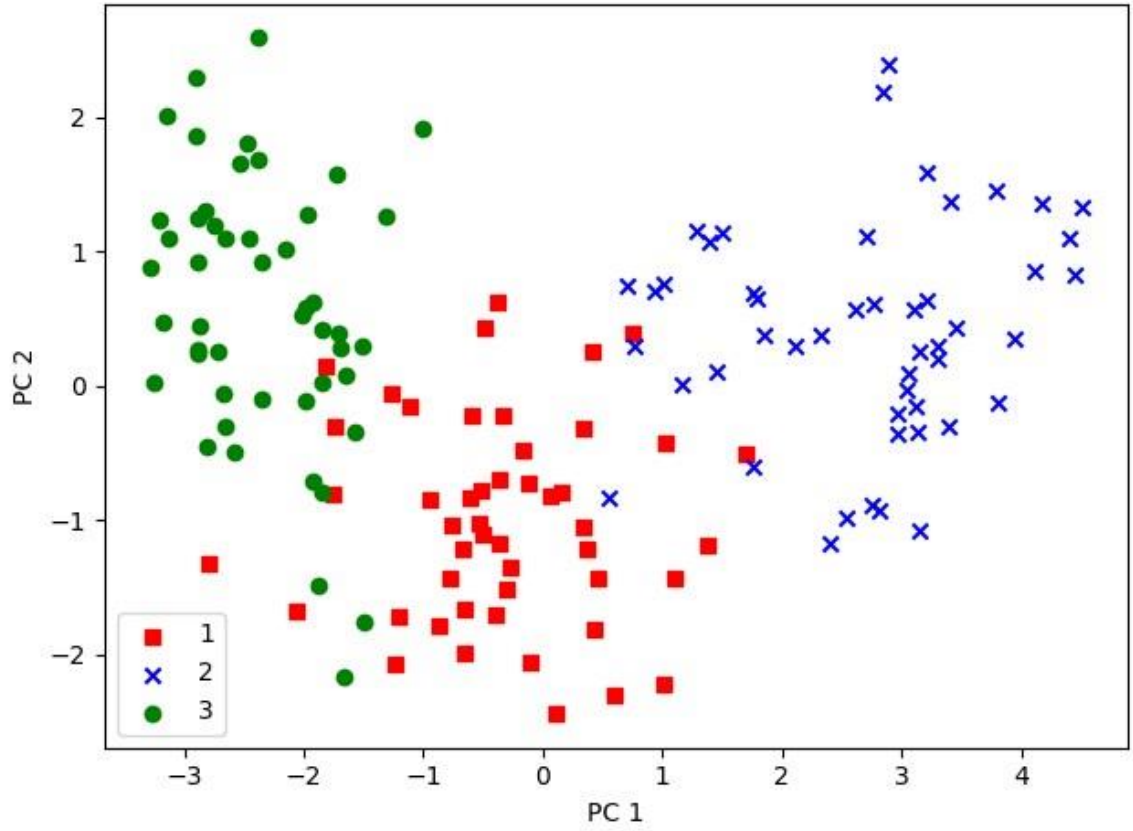
210 rows × 8 columns

```
array([[14.88  , 14.57  ,  0.8811, ...,  3.333 ,  1.018 ,  4.956 ],
[14.29  , 14.09  ,  0.905 , ...,  3.337 ,  2.699 ,  4.825 ],
       [13.84  , 13.94  ,  0.8955, ...,  3.379 ,  2.259 ,  4.805 ],
...,
       [13.2   , 13.66  ,  0.8883, ...,  3.232 ,  8.315 ,  5.056 ],
       [11.84  , 13.21  ,  0.8521, ...,  2.836 ,  3.598 ,  5.044 ],
       [12.3   , 13.34  ,  0.8684, ...,  2.974 ,  5.637 ,  5.063
]])  Eigenvalues  [5.04875044e+00  1.24521761e+00  6.67510668e-01
6.35891951e-02
 1.74990423e-02 8.53765692e-04 4.85513691e-03]

Eigenvalues
[5.04875044e+00 1.24521761e+00 6.67510668e-01 6.35891951e-02
```

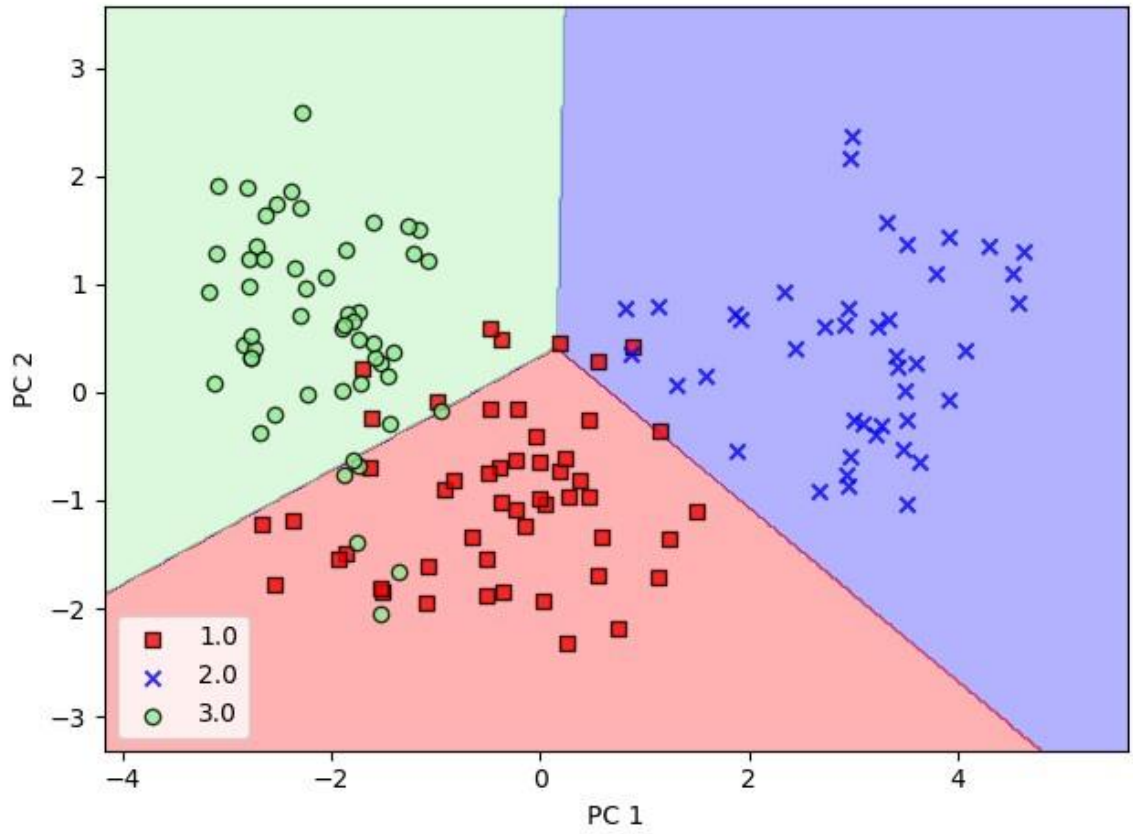
1.74990423e-02 8.53765692e-04 4.85513691e-03]




```
[(5.048750440204564,
  array([ 0.44540023,  0.4426748 ,  0.27741036,  0.4251548 ,  0.43349711,
         -0.07367381,  0.39318821])),
 (1.245217613795552,
  array([ 0.01842273,  0.07346738, -0.52672155,  0.18827822, -0.1213575 ,
          0.74344244,  0.33755677])),
(0.6675106679915351,
  array([-0.01713836,  0.07020297, -0.6287085 ,  0.22840353, -0.21756683,
         -0.66024235,  0.25314024])),
 (0.06358919514812561,
  array([ 0.19402791,  0.29417664, -0.33203905,  0.26764369,  0.1934684 ,
          0.06150257, -0.80790941])),
(0.017499042330519902,
  array([ 0.22815926,  0.19735385, -0.33010046, -0.77484338,  0.43216654,
         -0.04040314,  0.1060467 ])),
 (0.004855136906774908,
  array([-0.44369475, -0.43941243, -0.15866597,  0.23729198,  0.72502486,
         -0.02207876,  0.04920163])),
 (0.0008537656918963881,
  array([-0.7172287 ,  0.68930501,  0.07708693, -0.05747729,  0.01258185,
0.00656481,  0.03153036]))]
Matrix W:
 [[ 0.44540023  0.01842273]
 [ 0.4426748   0.07346738]
 [ 0.27741036 -0.52672155]
 [ 0.4251548   0.18827822]
 [ 0.43349711 -0.1213575 ]
 [-0.07367381  0.74344244]
 [ 0.39318821  0.33755677]]
```




```
C:\Users\prasad\AppData\Local\Temp\ipykernel_21004\701045780.py:108: Use
rWarning: You passed a edgecolor/edgecolors ('black') for an unfilled marker
('x').  Matplotlib is ignoring the edgecolor in favor of the facecolor.  This
b ehavior may change in the future.
  plt.scatter(x=X[mask, 0],
```




```
C:\Users\prasad\AppData\Local\Temp\ipykernel_21004\701045780.py:108: Use
rWarning: You passed a edgecolor/edgecolors ('black') for an unfilled marker
('x').  Matplotlib is ignoring the edgecolor in favor of the facecolor.  This
b ehavior may change in the future.
```

```
  plt.scatter(x=X[mask, 0],
```