CODE FOR RANDOMIZER

```
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\dinesh\Downloads\seeds_dataset.txt"
features = ['Area',
        'Perimeter',
        'Compactness',
        'Length of kernel',
        'Width of kernel',
        'Asymmetry coefficient',
        ' groove.']
df= pd.read_csv(path,header=None,names=features + ['target'])
display(df)
print(df.dtypes)
# Converting the features from in X Array and class in Y array
X = df.iloc[1:,[0,1,2,3,4,5,6]].values
X=X.astype('float64')
print(X.shape)
y = df.loc[1:,'target']
y=y.astype('int64')
#print (y)
print('Class labels:', np.unique(y))
random_rows = [np.random.randint(0,209,73)]
random_cols = [np.random.randint(0,7,73)]
random_rows,random_cols
```

```
X[random_rows, random_cols] = np.nan
display(X)
#Counting the numbers of NaN value in Dataframe
np.count_nonzero(np.isnan(X))
import numpy as np
#Create NumPy arrays
arr = np.array(X)
arr1 = np.array(y)
# Use concatenate() to join two arrays
con = np.column_stack((arr, arr1))
print(con.shape)
display(con)
#Rechecking number of data values in array
np.count_nonzero(np.isnan(X))
#Converting the concatenated array into df
import numpy as np
import pandas as pd
df = pd.DataFrame(con)
display (df)
#Checking number of Nan values in df
df.isnull().sum()
#Replacing the Nan values with Mean
from sklearn.impute import SimpleImputer
import numpy as np
imr = SimpleImputer(missing_values=np.nan, strategy='mean')
imr = imr.fit(df.values)
imputed_data = imr.transform(df.values)
imputed_data
#Printing imputed df
df = pd.DataFrame(imputed_data)
display (df)
```

```
#Checking number of Nan values in df
df.isnull().sum()
X = df.iloc[:, [2,4]].values
X=X.astype('float64')
#print(X)
y = df.loc[:, 7].values
y=y.astype('int64')
#print (y)
print('Class labels:', np.unique(y))
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1,stratify=y)
print('Labels counts in y:', np.bincount(y)[1:])
print('Labels counts in y_train:', np.bincount(y_train)[1:])
print('Labels counts in y_test:', np.bincount(y_test)[1:])
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
def plot_decision_regions(X, y, classifier, test_idx=None, resolution=0.02):
    # X = X.to_numpy()
    # y = y.to_numpy()
    # setup marker generator and color map
    markers = ('o', 's', '^', '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))
lab = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
lab = lab.reshape(xx1.shape)
plt.contourf(xx1, xx2, lab, alpha=0.5, cmap=cmap)
plt.xlim(xx1.min(), xx1.max())
plt.ylim(xx2.min(), xx2.max())

   # plot class examples
for idx, cl in enumerate(np.unique(y)):
      plt.scatter(x=X[y == cl, 0],
                  y=X[y == cl, 1],
                  alpha=1.0,
                  c=colors[idx],
                  marker=markers[idx],
                  label=f'Class {cl}',
                  edgecolor='black')
   # highlight test examples
if test_idx:
  X_test, y_test = X[test_idx, :], y[test_idx]
  plt.scatter(X_test[:, 0],
                  X_test[:, 1],
                  c='none',
                  edgecolor='black',
                  alpha=1.5,
                  linewidth=1,
                  marker='o',
                  s=150,
                  label='Test set')


# Logistic Regression Classifier
```

```
log_reg = LogisticRegression()

log_reg.fit(X_train_std,y_train)

y_pred= log_reg.predict(X_test_std)

print('predicted:',y_pred)

print('true class:', np.array(y_test))

plot_decision_regions(X, y, classifier=logistic_reg)

plt.xlabel(df.columns[2])

plt.ylabel(df.columns[4])

plt.legend(loc='upper left')

plt.tight_layout()

plt.show()
```

Output Random and learning

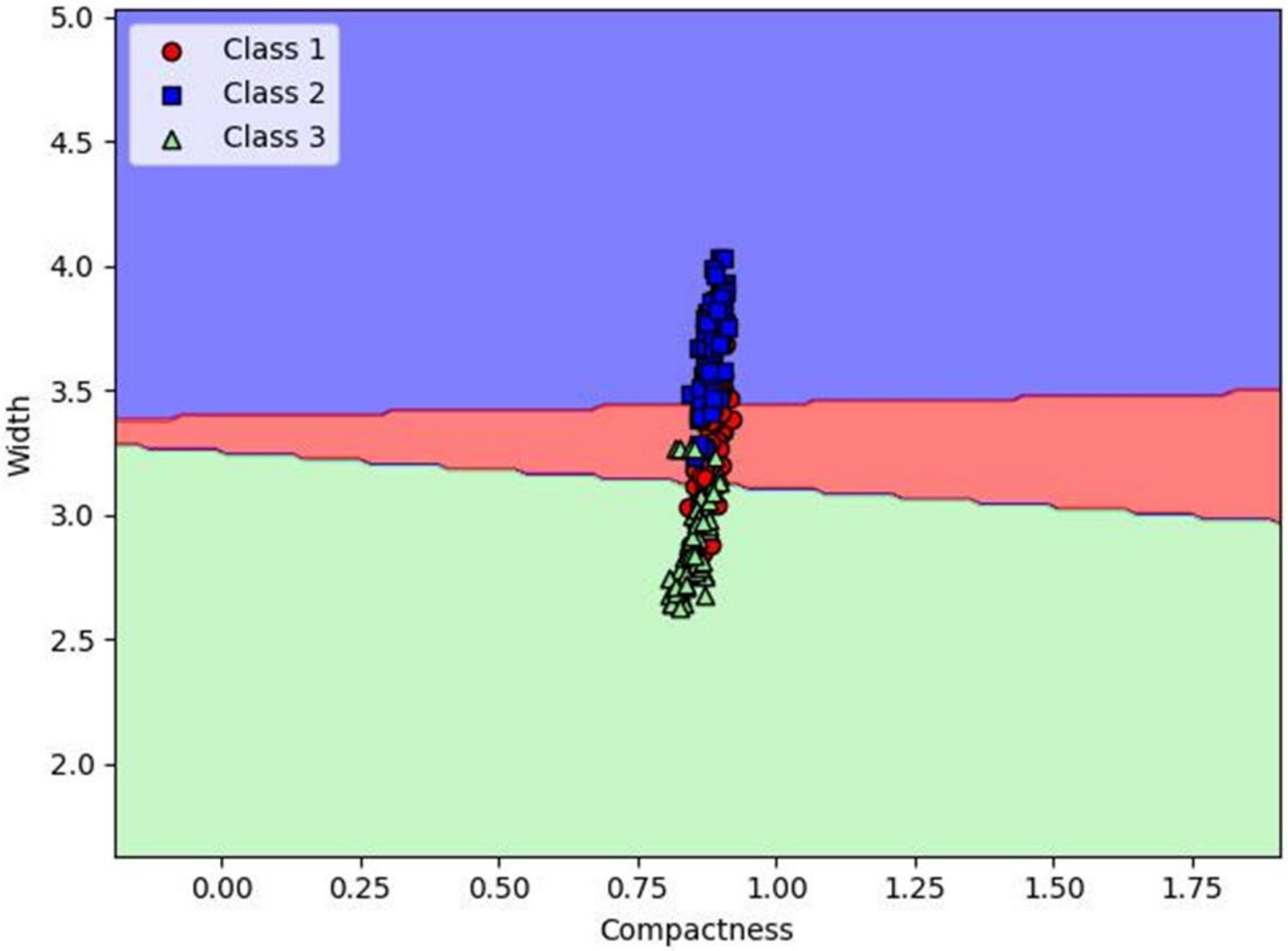
```
      Area  Perimeter  Compactness  ...  Asymmetry coefficient   groove.  target
0    15.26      14.84       0.8710  ...                  2.221     5.220       1
1    14.88      14.57       0.8811  ...                  1.018     4.956       1
2    14.29      14.09       0.9050  ...                  2.699     4.825       1
3    13.84      13.94       0.8955  ...                  2.259     4.805       1
4    16.14      14.99       0.9034  ...                  1.355     5.175       1
..     ...        ...          ...  ...                    ...       ...     ...
205  12.19      13.20       0.8783  ...                  3.631     4.870       3
206  11.23      12.88       0.8511  ...                  4.325     5.003       3
207  13.20      13.66       0.8883  ...                  8.315     5.056       3
208  11.84      13.21       0.8521  ...                  3.598     5.044       3
209  12.30      13.34       0.8684  ...                  5.637     5.063       3

[210 rows x 8 columns]
Area                    float64
Perimeter               float64
Compactness             float64
Length                  float64
Width                   float64
Asymmetry coefficient   float64
 groove.                float64
target                    int64
dtype: object
(209, 7)
Class labels: [1 2 3]
[[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 ]
 ...
 [   nan     nan  0.8883 ...  3.232   8.315   5.056 ]
 [11.84   13.21    0.8521 ...  2.836   3.598   5.044 ]
```

```
          0      1       2      3      4      5      6    7
0    14.88  14.57  0.8811  5.554  3.333  1.018  4.956  1.0
1    14.29  14.09  0.9050  5.291  3.337  2.699  4.825  1.0
2    13.84  13.94  0.8955  5.324  3.379  2.259  4.805  1.0
3    16.14  14.99  0.9034  5.658  3.562  1.355  5.175  1.0
4    14.38  14.21  0.8951  5.386  3.312    NaN    NaN  1.0
..     ...    ...     ...    ...    ...    ...    ...  ...
204  12.19  13.20  0.8783  5.137  2.981  3.631  4.870  3.0
205  11.23  12.88  0.8511  5.140    NaN  4.325  5.003  3.0
206    NaN    NaN  0.8883  5.236  3.232  8.315  5.056  3.0
207  11.84  13.21  0.8521  5.175  2.836  3.598  5.044  3.0
208  12.30  13.34  0.8684  5.243  2.974  5.637  5.063  3.0

[209 rows x 8 columns]
             0         1       2      3         4         5         6    7
0    14.880000  14.570000  0.8811  5.554  3.333000  1.018000  4.956000  1.0
1    14.290000  14.090000  0.9050  5.291  3.337000  2.699000  4.825000  1.0
2    13.840000  13.940000  0.8955  5.324  3.379000  2.259000  4.805000  1.0
3    16.140000  14.990000  0.9034  5.658  3.562000  1.355000  5.175000  1.0
4    14.380000  14.210000  0.8951  5.386  3.312000  3.689448  5.425418  1.0
..         ...        ...     ...    ...       ...       ...       ...  ...
204  12.190000  13.200000  0.8783  5.137  2.981000  3.631000  4.870000  3.0
205  11.230000  12.880000  0.8511  5.140  3.267493  4.325000  5.003000  3.0
206  14.871256  14.546856  0.8883  5.236  3.232000  8.315000  5.056000  3.0
207  11.840000  13.210000  0.8521  5.175  2.836000  3.598000  5.044000  3.0
208  12.300000  13.340000  0.8684  5.243  2.974000  5.637000  5.063000  3.0

[209 rows x 8 columns]
Class labels: [1 2 3]
Labels counts in y: [69 70 70]
Labels counts in y_train: [34 35 35]
```


Class 1
Class 2
Class 3
5.0
4.5
4.0
3.5
3.0
2.5
2.0
Width
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
Compactness