```
#  ------------------------------------------------
#                   Expt - 9.a
#  ------------------------------------------------
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report
from matplotlib.colors import ListedColormap

# Load Iris dataset
iris = load_iris()
X = iris.data[:, :2]  # Use first two features for 2D visualization
y = iris.target
target_names = iris.target_names

# Split and scale
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Elbow method to find optimal k
error_rates = []
k_range = range(1, 21)
for k in k_range:
    knn_temp = KNeighborsClassifier(n_neighbors=k)
    knn_temp.fit(X_train_scaled, y_train)
    y_pred_temp = knn_temp.predict(X_test_scaled)
    error = np.mean(y_pred_temp != y_test)
    error_rates.append(error)

# Plot Elbow Method
plt.figure(figsize=(8, 6))
plt.plot(k_range, error_rates, marker='o', linestyle='--', color='purple')
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Error Rate')
plt.xticks(k_range)
plt.grid(True)
plt.show()
```

```
# Train KNN model with chosen k
k = 5
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train_scaled, y_train)

# Predict and evaluate
y_pred = knn.predict(X_test_scaled)
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred, target_names=target_names))

# Create mesh grid for plotting
h = 0.02
x_min, x_max = X_train_scaled[:, 0].min() - 1, X_train_scaled[:, 0].max() + 1
y_min, y_max = X_train_scaled[:, 1].min() - 1, X_train_scaled[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot combined decision boundary
plt.figure(figsize=(8, 6))
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])
plt.contourf(xx, yy, Z, cmap=cmap_light)
plt.scatter(X_train_scaled[:, 0], X_train_scaled[:, 1], c=y_train, cmap=cmap_bold, edgecolor='k', s=40)
plt.title(f'Combined KNN Decision Boundary (k={k})')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()

# Plot separate graphs for each class
colors = ['red', 'green', 'blue']
for class_idx in range(3):
    plt.figure(figsize=(8, 6))
    plt.contourf(xx, yy, Z, cmap=ListedColormap(['white', 'white', 'white']))
    mask = y_train == class_idx
    plt.scatter(X_train_scaled[mask, 0], X_train_scaled[mask, 1], color=colors[class_idx], edgecolor='k', s=40,
label=target_names[class_idx])
    plt.title(f'Class: {target_names[class_idx]}')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.legend()
    plt.grid(True)
    plt.show()
```

```
Confusion Matrix:
 [[10  0  0]
 [ 0  7  2]
 [ 0  3  8]]

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       0.70      0.78      0.74         9
   virginica       0.80      0.73      0.76        11

    accuracy                           0.83        30
   macro avg       0.83      0.84      0.83        30
weighted avg       0.84      0.83      0.83        30
```

```
# ---------------------------------------------------
#                   Expt - 9.b
# ---------------------------------------------------
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score
from scipy.stats import mode

# Load and scale the two most discriminative features
iris = load_iris()
X = iris.data[:, 2:4]  # Petal length and petal width
y_true = iris.target
X_scaled = StandardScaler().fit_transform(X)

# Apply K-Means with optimized parameters
kmeans = KMeans(n_clusters=3, init='k-means++', n_init=50, max_iter=500, random_state=42)
kmeans.fit(X_scaled)
labels = kmeans.labels_

# Align cluster labels with true labels
def align_labels(true_labels, cluster_labels):
    label_map = {}
    for i in np.unique(cluster_labels):
        mask = cluster_labels == i
        most_common = mode(true_labels[mask], keepdims=True)[0][0]
        label_map[i] = most_common
    return np.vectorize(label_map.get)(cluster_labels)
```

```
aligned_labels = align_labels(y_true, labels)
accuracy = accuracy_score(y_true, aligned_labels) * 100
sample_preds = [np.float64(aligned_labels[i]) for i in range(3)]

# Plot clusters and centers
centers = kmeans.cluster_centers_
plt.figure(figsize=(8, 6))
colors = ['red', 'green', 'blue']
for i in range(3):
    plt.scatter(X_scaled[labels == i, 0], X_scaled[labels == i, 1], s=50, c=colors[i], label=f'Cluster {i}')
plt.scatter(centers[:, 0], centers[:, 1], s=200, c='yellow', marker='X', label='Centers')
plt.title('K-Means Clustering on Iris Dataset (Petal Features)')
plt.xlabel('Petal Length (scaled)')
plt.ylabel('Petal Width (scaled)')
plt.legend()
plt.grid(True)
plt.show()

# Output results
print(f"Predictions: {sample_preds}")
print(f"Accuracy: {accuracy:.2f} %")
```

```
Predictions: [np.float64(0.0), np.float64(0.0), np.float64(0.0)]
Accuracy: 96.00 %
```