#8. **Write a program to demonstrate the working of the decision tree. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample**

# Import necessary libraries

from sklearn.datasets import load\_iris

from sklearn.tree import DecisionTreeClassifier, plot\_tree

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

import matplotlib.pyplot as plt

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Split into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Create and train the decision tree classifier

clf = DecisionTreeClassifier(criterion='entropy', random\_state=42)

clf.fit(X\_train, y\_train)

# Predict on test data

y\_pred = clf.predict(X\_test)

# Evaluate accuracy

print(f"Accuracy: {accuracy\_score(y\_test, y\_pred):.2f}")

# Visualize the decision tree

plt.figure(figsize=(12, 8))

plot\_tree(clf, filled=True, feature\_names=iris.feature\_names, class\_names=iris.target\_names)

plt.title("Decision Tree for Iris Dataset")

plt.show()

# Classify a new sample

# Example: [sepal length, sepal width, petal length, petal width]

new\_sample = [[5.1, 3.5, 1.4, 0.2]] # Likely Iris-setosa

prediction = clf.predict(new\_sample)

print(f"\nPrediction for new sample {new\_sample}: {iris.target\_names[prediction[0]]}")

**9.Write a program to implement Naive Bayes algorithm to classify the iris data set. Print both correct and wrong predictions.**

# Import necessary libraries

from sklearn.datasets import load\_iris

from sklearn.naive\_bayes import GaussianNB

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

class\_names = iris.target\_names

# Split the dataset

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Create and train the Naive Bayes model

model = GaussianNB()

model.fit(X\_train, y\_train)

# Predict on test set

y\_pred = model.predict(X\_test)

# Accuracy

print(f"Accuracy: {accuracy\_score(y\_test, y\_pred):.2f}\n")

# Print correct and wrong predictions

print("Predictions on Test Set:")

for i in range(len(y\_test)):

predicted = y\_pred[i]

actual = y\_test[i]

sample = X\_test[i]

status = "✔ Correct" if predicted == actual else "✘ Wrong"

print(f"Sample {i+1}: Predicted = {class\_names[predicted]}, Actual =

{class\_names[actual]} --> {status}")

**10.Write a program to implement clustering using the k-Means algorithm using an**

**appropriate dataset.**

# Import necessary libraries

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.cluster import KMeans

from sklearn.decomposition import PCA

from sklearn.metrics import confusion\_matrix

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

target\_names = iris.target\_names

# Apply KMeans clustering

kmeans = KMeans(n\_clusters=3, random\_state=42)

kmeans.fit(X)

y\_kmeans = kmeans.labels\_

# Evaluate clustering with confusion matrix (optional)

print("Confusion Matrix (cluster vs actual labels):")

print(confusion\_matrix(y, y\_kmeans))

# Reduce dimensions for visualization using PCA

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X)

# Plot clustering result

plt.figure(figsize=(8, 6))

colors = ['red', 'green', 'blue']

for i in range(3):

plt.scatter(X\_pca[y\_kmeans == i, 0], X\_pca[y\_kmeans == i, 1],

label=f"Cluster {i}", c=colors[i], edgecolor='k')

plt.title('K-Means Clustering of Iris Dataset')

plt.xlabel('PCA Component 1')

plt.ylabel('PCA Component 2')

plt.legend()

plt.grid(True)

plt.show()