```python
# prgm-1
import csv

hypo = ['%', '%', '%', '%', '%', '%']

with open('enjoysport.csv', 'r') as csvfile:
    lines = csv.reader(csvfile, delimiter=',')
    data = []
    for line in lines:
        if line[len(line) - 1].upper() == 'YES':
            data.append(line[:-1])

print("Positive eg are: ")
for line in data:
    print(line)

print("Steps of candidate elimination algo are: ")
print(hypo)

hypo = data[0]

for i in range(len(data)):
    for j in range(len(data[0])):
        if hypo[j] != data[i][j]:
            hypo[j] = '?'
    print(hypo)

print("The maximally specific Find-s hypothesis for the given training examples is")
print(hypo)
```

```
# prgm-2
import numpy as np
import pandas as pd

data = pd.DataFrame(data=pd.read_csv(r"candidate.csv"))

concepts = np.array(data.iloc[:, 0:-1])
target = np.array(data.iloc[:, -1])

def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("initialization of specific_h and general_h")
    print(specific_h)

    general_h = [["?" for i in range(len(specific_h))] for i in
range(len(specific_h))]
    print(general_h)

    for i, h in enumerate(concepts):
        if target[i] == "Yes":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'
        if target[i] == "No":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
        print(" steps of Candidate Elimination Algorithm", i + 1)
        print(specific_h)
        print(general_h)

    indices = [i for i, val in enumerate(general_h) if val == ['?', '?',
'?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])

    return specific_h, general_h

s_final, g_final = learn(concepts, target)
print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

```python
# prgm-3
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn import tree

iris = load_iris()
print(iris.feature_names)
print(iris.target_names)

removed = [0, 50, 100]
new_target = np.delete(iris.target, removed)
new_data = np.delete(iris.data, removed, axis=0)

clf = tree.DecisionTreeClassifier()
clf = clf.fit(new_data, new_target)
prediction = clf.predict(iris.data[removed])

print("Original Labels", iris.target[removed])
print("Labels Predicted", prediction)
tree.plot_tree(clf)
```

```
# prgm-4
import numpy as np

class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size,
learning_rate):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.learning_rate = learning_rate

        self.weights_input_hidden = np.random.randn(self.input_size,
self.hidden_size)
        self.bias_input_hidden = np.random.randn(1, self.hidden_size)
        self.weights_hidden_output = np.random.randn(self.hidden_size,
self.output_size)
        self.bias_hidden_output = np.random.randn(1, self.output_size)

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def sigmoid_derivative(self, x):
        return x * (1 - x)

    def forward(self, inputs):
        self.hidden_output = self.sigmoid(np.dot(inputs,
self.weights_input_hidden) + self.bias_input_hidden)
        self.output = self.sigmoid(np.dot(self.hidden_output,
self.weights_hidden_output) + self.bias_hidden_output)

    def backward(self, inputs, targets):
        output_error = targets - self.output
        output_delta = output_error *
self.sigmoid_derivative(self.output)

        hidden_error = output_delta.dot(self.weights_hidden_output.T)
        hidden_delta = hidden_error *
self.sigmoid_derivative(self.hidden_output)

        self.weights_hidden_output +=
self.hidden_output.T.dot(output_delta) * self.learning_rate
        self.bias_hidden_output += np.sum(output_delta, axis=0,
keepdims=True) * self.learning_rate
        self.weights_input_hidden += inputs.T.dot(hidden_delta) *
self.learning_rate
        self.bias_input_hidden += np.sum(hidden_delta, axis=0,
keepdims=True) * self.learning_rate
```

```
    def train(self, inputs, targets, epochs):
        for epoch in range(epochs):
            self.forward(inputs)
            self.backward(inputs, targets)

    def predict(self, inputs):
        self.forward(inputs)
        return self.output

inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
targets = np.array([[0], [1], [1], [0]])

input_size = 2
hidden_size = 3
output_size = 1
learning_rate = 0.1
epochs = 10000
nn = NeuralNetwork(input_size, hidden_size, output_size, learning_rate)

nn.train(inputs, targets, epochs)

predictions = nn.predict(inputs)
print("Predictions:")
print(predictions)
```

```
# prgm-5
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score,
recall_score

newsgroups_train = fetch_20newsgroups(subset='train')
newsgroups_test = fetch_20newsgroups(subset='test')

vectorizer = CountVectorizer()
X_train = vectorizer.fit_transform(newsgroups_train.data)
X_test = vectorizer.transform(newsgroups_test.data)

y_train = newsgroups_train.target
y_test = newsgroups_test.target

nb_classifier = MultinomialNB()
nb_classifier.fit(X_train, y_train)

y_pred = nb_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
```

```
# prgm-6
import bayespy as bp
import numpy as np
import csv
from colorama import init
from colorama import Fore, Back, Style
init()

ageEnum = {'SuperSeniorCitizen': 0, 'SeniorCitizen': 1, 'MiddleAged': 2,
'Youth': 3, 'Teen': 4}
genderEnum = {'Male': 0, 'Female': 1}
familyHistoryEnum = {'Yes': 0, 'No': 1}
dietEnum = {'High': 0, 'Medium': 1, 'Low': 2}
lifeStyleEnum = {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedetary':
3}
cholesterolEnum = {'High': 0, 'BorderLine': 1, 'Normal': 2}
heartDiseaseEnum = {'Yes': 0, 'No': 1}

with open('heart_disease_data.csv') as csvfile:
    lines = csv.reader(csvfile)
    dataset = list(lines)
    data = []
    for x in dataset:
        data.append([ageEnum[x[0]], genderEnum[x[1]],
familyHistoryEnum[x[2]], dietEnum[x[3]], lifeStyleEnum[x[4]],
cholesterolEnum[x[5]], heartDiseaseEnum[x[6]]])

data = np.array(data)
N = len(data)

p_age = bp.nodes.Dirichlet(1.0 * np.ones(5))
age = bp.nodes.Categorical(p_age, plates=(N,))
age.observe(data[:, 0])

p_gender = bp.nodes.Dirichlet(1.0 * np.ones(2))
gender = bp.nodes.Categorical(p_gender, plates=(N,))
gender.observe(data[:, 1])

p_familyhistory = bp.nodes.Dirichlet(1.0 * np.ones(2))
familyhistory = bp.nodes.Categorical(p_familyhistory, plates=(N,))
familyhistory.observe(data[:, 2])

p_diet = bp.nodes.Dirichlet(1.0 * np.ones(3))
diet = bp.nodes.Categorical(p_diet, plates=(N,))
diet.observe(data[:, 3])
p_lifestyle = bp.nodes.Dirichlet(1.0 * np.ones(4))
lifestyle = bp.nodes.Categorical(p_lifestyle, plates=(N,))
lifestyle.observe(data[:, 4])
```

```
p_cholesterol = bp.nodes.Dirichlet(1.0 * np.ones(3))
cholesterol = bp.nodes.Categorical(p_cholesterol, plates=(N,))
cholesterol.observe(data[:, 5])

p_heartdisease = bp.nodes.Dirichlet(np.ones(2), plates=(5, 2, 2, 3, 4,
3))
heartdisease = bp.nodes.MultiMixture([age, gender, familyhistory, diet,
lifestyle, cholesterol], bp.nodes.Categorical, p_heartdisease)
heartdisease.observe(data[:, 6])
p_heartdisease.update()

m = 0
while m == 0:
    print("\n")
    res = bp.nodes.MultiMixture(
        [int(input('Enter Age: ' + str(ageEnum))), int(input('Enter
Gender: ' + str(genderEnum))), int(input('Enter FamilyHistory: ' +
str(familyHistoryEnum))), int(input('Enter dietEnum: ' +
str(dietEnum))), int(input('Enter LifeStyle: ' + str(lifeStyleEnum))),
int(input('Enter Cholesterol: ' + str(cholesterolEnum)))],
        bp.nodes.Categorical,
        p_heartdisease
    ).get_moments()[0][heartDiseaseEnum['Yes']]
    print("Probability(HeartDisease) = " + str(res))
    m = int(input("Enter for Continue: 0, Exit: 1 "))
```

```
# prgm-7
import numpy as np
import pandas as pd

np.random.seed(0)
num_samples = 100
num_features = 2
data = np.random.rand(num_samples, num_features)
df = pd.DataFrame(data, columns=['X', 'Y'])
df.to_csv('test_data.csv', index=False)

print("CSV file 'test_data.csv' has been generated successfully.")
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
import matplotlib.pyplot as plt
data = pd.read_csv('test_data.csv')
X = data.values
k = 3
kmeans = KMeans(n_clusters=k)
kmeans.fit(X)
kmeans_labels = kmeans.labels_
kmeans_centers = kmeans.cluster_centers_

em = GaussianMixture(n_components=k)
em.fit(X)
em_labels = em.predict(X)
em_centers = em.means_

print("K-means labels:")
print(kmeans_labels)
print("EM labels:")
print(em_labels)

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.scatter(X[:, 0], X[:, 1], c=kmeans_labels, cmap='viridis')
plt.scatter(kmeans_centers[:, 0], kmeans_centers[:, 1], marker='*',
s=300, c='r')
plt.title('K-means Clustering')

plt.subplot(1, 2, 2)
plt.scatter(X[:, 0], X[:, 1], c=em_labels, cmap='viridis')
plt.scatter(em_centers[:, 0], em_centers[:, 1], marker='*', s=300,
c='r')
plt.title('EM Clustering')

plt.show()
```

```
# prgm-8
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

iris = load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

k = 3
knn = KNeighborsClassifier(n_neighbors=k)

knn.fit(X_train, y_train)

predictions = knn.predict(X_test)

accuracy = accuracy_score(y_test, predictions)
print("Accuracy:", accuracy)

correct_predictions = []
wrong_predictions = []
for i in range(len(predictions)):
    if predictions[i] == y_test[i]:
        correct_predictions.append((X_test[i], y_test[i],
predictions[i]))
    else:
        wrong_predictions.append((X_test[i], y_test[i], predictions[i]))

print("\nCorrect Predictions:")
for prediction in correct_predictions:
    print("Input:", prediction[0], "Actual Class:",
iris.target_names[prediction[1]], "Predicted Class:",
iris.target_names[prediction[2]])
print("\nWrong Predictions:")
for prediction in wrong_predictions:
    print("Input:", prediction[0], "Actual Class:",
iris.target_names[prediction[1]], "Predicted Class:",
iris.target_names[prediction[2]])
conf_matrix = confusion_matrix(y_test, predictions)
print("\nConfusion Matrix:")
print(conf_matrix)
```

```
# prgm-9
import numpy as np
import matplotlib.pyplot as plt

def locally_weighted_regression(x_query, X_train, y_train, tau=0.1):
    m = X_train.shape[0]
    weights = np.exp(-np.sum((X_train - x_query) ** 2, axis=1) / (2 *
tau * tau))
    W = np.diag(weights)
    theta = np.linalg.inv(X_train.T @ W @ X_train) @ (X_train.T @ W @
y_train)
    prediction = x_query @ theta
    return prediction

np.random.seed(0)
X_train = np.linspace(0, 10, 50)
y_train = np.sin(X_train) + np.random.normal(0, 0.1, X_train.shape[0])

X_query = np.linspace(0, 10, 100)

tau = 0.5

predictions = []
for xq in X_query:
    x_query = np.array([1, xq])
    prediction = locally_weighted_regression(x_query,
np.c_[np.ones(X_train.shape[0]), X_train], y_train, tau)
    predictions.append(prediction)

plt.figure(figsize=(10, 6))
plt.scatter(X_train, y_train, color='blue', label='Training data')
plt.plot(X_query, predictions, color='red', label='Locally Weighted
Regression')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Locally Weighted Regression')
plt.legend()
plt.grid(True)
plt.show()
```

```
# PRGM-10
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

iris = datasets.load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

svm_classifier = SVC(kernel='linear', C=1.0, random_state=42)
svm_classifier.fit(X_train, y_train)

y_pred = svm_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```