Program 1

Read a dataset from the user and i. Use the Find-S algorithm to find the most specific hypothesis that is consistent with the positive examples. Ii. What is the final hypothesis after processing all the positive examples? Using the same dataset, apply the Candidate Elimination algorithm. Determine the final version space after processing all examples (both positive and negative). What are the most specific and most general hypotheses in the version space?

In [3]:
```
def get_input_data():
    print("Enter the dataset row-by-row as comma-separated values")
    print("Enter an empty line to finish.\n")
    dataset = []
    while True:
        row = input("Enter example: ").strip()
        if not row:
            break
        parts = row.split(",")
        dataset.append(parts)
    return dataset
def find_s_algorithm(data):
    print("\n=== Find-S Algorithm ===")
    hypothesis = None
    for row in data:
        *attributes, label = row
        if label.lower() == "yes":
            if hypothesis is None:
                hypothesis = attributes.copy()
            else:
                for i in range(len(attributes)):
                    if hypothesis[i] != attributes[i]:
                        hypothesis[i] = '?'
    print("Final Hypothesis (Find-S):", hypothesis)
    return hypothesis
def more_general(h1, h2):
    more_general_parts = []
    for x, y in zip(h1, h2):
        mg = x == '?' or (x != y and x != '0')
        more_general_parts.append(mg)
    return all(more_general_parts)
def generalize_S(example, S):
    for i in range(len(S)):
        if S[i] != example[i]:
            S[i] = '?'
    return S
def specialize_G(example, G, attributes):
    new_G = []
    for g in G:
        for i in range(len(g)):
            if g[i] == '?':
                for value in attributes[i]:
                    if value != example[i]:
                        new_hypothesis = g.copy()
                        new_hypothesis[i] = value
                        new_G.append(new_hypothesis)
    return new_G
def candidate_elimination_algorithm(data):
    print("\n=== Candidate Elimination Algorithm ===")
    attributes = [list(set([row[i] for row in data])) for i in range(len(data[0]) - 1)]
    S = ['0'] * (len(data[0]) - 1)
    G = [['?'] * (len(data[0]) - 1)]
    for row in data:
        *x, label = row
        if label.lower() == 'yes':
            G = [g for g in G if all(g[i] == '?' or g[i] == x[i] for i in range(len(g)))]
            if S == ['0'] * len(x):
                S = x.copy()
            else:
                S = generalize_S(x, S)
```

```
            G = [g for g in G if more_general(g, S)]
        else:
            if all(S[i] == x[i] or S[i] == '?' for i in range(len(x))):
                S = ['0'] * len(x)
            G_temp = []
            for g in G:
                if all(g[i] == '?' or g[i] == x[i] for i in range(len(x))):
                    G_temp += specialize_G(x, [g], attributes)
                else:
                    G_temp.append(g)
            G = []
            for h in G_temp:
                if not any(more_general(h2, h) and h != h2 for h2 in G_temp):
                    G.append(h)
    print("Final Specific Hypothesis (S):", S)
    print("Final General Hypotheses (G):", G)
    return S, G
if __name__ == "__main__":
    data = get_input_data()
    final_hypothesis = find_s_algorithm(data)
    S_final, G_final = candidate_elimination_algorithm(data)
    print("\n=== Results ===")
    print("Find-S Final Hypothesis:", final_hypothesis)
    print("Candidate Elimination Version Space:")
    print("Most Specific Hypothesis:", S_final)
    print("Most General Hypotheses:")
    for g in G_final:
        print(g)
```

```
Enter the dataset row-by-row as comma-separated values
Enter an empty line to finish.

Enter example: sunny,warm,normal,strong,warm,same,yes,sunny,warm,high,strong,warm,same,yes,rainy,cloud,hig
h,strong,warm,change,yes,sunny,warm,high,strong,cool,change,yes
Enter example:

=== Find-S Algorithm ===
Final Hypothesis (Find-S): ['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes', 'sunny', 'warm', 'h
igh', 'strong', 'warm', 'same', 'yes', 'rainy', 'cloud', 'high', 'strong', 'warm', 'change', 'yes', 'sunn
y', 'warm', 'high', 'strong', 'cool', 'change']

=== Candidate Elimination Algorithm ===
Final Specific Hypothesis (S): ['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes', 'sunny', 'war
m', 'high', 'strong', 'warm', 'same', 'yes', 'rainy', 'cloud', 'high', 'strong', 'warm', 'change', 'yes',
'sunny', 'warm', 'high', 'strong', 'cool', 'change']
Final General Hypotheses (G): [['?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?',
'?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?']]

=== Results ===
Find-S Final Hypothesis: ['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes', 'sunny', 'warm', 'hig
h', 'strong', 'warm', 'same', 'yes', 'rainy', 'cloud', 'high', 'strong', 'warm', 'change', 'yes', 'sunny',
'warm', 'high', 'strong', 'cool', 'change']
Candidate Elimination Version Space:
Most Specific Hypothesis: ['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes', 'sunny', 'warm', 'hi
gh', 'strong', 'warm', 'same', 'yes', 'rainy', 'cloud', 'high', 'strong', 'warm', 'change', 'yes', 'sunny',
'warm', 'high', 'strong', 'cool', 'change']
Most General Hypotheses:
['?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?',
'?', '?', '?', '?', '?', '?', '?']
```

Program 2

Read a dataset and use an example-based method (such as RIPPER or CN2) to generate a set of classification rules. Apply the FOIL algorithm (First-Order Inductive Learner) to learn first-order rules for predicting.

In [ ]:

```
import pandas as pd
import Orange
from sklearn.datasets import load_iris

data = load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['species'] = pd.Categorical.from_codes(data.target, data.target_names)
df.to_csv("iris.csv", index=False)

def load_dataset(csv_path):
    df = pd.read_csv(csv_path)

    attributes = []
    for col in df.columns[:-1]:
        if df[col].dtype == 'object':
            values = list(map(str, df[col].unique()))
            attributes.append(Orange.data.DiscreteVariable(col, values))
        else:
            attributes.append(Orange.data.ContinuousVariable(col))

    class_col = df.columns[-1]
    class_values = list(map(str, df[class_col].unique()))
    class_var = Orange.data.DiscreteVariable(class_col, class_values)

    domain = Orange.data.Domain(attributes, class_var)

    data_as_str = df.astype(str).values.tolist()
    table = Orange.data.Table.from_list(domain, data_as_str)

    return table

def apply_cn2_learner(table):
    learner = Orange.classification.rules.CN2Learner()
    classifier = learner(table)
    return classifier

def apply_foil_like_learner(table):
    learner = Orange.classification.rules.CN2SDUnorderedLearner()
    classifier = learner(table)
    return classifier

def display_rules(classifier):
    print("\nLearned Rules:\n")
    for rule in classifier.rule_list:
        print(rule)

def main():
    csv_path = "iris.csv"
    table = load_dataset(csv_path)

    print("=== CN2 RULES ===")
    cn2_classifier = apply_cn2_learner(table)
    display_rules(cn2_classifier)

    print("\n=== FOIL-LIKE RULES ===")
    foil_classifier = apply_foil_like_learner(table)
    display_rules(foil_classifier)

if __name__ == "__main__":
    main()
```

```
=== CN2 RULES ===

Learned Rules:

IF petal length (cm)<=3.0 AND sepal width (cm)>=2.9 THEN species=setosa
IF petal width (cm)>=1.8 AND sepal length (cm)>=6.0 THEN species=virginica
IF sepal length (cm)>=4.9 AND sepal width (cm)>=3.1 THEN species=versicolor
IF petal length (cm)<=4.9 AND petal width (cm)>=1.7 THEN species=virginica
IF petal width (cm)>=1.8 THEN species=virginica
IF petal length (cm)<=5.0 AND sepal width (cm)>=2.4 THEN species=versicolor
IF sepal width (cm)>=2.8 THEN species=virginica
IF petal width (cm)<=1.0 AND sepal length (cm)>=5.0 THEN species=versicolor
IF sepal width (cm)>=2.7 THEN species=versicolor
IF sepal width (cm)>=2.6 THEN species=virginica
IF sepal length (cm)>=5.5 AND sepal length (cm)>=6.2 THEN species=versicolor
IF sepal length (cm)<=5.5 AND petal length (cm)>=4.0 THEN species=versicolor
IF sepal length (cm)>=6.0 THEN species=virginica
IF sepal length (cm)<=4.5 THEN species=setosa
IF TRUE THEN species=virginica

=== FOIL-LIKE RULES ===

Learned Rules:

IF petal length (cm)<=3.0 THEN species=setosa
IF petal width (cm)<=1.0 AND petal length (cm)<=3.3 THEN species=setosa
IF petal length (cm)<=3.0 AND sepal width (cm)>=2.9 THEN species=setosa
IF petal length (cm)<=3.0 AND sepal width (cm)>=2.9 AND sepal length (cm)>=4.4 THEN species=setosa
IF petal length (cm)<=3.0 AND sepal width (cm)>=2.9 AND sepal width (cm)>=3.0 THEN species=setosa
IF sepal length (cm)<=5.8 AND petal width (cm)<=1.0 THEN species=setosa
IF sepal length (cm)<=5.8 AND sepal length (cm)<=5.5 AND petal length (cm)<=3.0 THEN species=setosa
IF petal width (cm)<=1.0 THEN species=setosa
IF petal length (cm)<=3.0 AND sepal width (cm)>=2.9 AND petal length (cm)>=1.1 THEN species=setosa
IF sepal width (cm)>=3.0 AND petal length (cm)<=4.1 THEN species=setosa
IF sepal width (cm)>=3.0 AND sepal length (cm)<=5.9 AND petal width (cm)<=1.2 THEN species=setosa
IF sepal length (cm)<=6.2 AND sepal length (cm)<=5.8 AND petal width (cm)<=1.7 THEN species=setosa
IF petal length (cm)>=3.0 AND petal width (cm)<=1.8 AND petal width (cm)<=1.7 AND petal length (cm)<=5.0 TH
EN species=versicolor
IF petal length (cm)>=3.0 AND petal width (cm)<=1.8 AND petal width (cm)<=1.7 THEN species=versicolor
IF petal length (cm)>=3.0 AND petal width (cm)<=1.7 AND petal length (cm)<=5.0 AND sepal length (cm)>=5.0 T
HEN species=versicolor
IF petal length (cm)>=3.0 AND petal length (cm)<=4.8 THEN species=versicolor
IF petal length (cm)>=3.0 AND petal width (cm)<=1.7 AND petal length (cm)<=5.0 AND petal width (cm)<=1.5 TH
EN species=versicolor
IF petal length (cm)>=3.0 AND petal width (cm)<=1.8 THEN species=versicolor
IF petal length (cm)>=3.0 AND petal width (cm)<=1.8 AND sepal width (cm)<=3.3 THEN species=versicolor
IF petal length (cm)>=3.0 AND petal width (cm)<=1.7 AND petal length (cm)<=5.0 AND sepal length (cm)>=5.0 A
ND sepal width (cm)>=2.3 THEN species=versicolor
IF sepal width (cm)<=3.4 AND sepal length (cm)>=5.5 AND petal width (cm)<=1.8 THEN species=versicolor
IF petal length (cm)>=3.0 AND petal length (cm)<=4.8 AND petal width (cm)<=1.7 THEN species=versicolor
IF sepal width (cm)<=3.4 AND sepal length (cm)>=5.5 AND petal width (cm)<=1.8 AND sepal width (cm)<=3.3 THE
N species=versicolor
IF sepal width (cm)<=3.4 AND sepal length (cm)>=5.5 AND petal length (cm)<=4.8 THEN species=versicolor
IF sepal width (cm)<=3.4 AND petal length (cm)>=3.0 THEN species=versicolor
IF petal length (cm)>=4.8 THEN species=virginica
IF petal width (cm)>=1.8 THEN species=virginica
IF petal length (cm)>=5.0 THEN species=virginica
IF petal width (cm)>=1.7 THEN species=virginica
IF sepal width (cm)<=3.4 AND petal length (cm)>=4.9 THEN species=virginica
IF sepal length (cm)>=5.5 AND petal length (cm)>=4.8 AND sepal width (cm)<=3.3 THEN species=virginica
IF sepal length (cm)>=5.5 AND petal width (cm)>=1.8 AND petal length (cm)>=4.9 THEN species=virginica
IF sepal length (cm)>=5.5 AND sepal length (cm)>=5.9 AND petal width (cm)>=1.8 THEN species=virginica
IF sepal width (cm)<=3.1 AND petal width (cm)>=1.7 THEN species=virginica
IF petal length (cm)>=5.2 THEN species=virginica
IF petal length (cm)>=3.0 AND sepal width (cm)<=3.6 AND petal width (cm)>=1.7 THEN species=virginica
IF petal length (cm)>=3.0 AND sepal width (cm)<=3.6 AND petal length (cm)>=5.0 THEN species=virginica
IF sepal width (cm)<=3.4 AND petal length (cm)>=3.0 AND petal width (cm)>=1.7 THEN species=virginica
IF sepal width (cm)<=3.4 AND petal length (cm)>=3.0 AND petal length (cm)>=5.0 THEN species=virginica
IF petal length (cm)>=3.0 THEN species=virginica
IF TRUE THEN species=versicolor
```

Program 3

Read a supervised dataset and use bagging and boosting technique to classify the dataset. Indicate the performance of the model.

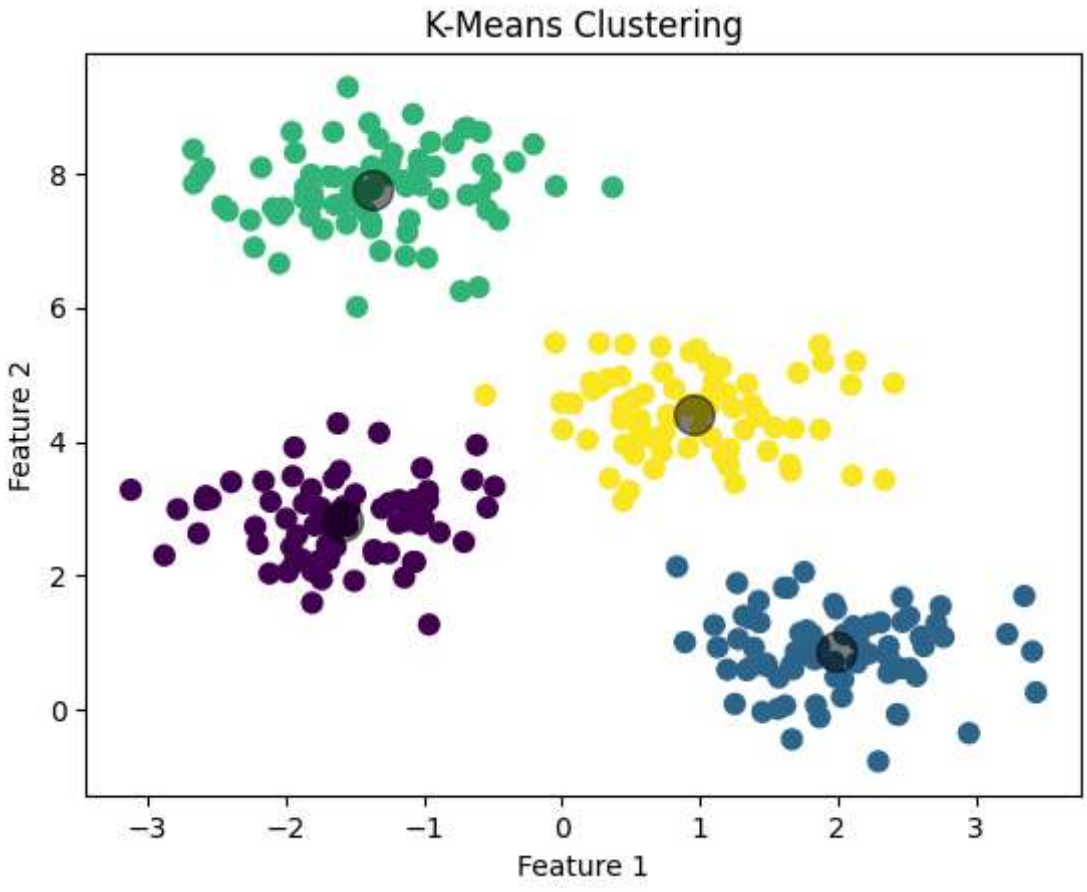
In [ ]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingClassifier, AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

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

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

bagging = BaggingClassifier(
    estimator=DecisionTreeClassifier(),
    n_estimators=50,
    random_state=42
)
bagging.fit(X_train, y_train)
y_pred_bag = bagging.predict(X_test)

boosting = AdaBoostClassifier(
    estimator=DecisionTreeClassifier(max_depth=1),
    n_estimators=50,
    random_state=42
)
boosting.fit(X_train, y_train)
y_pred_boost = boosting.predict(X_test)

print("=== Bagging Performance ===")
print("Accuracy:", accuracy_score(y_test, y_pred_bag))
print(classification_report(y_test, y_pred_bag))

print("\n=== Boosting Performance ===")
print("Accuracy:", accuracy_score(y_test, y_pred_boost))
print(classification_report(y_test, y_pred_boost))
```

```
=== Bagging Performance ===
Accuracy: 0.9666666666666667
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      0.90      0.95        10
           2       0.91      1.00      0.95        10

    accuracy                           0.97        30
   macro avg       0.97      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30


=== Boosting Performance ===
Accuracy: 0.9333333333333333
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       0.90      0.90      0.90        10
           2       0.90      0.90      0.90        10

    accuracy                           0.93        30
   macro avg       0.93      0.93      0.93        30
weighted avg       0.93      0.93      0.93        30
```

Program 4

Read an unsupervised dataset and group the dataset based on similarity based on k-means clustering.

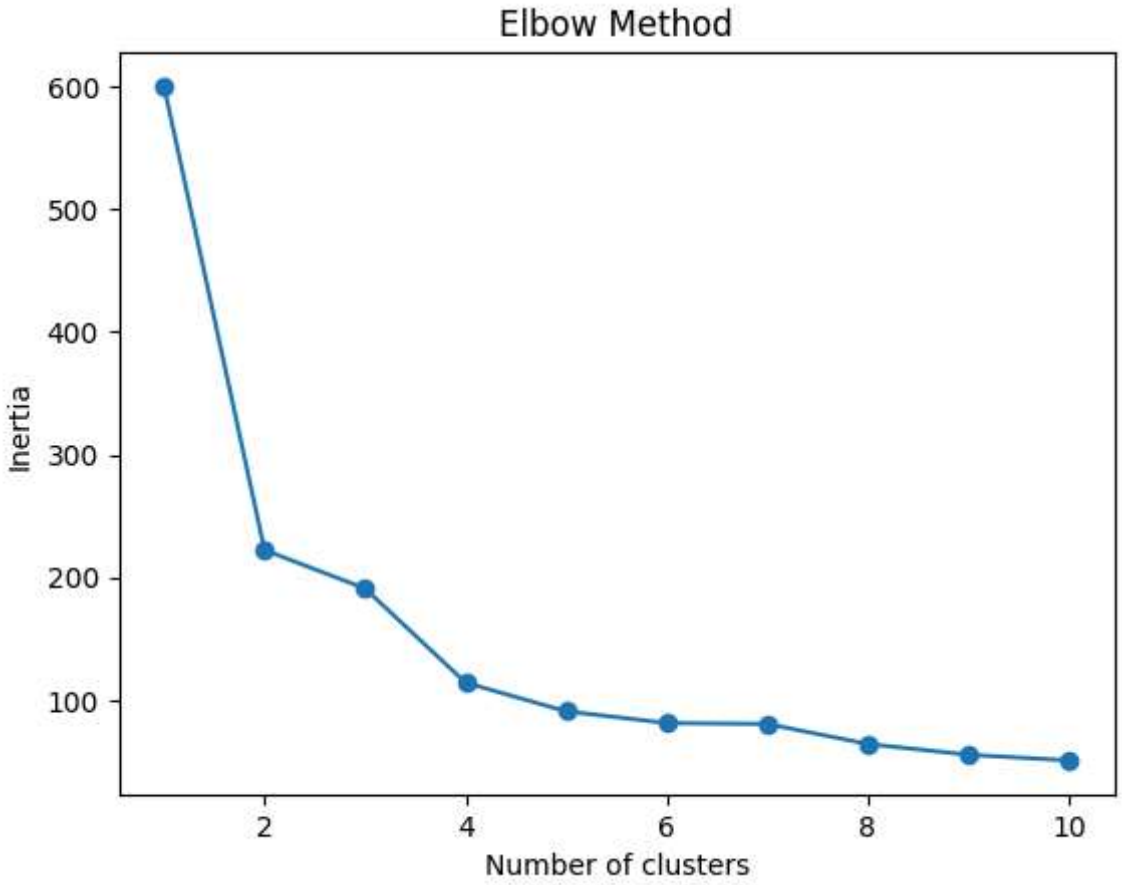
In [ ]:
```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

X, y = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

kmeans = KMeans(n_clusters=4, n_init=10)
y_kmeans = kmeans.predict(X)

plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')

centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)
plt.title('K-Means Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```



In [ ]:
```
from sklearn.datasets import load_iris
import pandas as pd
data = load_iris()
X = pd.DataFrame(data.data, columns=data.feature_names)
```

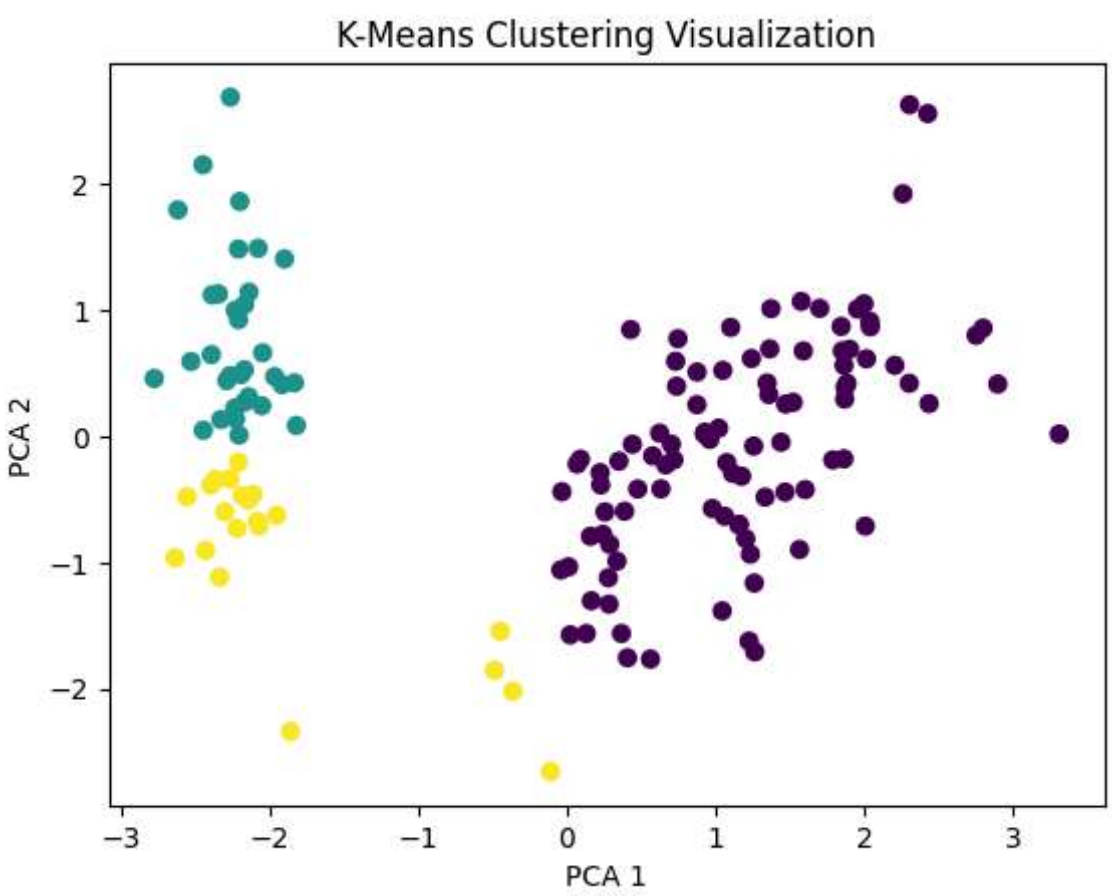
In [ ]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [ ]:
```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

inertia = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

plt.plot(range(1, 11), inertia, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method')
plt.show()
```



In [ ]:
```
kmeans = KMeans(n_clusters=3, random_state=42)
clusters = kmeans.fit_predict(X_scaled)
X['Cluster'] = clusters
```

In [ ]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=clusters, cmap='viridis')
plt.title('K-Means Clustering Visualization')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.show()
```


K-Means Clustering Visualization
PCA 2
PCA 1
2
1
0
−1
−2
−3
−2
−1
0
1
2
3

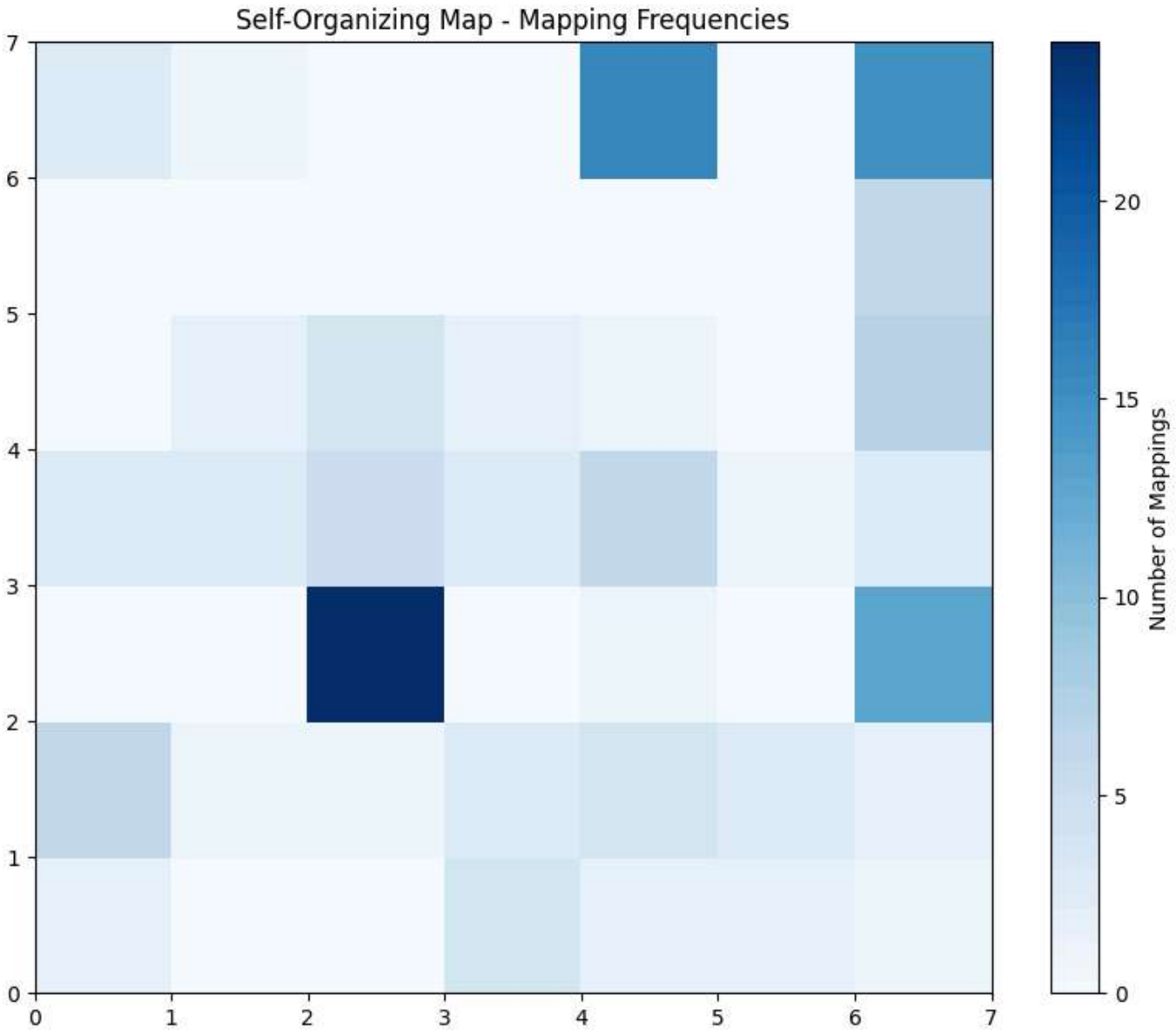
Program 5

Read a dataset and perform unsupervised learning using SOM algorithm.

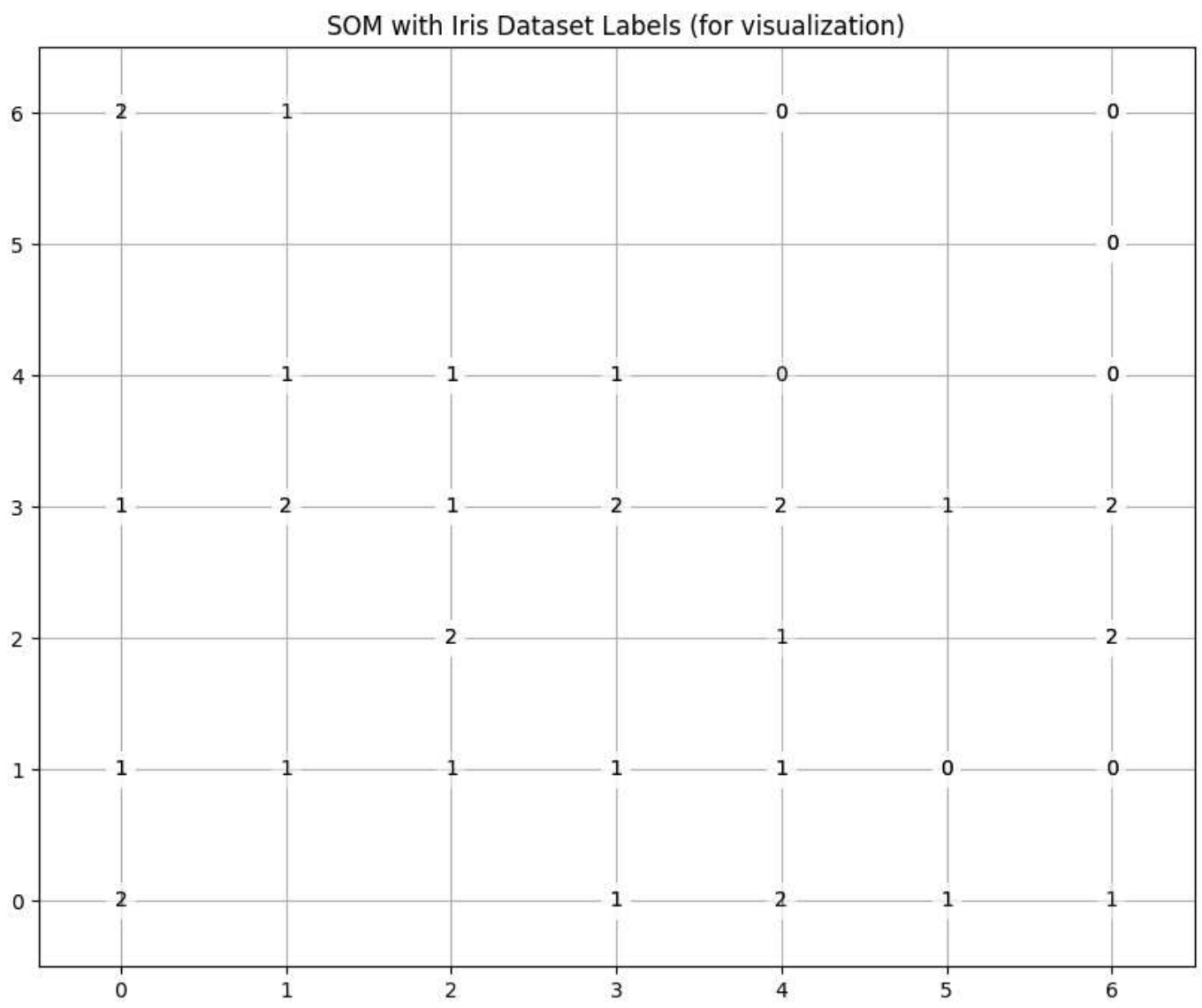
In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.datasets import load_iris
from minisom import MiniSom
```

In [ ]:
```
data = load_iris()
X = data.data
y = data.target

scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

som = MiniSom(x=7, y=7, input_len=X.shape[1], sigma=1.0, learning_rate=0.5)
som.random_weights_init(X_scaled)
som.train_random(data=X_scaled, num_iteration=100)
```

In [ ]:
```
plt.figure(figsize=(10, 8))
frequencies = np.zeros((7, 7))
for x in X_scaled:
    w = som.winner(x)
    frequencies[w[0], w[1]] += 1

plt.pcolor(frequencies.T, cmap='Blues')
plt.colorbar(label='Number of Mappings')
plt.title("Self-Organizing Map - Mapping Frequencies")
plt.show()
```



In [ ]:
```
plt.figure(figsize=(10, 8))
for i, x in enumerate(X_scaled):
    w = som.winner(x)
    plt.text(w[0], w[1], str(y[i]),
             ha='center', va='center',
             bbox=dict(facecolor='white', alpha=0.5, lw=0))

plt.title("SOM with Iris Dataset Labels (for visualization)")
plt.xlim(-0.5, 6.5)
plt.ylim(-0.5, 6.5)
plt.grid()
plt.show()
```


SOM with Iris Dataset Labels (for visualization)
6
5
4
3
2
1
0
0 1 2 3 4 5 6
2 1 0 0
0
1 1 1 0 0
1 2 1 2 2 1 2
2 1 2
1 1 1 1 1 0 0
2 1 2 1 1

Program 6

Write a function to generate uniform random numbers in the interval [0, 1]. Use this function to generate 10 random samples and evaluate f(x) for each sample. What are the sampled function values? Using the samples generated in the previous step, estimate the integral I using the Monte Carlo method.

In [ ]:
```python
import random
def generate_uniform():
    return random.uniform(0, 1)

def f(x):
    return x**2

samples = [generate_uniform() for _ in range(10)]
f_values = [f(x) for x in samples]

for i, (x, fx) in enumerate(zip(samples, f_values), 1):
    print(f"Sample {i}: x = {x:.4f}, f(x) = {fx:.4f}")

I_estimate = sum(f_values) / len(f_values)
print(f"\nEstimated integral I using Monte Carlo method: {I_estimate:.4f}")
```

```
Sample 1: x = 0.3084, f(x) = 0.0951
Sample 2: x = 0.4687, f(x) = 0.2197
Sample 3: x = 0.1529, f(x) = 0.0234
Sample 4: x = 0.6344, f(x) = 0.4024
Sample 5: x = 0.7167, f(x) = 0.5136
Sample 6: x = 0.7785, f(x) = 0.6060
Sample 7: x = 0.8678, f(x) = 0.7532
Sample 8: x = 0.2429, f(x) = 0.0590
Sample 9: x = 0.2963, f(x) = 0.0878
Sample 10: x = 0.8678, f(x) = 0.7530

Estimated integral I using Monte Carlo method: 0.3513
```

Program 7

Read a dataset and indicate the likelihood of an event occurring using Bayesian Networks.

In [ ]:
```
!pip install pgmpy
```

In [ ]:
```
import pandas as pd
from pgmpy.models import DiscreteBayesianNetwork
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.inference import VariableElimination
data = pd.DataFrame([
    ['Sunny', 'Hot', 'High', 'False', 'No'],
    ['Sunny', 'Hot', 'High', 'True', 'No'],
    ['Overcast', 'Hot', 'High', 'False', 'Yes'],
    ['Rain', 'Mild', 'High', 'False', 'Yes'],
    ['Rain', 'Cool', 'Normal', 'False', 'Yes'],
    ['Rain', 'Cool', 'Normal', 'True', 'No'],
    ['Overcast', 'Cool', 'Normal', 'True', 'Yes'],
    ['Sunny', 'Mild', 'High', 'False', 'No'],
    ['Sunny', 'Cool', 'Normal', 'False', 'Yes'],
    ['Rain', 'Mild', 'Normal', 'False', 'Yes'],
    ['Sunny', 'Mild', 'Normal', 'True', 'Yes'],
    ['Overcast', 'Mild', 'High', 'True', 'Yes'],
    ['Overcast', 'Hot', 'Normal', 'False', 'Yes'],
    ['Rain', 'Mild', 'High', 'True', 'No']
], columns=['Outlook', 'Temperature', 'Humidity', 'Windy', 'Play'])

model = DiscreteBayesianNetwork([
    ('Outlook', 'Play'),
    ('Temperature', 'Play'),
    ('Humidity', 'Play'),
    ('Windy', 'Play')
])

model.fit(data, estimator=MaximumLikelihoodEstimator)
print("\nLearned CPDs:")
for cpd in model.get_cpds():
    print(cpd)
infer = VariableElimination(model)
query_result = infer.query(
    variables=['Play'],
    evidence={'Outlook': 'Sunny', 'Humidity': 'High'}
)
print("\nProbability of Play given Outlook='Sunny' and Humidity='High':")
print(query_result)
```

```
Learned CPDs:
+------------------+----------+
| Outlook(Overcast) | 0.285714 |
+------------------+----------+
| Outlook(Rain)     | 0.357143 |
+------------------+----------+
| Outlook(Sunny)    | 0.357143 |
+------------------+----------+
+-------------+------------------+-----+------------------+
| Humidity    | Humidity(High)    | ... | Humidity(Normal)  |
+-------------+------------------+-----+------------------+
| Outlook     | Outlook(Overcast) | ... | Outlook(Sunny)    |
+-------------+------------------+-----+------------------+
| Temperature | Temperature(Cool) | ... | Temperature(Mild) |
+-------------+------------------+-----+------------------+
| Windy       | Windy(False)      | ... | Windy(True)       |
+-------------+------------------+-----+------------------+
| Play(No)    | 0.5               | ... | 0.0               |
+-------------+------------------+-----+------------------+
| Play(Yes)   | 0.5               | ... | 1.0               |
+-------------+------------------+-----+------------------+
+------------------+----------+
| Temperature(Cool) | 0.285714 |
+------------------+----------+
| Temperature(Hot)  | 0.285714 |
+------------------+----------+
| Temperature(Mild) | 0.428571 |
+------------------+----------+
+------------------+-----+
| Humidity(High)   | 0.5 |
+------------------+-----+
| Humidity(Normal) | 0.5 |
+------------------+-----+
+--------------+----------+
| Windy(False) | 0.571429 |
+--------------+----------+
| Windy(True)  | 0.428571 |
+--------------+----------+

Probability of Play given Outlook='Sunny' and Humidity='High':
+-----------+-------------+
| Play      |   phi(Play) |
+===========+=============+
| Play(No)  |      0.7653 |
+-----------+-------------+
| Play(Yes) |      0.2347 |
+-----------+-------------+
```

Program 8

Refer to the dataset in question 7 and indicate inferences based on the sequence of steps.

In [ ]:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier, export_text, plot_tree
import matplotlib.pyplot as plt

data = pd.DataFrame([
    ['Sunny', 'Hot', 'High', 'False', 'No'],
    ['Sunny', 'Hot', 'High', 'True', 'No'],
    ['Overcast', 'Hot', 'High', 'False', 'Yes'],
    ['Rain', 'Mild', 'High', 'False', 'Yes'],
    ['Rain', 'Cool', 'Normal', 'False', 'Yes'],
    ['Rain', 'Cool', 'Normal', 'True', 'No'],
    ['Overcast', 'Cool', 'Normal', 'True', 'Yes'],
    ['Sunny', 'Mild', 'High', 'False', 'No'],
    ['Sunny', 'Cool', 'Normal', 'False', 'Yes'],
    ['Rain', 'Mild', 'Normal', 'False', 'Yes'],
    ['Sunny', 'Mild', 'Normal', 'True', 'Yes'],
    ['Overcast', 'Mild', 'High', 'True', 'Yes'],
    ['Overcast', 'Hot', 'Normal', 'False', 'Yes'],
    ['Rain', 'Mild', 'High', 'True', 'No']
], columns=['Outlook', 'Temperature', 'Humidity', 'Windy', 'Play'])
label_encoders = {}
for column in data.columns:
    le = LabelEncoder()
    data[column] = le.fit_transform(data[column])
    label_encoders[column] = le

X = data.drop('Play', axis=1)
y = data['Play']

clf = DecisionTreeClassifier(criterion='entropy')
clf.fit(X, y)
feature_names = X.columns
tree_rules = export_text(clf, feature_names=list(feature_names))
print(tree_rules)

plt.figure(figsize=(12, 6))
plot_tree(clf, feature_names=feature_names, class_names=label_encoders['Play'].classes_, fill
plt.title("Decision Tree")
plt.show()
```

```
|--- Outlook <= 0.50
|   |--- class: 1
|--- Outlook >  0.50
|   |--- Humidity <= 0.50
|   |   |--- Outlook <= 1.50
|   |   |   |--- Windy <= 0.50
|   |   |   |   |--- class: 1
|   |   |   |--- Windy >  0.50
|   |   |   |   |--- class: 0
|   |   |--- Outlook >  1.50
|   |   |   |--- class: 0
|   |--- Humidity >  0.50
|   |   |--- Windy <= 0.50
|   |   |   |--- class: 1
|   |   |--- Windy >  0.50
|   |   |   |--- Temperature <= 1.00
|   |   |   |   |--- class: 0
|   |   |   |--- Temperature >  1.00
|   |   |   |   |--- class: 1
```