```
pip install pandas mlxtend
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: mlxtend in /usr/local/lib/python3.11/dist-packages (0.23.4)
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: scipy>=1.2.1 in /usr/local/lib/python3.11/dist-packages (from mlxtend) (1.15.2)
Requirement already satisfied: scikit-learn>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from mlxtend) (1.6.1)
Requirement already satisfied: matplotlib>=3.0.0 in /usr/local/lib/python3.11/dist-packages (from mlxtend) (3.10.0)
Requirement already satisfied: joblib>=0.13.2 in /usr/local/lib/python3.11/dist-packages (from mlxtend) (1.4.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.0.0->mlxtend) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.0.0->mlxtend) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.0.0->mlxtend) (4.57.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.0.0->mlxtend) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.0.0->mlxtend) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.0.0->mlxtend) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.0.0->mlxtend) (3.2.3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=1.3.1->mlxtend) (3.6.
```

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import fpgrowth, association_rules

# Dataset mẫu
dataset = [
    ['A', 'B', 'C'],
    ['B', 'C', 'D'],
    ['A', 'C', 'D', 'E'],
    ['A', 'D', 'E'],
    ['A', 'B', 'C', 'E']
]

min_support_count = 2 # Ngưỡng hỗ trợ tuyệt đối
num_transactions = len(dataset)
min_support_ratio = min_support_count / num_transactions # Ngưỡng hỗ trợ tương đối

# Chuẩn bị dữ liệu One-Hot
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df_onehot = pd.DataFrame(te_ary, columns=te.columns_)

print("DataFrame One-Hot Encoded:")
print(df_onehot)
print(f"\\nmin_support ratio ({min_support_count}/{num_transactions}): {min_support_ratio}")
print("-" * 30)
```

```
DataFrame One-Hot Encoded:
       A      B      C      D      E
0   True   True   True  False  False
1  False   True   True   True  False
2   True  False   True   True   True
3   True  False  False   True   True
4   True   True   True  False   True
\nmin_support ratio (2/5): 0.4
------------------------------
```

```
# Áp dụng thuật toán FP-Growth
frequent_itemsets = fpgrowth(df_onehot, min_support=min_support_ratio, use_colnames=True)

print(f"Các tập phổ biến (min_support={min_support_ratio}):")
print(frequent_itemsets)
print("-" * 30)
```

```
Các tập phổ biến (min_support=0.4):
    support   itemsets
0       0.8        (C)
1       0.8        (A)
2       0.6        (B)
3       0.6        (D)
4       0.6        (E)
5       0.6     (A, C)
```

```
6       0.6     (C, B)
7       0.4     (A, B)
8       0.4  (A, C, B)
9       0.4     (D, C)
10      0.4     (D, E)
11      0.4     (D, A)
12      0.4  (D, A, E)
13      0.6     (A, E)
14      0.4     (C, E)
15      0.4  (A, C, E)
------------------------------
```

```
# Tạo luật kết hợp
rules_confidence = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)

print("Luật kết hợp (min_confidence=0.7):")
# Sắp xếp kết quả theo confidence và lift để dễ phân tích
rules_confidence = rules_confidence.sort_values(['confidence', 'lift'], ascending=[False, False])
print(rules_confidence[['antecedents', 'consequents', 'support', 'confidence', 'lift']])
print("-" * 30)
```

```
Luật kết hợp (min_confidence=0.7):
  antecedents consequents  support  confidence      lift
5      (D, A)         (E)      0.4        1.00  1.666667
3         (B)         (C)      0.6        1.00  1.250000
4      (A, B)         (C)      0.4        1.00  1.250000
6      (D, E)         (A)      0.4        1.00  1.250000
8         (E)         (A)      0.6        1.00  1.250000
9      (C, E)         (A)      0.4        1.00  1.250000
2         (C)         (B)      0.6        0.75  1.250000
7         (A)         (E)      0.6        0.75  1.250000
0         (A)         (C)      0.6        0.75  0.937500
1         (C)         (A)      0.6        0.75  0.937500
------------------------------
```

```
# Tạo luật kết hợp
rules_confidence = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)

print("Luật kết hợp (min_confidence=0.7):")
# Sắp xếp kết quả theo confidence và lift để dễ phân tích
rules_confidence = rules_confidence.sort_values(['confidence', 'lift'], ascending=[False, False])
print(rules_confidence[['antecedents', 'consequents', 'support', 'confidence', 'lift']])
print("-" * 30)
```

```
Luật kết hợp (min_confidence=0.7):
  antecedents consequents  support  confidence      lift
5      (D, A)         (E)      0.4        1.00  1.666667
3         (B)         (C)      0.6        1.00  1.250000
4      (A, B)         (C)      0.4        1.00  1.250000
6      (D, E)         (A)      0.4        1.00  1.250000
8         (E)         (A)      0.6        1.00  1.250000
9      (C, E)         (A)      0.4        1.00  1.250000
2         (C)         (B)      0.6        0.75  1.250000
7         (A)         (E)      0.6        0.75  1.250000
0         (A)         (C)      0.6        0.75  0.937500
1         (C)         (A)      0.6        0.75  0.937500
------------------------------
```

```
rules_lift = association_rules(frequent_itemsets, metric="lift", min_threshold=1.0)

print("Luật kết hợp (min_lift=1.0):")
rules_lift = rules_lift.sort_values(['lift', 'confidence'], ascending=[False, False])
print(rules_lift[['antecedents', 'consequents', 'support', 'confidence', 'lift']])
print("-" * 30)
```

```
Luật kết hợp (min_lift=1.0):
   antecedents consequents  support  confidence      lift
8       (D, A)         (E)      0.4    1.000000  1.666667
13         (E)      (D, A)      0.4    0.666667  1.666667
1          (B)         (C)      0.6    1.000000  1.250000
3       (A, B)         (C)      0.4    1.000000  1.250000
9       (D, E)         (A)      0.4    1.000000  1.250000
15         (E)         (A)      0.6    1.000000  1.250000
17      (C, E)         (A)      0.4    1.000000  1.250000
4          (C)      (A, B)      0.4    0.500000  1.250000
```

```
12         (A)      (D, E)      0.4    0.500000  1.250000
18         (A)      (C, E)      0.4    0.500000  1.250000
0          (C)         (B)      0.6    0.750000  1.250000
14         (A)         (E)      0.6    0.750000  1.250000
2       (A, C)         (B)      0.4    0.666667  1.111111
5          (B)      (A, C)      0.4    0.666667  1.111111
6          (D)         (E)      0.4    0.666667  1.111111
7          (E)         (D)      0.4    0.666667  1.111111
10      (A, E)         (D)      0.4    0.666667  1.111111
11         (D)      (A, E)      0.4    0.666667  1.111111
16      (A, C)         (E)      0.4    0.666667  1.111111
19         (E)      (A, C)      0.4    0.666667  1.111111
------------------------------
```

```
# === Bước 1: Chuẩn bị đồ chơi ===
# Mình cần mấy món đồ chơi này để làm việc

import numpy as np # Đồ chơi xử lý số liệu (như bảng tính Excel mini)
import matplotlib.pyplot as plt # Đồ chơi để vẽ vời đồ thị, hình ảnh
from sklearn.tree import DecisionTreeClassifier, plot_tree # Đồ chơi chính: Máy học đoán Gà/Vịt và vẽ cái cây quyết định của nó
from sklearn.metrics import accuracy_score, classification_report # Đồ chơi để xem máy đoán đúng bao nhiêu %

# === Bước 2: Tự chế dữ liệu Gà và Vịt ===
# Tưởng tượng mình đo mỏ và chân của 15 con Vịt, 15 con Gà

# Đặt tên cho dễ nhớ
ten_dac_trung = ['Độ dài mỏ (cm)', 'Chiều dài chân (cm)'] # Mình sẽ đo 2 cái này
ten_loai = ['Vịt', 'Gà'] # Có 2 loại là Vịt (số 0) và Gà (số 1)

# ---- Dữ liệu Vịt (Gán nhãn là số 0) ----
np.random.seed(42) # Để lần nào chạy số liệu cũng giống nhau, dễ kiểm tra
mo_vit = np.random.uniform(3.0, 5.0, 15)  # Đo mỏ 15 con vịt (từ 3 đến 5 cm)
chan_vit = np.random.uniform(1.5, 3.5, 15) # Đo chân 15 con vịt (từ 1.5 đến 3.5 cm)
so_lieu_vit = np.column_stack((mo_vit, chan_vit)) # Ghép số đo mỏ và chân vịt thành bảng
nhan_vit = np.zeros(15, dtype=int) # Tạo 15 nhãn số 0 (0 là Vịt)

# ---- Dữ liệu Gà (Gán nhãn là số 1) ----
mo_ga = np.random.uniform(1.0, 3.2, 15)  # Đo mỏ 15 con gà (từ 1 đến 3.2 cm - hơi ngắn hơn vịt)
chan_ga = np.random.uniform(2.5, 4.5, 15) # Đo chân 15 con gà (từ 2.5 đến 4.5 cm - hơi dài hơn vịt)
so_lieu_ga = np.column_stack((mo_ga, chan_ga)) # Ghép số đo mỏ và chân gà thành bảng
nhan_ga = np.ones(15, dtype=int) # Tạo 15 nhãn số 1 (1 là Gà)

# ---- Gom tất cả lại ----
X = np.vstack((so_lieu_vit, so_lieu_ga)) # Chồng bảng vịt lên bảng gà -> có bảng số liệu 30 con
y = np.concatenate((nhan_vit, nhan_ga)) # Nối nhãn vịt và gà -> có danh sách 30 nhãn (0 hoặc 1)

print("--- Xong phần chuẩn bị dữ liệu ---")
print("Tổng cộng có:", X.shape[0], "con vật") # In ra tổng số con vật
print("Mỗi con có:", X.shape[1], "số đo (đặc trưng)") # In ra số lượng số đo
print("Đây là số đo của 5 con đầu tiên:\n", X[:5]) # In thử 5 dòng đầu của bảng số liệu
print("Đây là nhãn của 5 con đó (0=Vịt, 1=Gà):", y[:5]) # In nhãn tương ứng
print("-" * 30) # In dòng gạch ngang cho đẹp

# === Bước 3: Nhìn thử dữ liệu (Vẽ hình) - Không bắt buộc nhưng hay ===
plt.figure(figsize=(8, 6)) # Tạo khung ảnh kích thước 8x6
# Vẽ chấm đỏ cho Vịt
plt.scatter(X[y == 0, 0], X[y == 0, 1], color='red', label='Vịt (0)', marker='o')
# Vẽ dấu X xanh cho Gà
plt.scatter(X[y == 1, 0], X[y == 1, 1], color='blue', label='Gà (1)', marker='x')
plt.xlabel(ten_dac_trung[0]) # Ghi tên trục X (Mỏ)
plt.ylabel(ten_dac_trung[1]) # Ghi tên trục Y (Chân)
plt.title('Hình ảnh dữ liệu Gà và Vịt') # Tiêu đề hình
plt.legend() # Hiện chú thích (Đỏ là Vịt, Xanh là Gà)
plt.grid(True) # Vẽ lưới cho dễ nhìn
plt.show() # Hiển thị hình vẽ lên

# === Bước 4: Tạo "Bộ Não" Cây Quyết Định và Dạy Nó ===

# Tạo một "bộ não" cây quyết định còn trống
# Mình bảo nó đừng tạo cây phức tạp quá (max_depth=2), chỉ 2 tầng thôi cho dễ hiểu
# criterion='gini' là cách nó chọn câu hỏi hay nhất để hỏi (ví dụ: "Mỏ dài hơn 3cm không?")
may_hoc = DecisionTreeClassifier(criterion='gini', max_depth=2, random_state=42)

print("--- Bắt đầu dạy máy học ---")
# Dạy máy học bằng dữ liệu mình vừa tạo (X là số đo, y là nhãn Gà/Vịt)
# Lệnh .fit() này là máy tự nhìn dữ liệu và xây cái cây quyết định
may_hoc.fit(X, y)
print("Dạy xong!")
```

```
print("-" * 30)

# === Bước 5: Xem "Bộ Não" (Cây Quyết Định) Nó Học Được Gì ===
print("--- Đây là cái cây máy học đã học được ---")
plt.figure(figsize=(12, 8)) # Tạo khung ảnh to hơn để vẽ cây
# Vẽ cái cây ra
plot_tree(may_hoc, # Cái cây cần vẽ
          filled=True, # Tô màu cho đẹp và dễ biết kết quả
          feature_names=ten_dac_trung, # Ghi tên đặc trưng (Mỏ, Chân)
          class_names=ten_loai, # Ghi tên lớp (Vịt, Gà)
          rounded=True, # Bo tròn góc ô cho đẹp
          fontsize=12) # Chữ to hơn chút
plt.title(f"Cây Quyết Định Gà/Vịt (Độ sâu tối đa = {may_hoc.max_depth})") # Tiêu đề
plt.show() # Hiện hình cây lên

# === Bước 6: Kiểm tra xem máy học đoán giỏi cỡ nào ===
# (Lưu ý: Kiểm tra trên chính dữ liệu đã học thì thường điểm cao, không khách quan lắm)
print("--- Kiểm tra kết quả ---")
# Bảo máy đoán lại nhãn cho toàn bộ dữ liệu X
du_doan = may_hoc.predict(X)

# So sánh dự đoán với nhãn thật xem đúng bao nhiêu %
do_chinh_xac = accuracy_score(y, du_doan)
print(f"Tỷ lệ đoán đúng (trên dữ liệu đã học): {do_chinh_xac * 100:.2f}%") # In tỷ lệ %

# In thêm thông tin chi tiết (không cần hiểu sâu cái này cũng được)
print("\nBáo cáo chi tiết:")
print(classification_report(y, du_doan, target_names=ten_loai))
print("-" * 30)

# === Bước 7: Thử đoán cho con vật mới ===
# Giờ mình có 2 con vật mới, chỉ biết số đo, thử hỏi máy xem nó là Gà hay Vịt

# Con 1: Mỏ 4.0cm, Chân 2.0cm (Trông giống Vịt)
# Con 2: Mỏ 2.0cm, Chân 4.0cm (Trông giống Gà)
mau_moi = np.array([
    [4.0, 2.0],
    [2.0, 4.0]
])

print("--- Thử đoán cho con vật mới ---")
# Đưa số đo 2 con mới cho máy đoán
ket_qua_doan = may_hoc.predict(mau_moi)

# Xem máy đoán là gì
for i in range(len(mau_moi)):
    so_do = mau_moi[i] # Lấy số đo con thứ i
    nhan_doan_duoc = ket_qua_doan[i] # Lấy nhãn máy đoán (0 hoặc 1)
    ten_loai_doan_duoc = ten_loai[nhan_doan_duoc] # Đổi số 0/1 thành chữ 'Vịt'/'Gà'
    print(f"Con vật có số đo {so_do}: Máy đoán là '{ten_loai_doan_duoc}'")
```