```python
import pandas as pd
import numpy as np

# Tạo dữ liệu 50 dòng
data = {
    'ID': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
           11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
           21, 22, 13, 14, 15, 16, 17, 18, 19, 30,
           31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
           42, 42, 43, 44, 45, 46, 47, 48, 49, 50],
    'Age': [25, "NaN", 30, 45, 28, "NaN", 35, 50, 22, 40,
            33, "NaN", 27, 48, 29, 38, "NaN", 42, 31, 46,
            24, 37, "NaN", 49, 26, 34, 41, 23, "NaN", 47,
            32, 39, 28, 44, "NaN", 36, 25, 50, 31, 43,
            "NaN", 29, 35, 48, 27, 40, 33, "NaN", 46, 24],
    'Income': [30000, 45000, "NaN" ,60000, 35000, 50000, 70000, "NaN", 25000, 80000,
               55000, 40000, 65000, "NaN", 32000, 72000, 48000, 58000, 36000, "NaN",
               29000, 67000, 43000, 78000, 31000, "NaN", 62000, 27000, 53000, 74000,
               39000, "NaN" ,46000, 69000, 34000, 57000, 28000, "NaN", 41000, 76000,
               33000, 59000, 47000, "NaN", 37000, 68000, 52000, 30000, 71000, "NaN"],
    'Spending_Score': [45, 60, 70, "NaN", 55, 65, 80, 40, "NaN", 90,
                       50, 75, "NaN", 85, 60, 70, 45, "NaN", 55, 95,
                       65, 75, 50, "NaN", 85, 60, 70, "NaN", 80, 90,
                       45, 55, 65, "NaN", 75, 85, 50, 95, 60, "NaN",
                       70, 80, 45, 55, 65, "NaN", 75, 85, 90, 50],
    'Purchased': [0, 1, 1, 0, 1, 0, 1, 0, 1, 1,
                  0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
                  0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
                  0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
                  0, 1, 0, 1, 0, 1, 0, 1, 0, 1],
}
# Tạo DataFrame
df = pd.DataFrame(data)
df.replace("NaN", np.nan, inplace=True)
```

```python
#code here

df.to_csv('customer_data.csv', index=False)
```

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   ID              50 non-null     int64  
 1   Age             41 non-null     float64
 2   Income          41 non-null     float64
 3   Spending_Score  41 non-null     float64
 4   Purchased       50 non-null     int64  
dtypes: float64(3), int64(2)
memory usage: 2.1 KB
```

```
df.isnull().sum()
```

```
ID               0
Age              9
Income           9
Spending_Score   9
Purchased        0
dtype: int64
```

```
df.describe()
```

	ID	Age	Income	Spending_Score	Purchased
count	50.000000	41.000000	41.000000	41.000000	50.000000
mean	24.120000	35.609756	50048.780488	67.317073	0.520000
std	14.968866	8.622291	16613.174320	15.696240	0.504672
min	1.000000	22.000000	25000.000000	40.000000	0.000000
25%	13.000000	28.000000	35000.000000	55.000000	0.000000
50%	19.000000	35.000000	48000.000000	65.000000	1.000000
75%	37.750000	43.000000	65000.000000	80.000000	1.000000
max	50.000000	50.000000	80000.000000	95.000000	1.000000

```
df['Purchased'].value_counts()
```

```
Purchased
1    26
0    24
Name: count, dtype: int64
```

6. Xóa dòng có giá trị thiếu

6. Xóa dòng có giá trị thiếu.
7. Điền NaN trong `Age` bằng trung bình.
8. Điền NaN trong `Income` bằng trung vị.
9. Điền NaN trong `Spending_Score` bằng mode.
10. Điền NaN bằng 0.

```
#code here
drop_nan=df.dropna()
print(drop_nan)
```

	ID	Age	Income	Spending_Score	Purchased
0	1	25.0	30000.0	45.0	0
4	5	28.0	35000.0	55.0	1
6	7	35.0	70000.0	80.0	1
9	10	40.0	80000.0	90.0	1
10	11	33.0	55000.0	50.0	0
14	15	29.0	32000.0	60.0	0
15	16	38.0	72000.0	70.0	1
18	19	31.0	36000.0	55.0	0
20	21	24.0	29000.0	65.0	0
21	22	37.0	67000.0	75.0	1
24	15	26.0	31000.0	85.0	0
26	17	41.0	62000.0	70.0	0
29	30	47.0	74000.0	90.0	1
30	31	32.0	39000.0	45.0	0
32	33	28.0	46000.0	65.0	0
35	36	36.0	57000.0	85.0	1
36	37	25.0	28000.0	50.0	0
38	39	31.0	41000.0	60.0	0
41	42	29.0	59000.0	80.0	1
42	43	35.0	47000.0	45.0	0
44	45	27.0	37000.0	65.0	0
46	47	33.0	52000.0	75.0	0
48	49	46.0	71000.0	90.0	0

```
df['Age'].fillna(df['Age'].mean(), inplace= True)
```

```
C:\Users\huykg\AppData\Local\Temp\ipykernel_22712\1294759745.py:1: FutureWarning: A value is trying to be set on a copy of
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col]


  df['Age'].fillna(df['Age'].mean(), inplace= True)
```

```
df['Income'].fillna(df['Income'].median(),inplace= True)
```

```
C:\Users\huykg\AppData\Local\Temp\ipykernel_22712\4043148233.py:1: FutureWarning: A value is trying to be set on a copy of
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col]


  df['Income'].fillna(df['Income'].median(),inplace= True)
```

```
df['Spending_Score'].fillna(df['Spending_Score'].mode(), inplace= True)
```

```
C:\Users\huykg\AppData\Local\Temp\ipykernel_22712\169162806.py:1: FutureWarning: A value is trying
The behavior will change in pandas 3.0. This inplace method will never work because the intermediat

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, i


  df['Spending_Score'].fillna(df['Spending_Score'].mode(), inplace= True)
```

```
df.replace(np.nan,0,inplace= True)
```

```python
#code here
from sklearn.preprocessing import MinMaxScaler, StandardScaler
df['Income'] = MinMaxScaler().fit_transform(df[['Income']])
df['Spending_Score'] = MinMaxScaler().fit_transform(df[['Spending_Score']])
print(df)
```

Chat Ctrl+L Edit Ctrl+I

	ID	Age	Income	Spending_Score	Purchased
0	1	25.000000	0.090909	0.473684	0
1	2	35.609756	0.363636	0.631579	1
2	3	30.000000	0.418182	0.736842	1
3	4	45.000000	0.636364	0.631579	0
4	5	28.000000	0.181818	0.578947	1
5	6	35.609756	0.454545	0.684211	0
6	7	35.000000	0.818182	0.842105	1
7	8	50.000000	0.418182	0.421053	0
8	9	22.000000	0.000000	0.000000	1
9	10	40.000000	1.000000	0.947368	1
10	11	33.000000	0.545455	0.526316	0
11	12	35.609756	0.272727	0.789474	1
12	13	27.000000	0.727273	0.000000	0
13	14	48.000000	0.418182	0.894737	1
14	15	29.000000	0.127273	0.631579	0
15	16	38.000000	0.854545	0.736842	1
16	17	35.609756	0.418182	0.473684	0
17	18	42.000000	0.600000	0.000000	1
18	19	31.000000	0.200000	0.578947	0
19	20	46.000000	0.418182	1.000000	1
20	21	24.000000	0.072727	0.684211	0
21	22	37.000000	0.763636	0.789474	1
22	13	35.609756	0.327273	0.526316	0
23	14	49.000000	0.963636	0.000000	1
...					
46	47	33.000000	0.490909	0.789474	0
47	48	35.609756	0.090909	0.894737	1
48	49	46.000000	0.836364	0.947368	0
49	50	24.000000	0.418182	0.526316	1

```
df['Income'] = StandardScaler().fit_transform(df[['Income']])
df['Age'] = StandardScaler().fit_transform(df[['Age']])
print(df)
```

```
    ID       Age    Income  Spending_Score  Purchased
0    1 -1.375744 -1.322571        0.473684          0
1    2  0.000000 -0.314514        0.631579          1
2    3 -0.727405 -0.112902        0.736842          1
3    4  1.217613  0.693543        0.631579          0
4    5 -0.986741 -0.986552        0.578947          1
5    6  0.000000  0.021505        0.684211          0
6    7 -0.079066  1.365582        0.842105          1
7    8  1.865952 -0.112902        0.421053          0
8    9 -1.764748 -1.658590        0.000000          1
9   10  0.569274  2.037620        0.947368          1
10  11 -0.338401  0.357524        0.526316          0
11  12  0.000000 -0.650533        0.789474          1
12  13 -1.116409  1.029562        0.000000          0
13  14  1.606616 -0.112902        0.894737          1
14  15 -0.857073 -1.188163        0.631579          0
15  16  0.309938  1.499989        0.736842          1
16  17  0.000000 -0.112902        0.473684          0
17  18  0.828609  0.559136        0.000000          1
18  19 -0.597737 -0.919348        0.578947          0
19  20  1.347281 -0.112902        1.000000          1
20  21 -1.505412 -1.389775        0.684211          0
21  22  0.180270  1.163970        0.789474          1
22  13  0.000000 -0.448921        0.526316          0
23  14  1.736284  1.903212        0.000000          1
...
46  47 -0.338401  0.155913        0.789474          0
47  48  0.000000 -1.322571        0.894737          1
48  49  1.347281  1.432785        0.947368          0
49  50 -1.505412 -0.112902        0.526316          1
```

```
df['Age_per_Income'] = df['Age']/ df['Income']
print(df)
```

```
    ID       Age    Income  Spending_Score  Purchased  Age_per_Income
0    1 -1.375744 -1.322571        0.473684          0        1.040204
1    2  0.000000 -0.314514        0.631579          1       -0.000000
2    3 -0.727405 -0.112902        0.736842          1        6.442777
3    4  1.217613  0.693543        0.631579          0        1.755640
4    5 -0.986741 -0.986552        0.578947          1        1.000191
5    6  0.000000  0.021505        0.684211          0        0.000000
6    7 -0.079066  1.365582        0.842105          1       -0.057899
7    8  1.865952 -0.112902        0.421053          0      -16.527123
8    9 -1.764748 -1.658590        0.000000          1        1.064005
9   10  0.569274  2.037620        0.947368          1        0.279382
10  11 -0.338401  0.357524        0.526316          0       -0.946513
11  12  0.000000 -0.650533        0.789474          1       -0.000000
12  13 -1.116409  1.029562        0.000000          0       -1.084352
13  14  1.606616 -0.112902        0.894737          1      -14.230133
14  15 -0.857073 -1.188163        0.631579          0        0.721343
15  16  0.309938  1.499989        0.736842          1        0.206627
16  17  0.000000 -0.112902        0.473684          0       -0.000000
17  18  0.828609  0.559136        0.000000          1        1.481946
18  19 -0.597737 -0.919348        0.578947          0        0.650175
19  20  1.347281 -0.112902        1.000000          1      -11.933143
20  21 -1.505412 -1.389775        0.684211          0        1.083206
21  22  0.180270  1.163970        0.789474          1        0.154875
22  13  0.000000 -0.448921        0.526316          0       -0.000000
23  14  1.736284  1.903212        0.000000          1        0.912292
...
46  47 -0.338401  0.155913        0.789474          0       -2.170453
47  48  0.000000 -1.322571        0.894737          1       -0.000000
48  49  1.347281  1.432785        0.947368          0        0.940323
49  50 -1.505412 -0.112902        0.526316          1       13.333747
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
df[df['Income'] <= 100000]
```

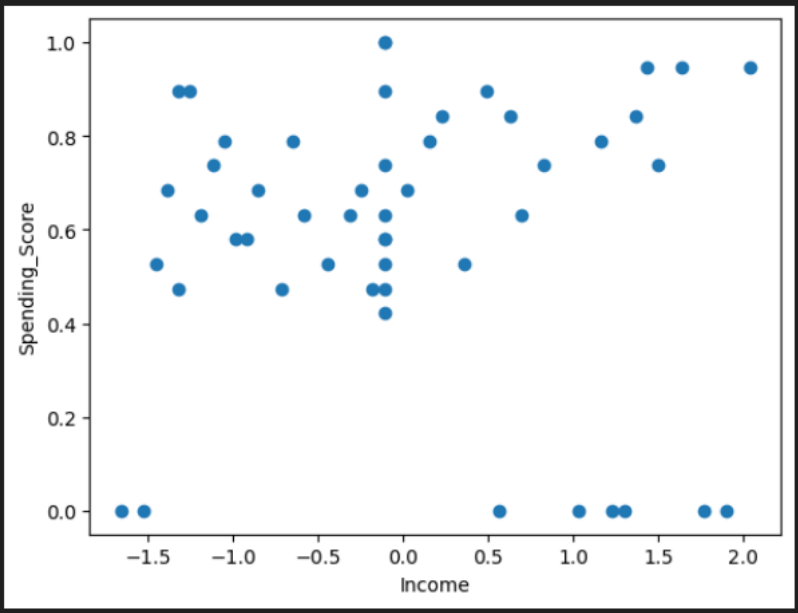
	ID	Age	Income	Spending_Score	Purchased	Age_per_Income
0	1	-1.375744	-1.322571	0.473684	0	1.040204
1	2	0.000000	-0.314514	0.631579	1	-0.000000
2	3	-0.727405	-0.112902	0.736842	1	6.442777
3	4	1.217613	0.693543	0.631579	0	1.755640
4	5	-0.986741	-0.986552	0.578947	1	1.000191
5	6	0.000000	0.021505	0.684211	0	0.000000
6	7	-0.079066	1.365582	0.842105	1	-0.057899
7	8	1.865952	-0.112902	0.421053	0	-16.527123
8	9	-1.764748	-1.658590	0.000000	1	1.064005
9	10	0.569274	2.037620	0.947368	1	0.279382
10	11	-0.338401	0.357524	0.526316	0	-0.946513
11	12	0.000000	-0.650533	0.789474	1	-0.000000
12	13	-1.116409	1.029562	0.000000	0	-1.084352
13	14	1.606616	-0.112902	0.894737	1	-14.230133
14	15	-0.857073	-1.188163	0.631579	0	0.721343
15	16	0.309938	1.499989	0.736842	1	0.206627
16	17	0.000000	-0.112902	0.473684	0	-0.000000
17	18	0.828609	0.559136	0.000000	1	1.481946
18	19	-0.597737	-0.919348	0.578947	0	0.650175
19	20	1.347281	-0.112902	1.000000	1	-11.933143
20	21	-1.505412	-1.389775	0.684211	0	1.083206
21	22	0.180270	1.163970	0.789474	1	0.154875
22	13	0.000000	-0.448921	0.526316	0	-0.000000
23	14	1.736284	1.903212	0.000000	1	0.912292
24	15	-1.246076	-1.255367	0.894737	0	0.992599

```
print(df.corr())
```

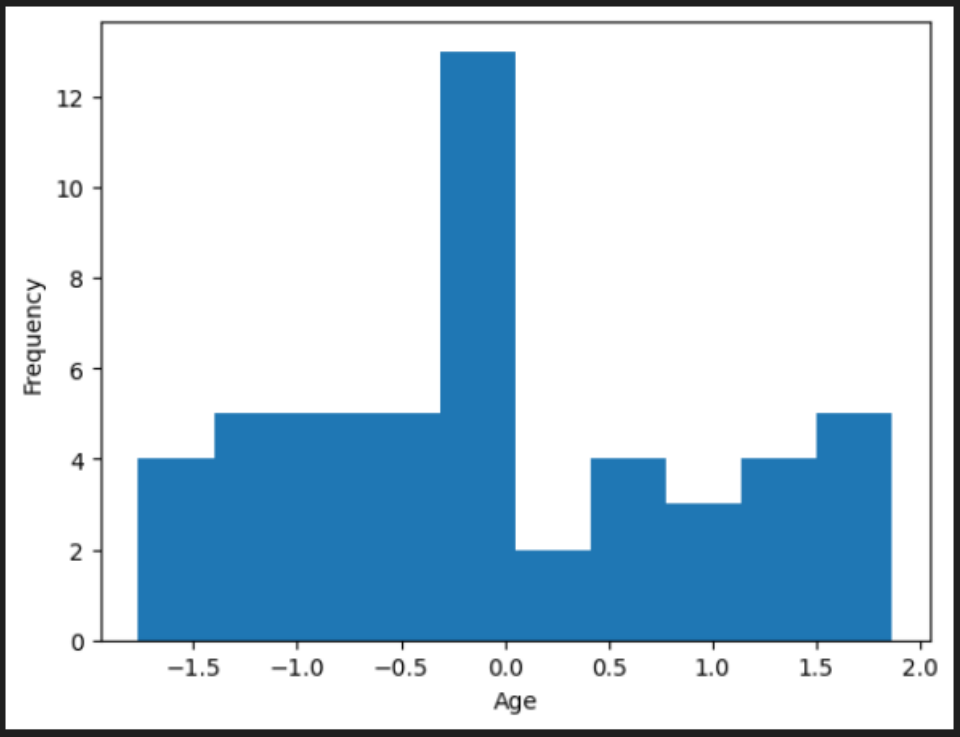
```
                    ID       Age        Income  Spending_Score  Purchased  \
ID              1.000000  0.069816  3.074109e-02        0.076233  -0.003026
Age             0.069816  1.000000  5.972058e-01        0.119755   0.274989
Income          0.030741  0.597206  1.000000e+00       -0.059611   0.329078
Spending_Score  0.076233  0.119755 -5.961117e-02        1.000000  -0.079625
Purchased      -0.003026  0.274989  3.290782e-01       -0.079625   1.000000
Age_per_Income  0.017675 -0.593772  2.305895e-17       -0.191493  -0.099375

                Age_per_Income
ID                1.767542e-02
Age              -5.937722e-01
Income            2.305895e-17
Spending_Score   -1.914934e-01
Purchased        -9.937546e-02
Age_per_Income    1.000000e+00
```

```
import matplotlib.pyplot as plt
plt.scatter(df['Income'], df['Spending_Score'])
plt.xlabel('Income')
plt.ylabel('Spending_Score')
plt.show()
```



```
plt.hist(df['Age'], bins=10)
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```



```
plt.boxplot(df['Income'])
plt.xlabel('Income')
plt.show()
```

Ứng dụng ANN để dự đoán

Chúng ta sẽ xây dựng một mạng nơ-ron nhân tạo để dự đoán `Purchased` dựa trên `Age`, `Income`, và `Spending_Score`.

```
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# 1. Chuẩn bị dữ liệu
# Tải dữ liệu từ file CSV (giả sử đã xử lý NaN)
data = pd.read_csv('customer_data_fix.csv', index_col= 0)


# Tách đặc trưng (X) và nhãn (y)
X = data.drop(columns=['ID', 'Purchased'], errors='ignore')
y = data['Purchased']

# Chuẩn hóa dữ liệu
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Chia tập train/test (80% and 20%)
X_train, X_test, y_train, y_test = train_test_split(X , y, test_size= 0.2, random_state= 42, stratify=y)

# Chuyển dữ liệu sang tensor của PyTorch
X_train = torch.tensor(X_train, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_train = torch.tensor(y_train.values, dtype=torch.float32).view(-1, 1)  # Reshape để phù hợp với đầu ra
y_test = torch.tensor(y_test.values, dtype=torch.float32).view(-1, 1)    # Reshape để phù hợp với đầu ra
```

✓ 0.0s

```
print(X)
```

✓ 0.0s

```
[[-1.24387609 -1.21618242 -1.4973208 ]
 [-0.76969128 -0.91213681 -0.82861442]
 [ 0.33673994  1.21618242  0.84315152]
 [ 1.12704795  1.82427363  1.51185789]
 [ 0.02061673  0.3040456  -1.16296761]
 [-0.61162968 -1.09456418 -0.49426123]
 [ 0.81092475  1.33780066  0.17444514]
 [-0.29550647 -0.85132769 -0.82861442]
 [-1.40193769 -1.27699154 -0.15990805]
 [ 0.65286314  1.03375506  0.50879833]
 [-1.08581449 -1.1553733   1.1775047 ]
 [ 1.28510955  0.72970945  0.17444514]
 [ 2.23347917  1.4594189   1.51185789]
 [-0.13744487 -0.66890033 -1.4973208 ]
 [-0.76969128 -0.24323648 -0.15990805]
 [ 0.49480154  0.42566385  1.1775047 ]
 [-1.24387609 -1.33780066 -1.16296761]
 [-0.29550647 -0.54728209 -0.49426123]
 [-0.61162968  0.54728209  0.84315152]
 [ 0.33673994 -0.18242736 -1.4973208 ]
 [-0.92775289 -0.79051857 -0.15990805]
 [ 0.02061673  0.12161824  0.50879833]
 [ 2.07541757  1.27699154  1.51185789]]
```

+ Code

```python
# Bước 2: Xây dựng mô hình ANN
# 2. Định nghĩa mô hình ANN
class ANN(nn.Module):
    def __init__(self, input_size):
        super(ANN, self).__init__()
        self.layers = nn.Sequential(
            # Lớp ẩn 1: 16 nơ-ron
            nn.Linear(input_size,16),
            nn.ReLU(),
            # Lớp ẩn 2: 8 nơ-ron
            nn.Linear(16,8),
            nn.ReLU(),
            # Lớp đầu ra: 1 nơ-ron
            nn.Linear(8,1),
            # Hàm kích hoạt sigmoid cho phân loại nhị phân
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.layers(x)
```

✓ 0.0s

```python
# Khởi tạo mô hình
input_size =  3#Số đặc trưng (3)
model = ANN(input_size)
print(model)
```

✓ 0.0s

```
ANN(
```

```python
# 3. Định nghĩa hàm mất mát và tối ưu hóa
criterion =  nn.BCELoss()# Binary Cross Entropy Loss
optimizer = optim.Adam(model.parameters(), lr=0.01)

# Huấn luyện mô hình
num_epochs = 50
for epoch in range(num_epochs):
    model.train()

    # Forward pass
    outputs = model(X_train)
    loss = criterion(outputs,y_train)

    # Backward pass và tối ưu hóa
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    # In loss mỗi 10 epochs
    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')
```

✓ 0.1s

```
Epoch [10/50], Loss: 0.6002
Epoch [20/50], Loss: 0.4315
Epoch [30/50], Loss: 0.4230
Epoch [40/50], Loss: 0.3863
Epoch [50/50], Loss: 0.3563
```

+ Code

```python
# 4. Đánh giá mô hình
model.eval()
with torch.no_grad():
    y_pred =model(X_test)
    y_pred_class = (y_pred >= 0.5).float() # Chuyển thành 0 hoặc 1
    accuracy = (y_pred_class.eq(y_test).sum() / float(y_test.shape[0])).item()
    print(f'Accuracy trên tập kiểm tra: {accuracy * 100:.2f}%')


    # In một số dự đoán mẫu
    print("Dự đoán mẫu (5 dòng đầu tiên):")
    print(y_pred_class[:5].numpy())
    print("Nhãn thực tế:")
    print(y_test[:5].numpy())
```

✓ 0.0s

```
Accuracy trên tập kiểm tra: 80.00%
Dự đoán mẫu (5 dòng đầu tiên):
[[1.]
 [0.]
 [1.]
 [0.]
 [1.]]
Nhãn thực tế:
[[1.]
 [0.]
 [0.]
 [0.]
 [1.]]
```