

# Cardiovascular Disease Dataset

Nhóm 3

# — Thành viên

1

**Nguyễn Minh Đăng**

2

**Nguyễn Thị Thu Trang**

3

**Tô Minh Đức**

4

**Nguyễn Trường Giang**

5

**Nguyễn Văn Thi**

# Mục lục

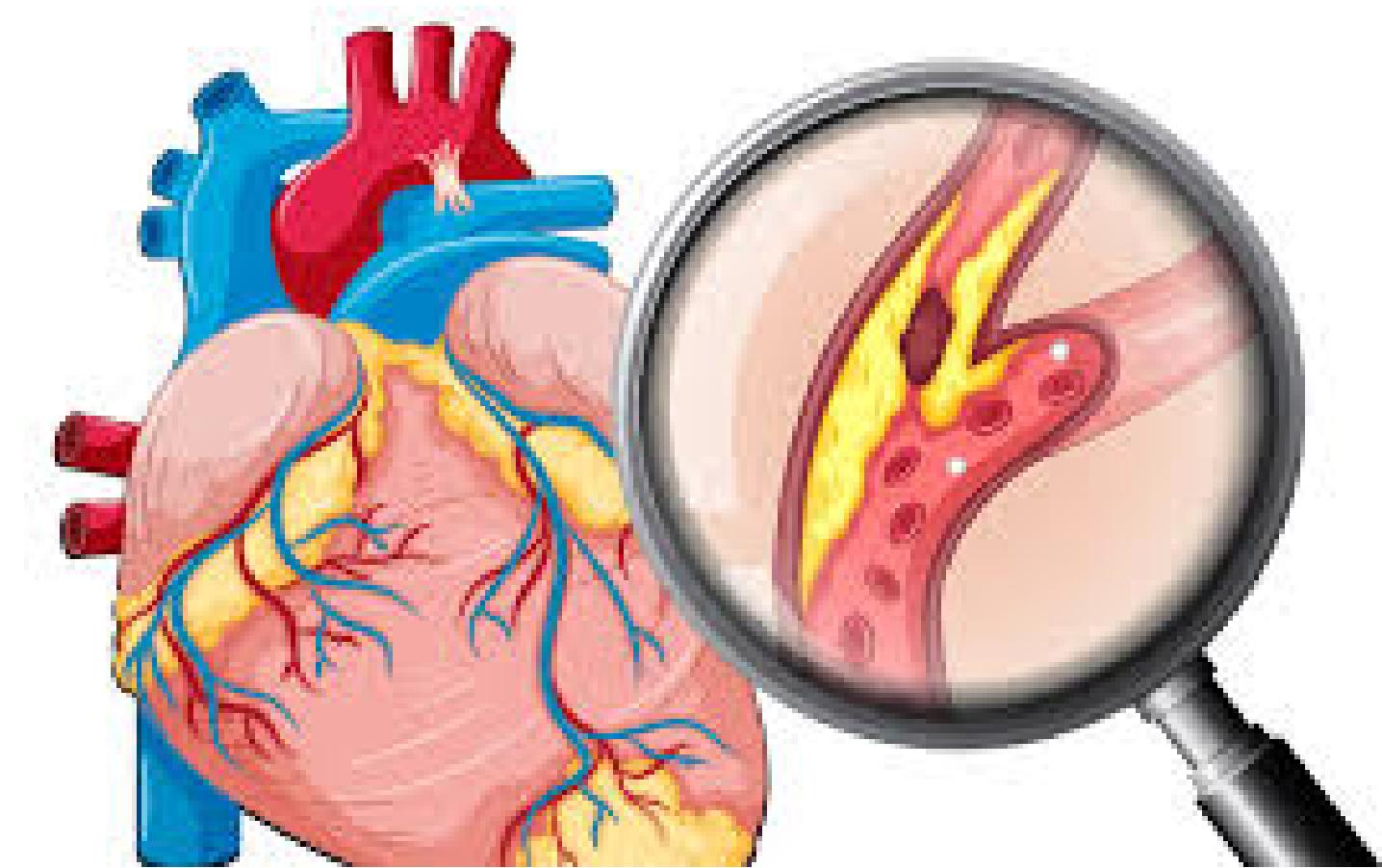
- 1. Phân tích tập dữ liệu
- 2. Tiền xử lý dữ liệu
- 3. Các thuật toán phân loại
- 4. Kết luận

01

# Phân tích tập dữ liệu

# Tổng quan về tập dữ liệu

- Nguồn: Kaggle
- Gồm 13 thuộc tính: tuổi, giới tính, chiều cao, cân nặng, huyết áp, cholesterol, glucose, thói quen sinh hoạt...
- Quy mô: 70.000 bản ghi với thông tin sức khỏe của các bệnh nhân
- Cột mục tiêu là Cardio – biểu thị kết quả của việc bị bệnh (1) và không bị bệnh (0)



# Tổng quan về tập dữ liệu

Dataset gồm có 13 cột

Tên cột	Ý nghĩa	Đơn vị	Kiểu dữ liệu	Dạng dữ liệu
<code>id</code>	Định danh duy nhất cho mỗi bệnh nhân	-	Số nguyên	Danh mục (ID)
<code>age</code>	Tuổi của bệnh nhân (tính theo ngày)	Ngày	Số nguyên	Số (numerical)
<code>gender</code>	Giới tính của bệnh nhân: 1 = Nữ, 2 = Nam	-	Số nguyên	Danh mục (categorical)
<code>height</code>	Chiều cao bệnh nhân	cm	Số nguyên	Số (numerical)
<code>weight</code>	Cân nặng bệnh nhân	kg	Số thực	Số (numerical)
<code>ap_hi</code>	Huyết áp tâm thu	mmHg	Số nguyên	Số (numerical)
<code>ap_lo</code>	Huyết áp tâm trương	mmHg	Số nguyên	Số (numerical)
<code>cholesterol</code>	Mức cholesterol trong máu: 1 = Bình thường, 2 = Cao, 3 = Rất cao	-	Số nguyên	Danh mục thứ bậc (ordinal)
<code>gluc</code>	Mức glucose trong máu: 1 = Bình thường, 2 = Cao, 3 = Rất cao	-	Số nguyên	Danh mục thứ bậc (ordinal)
<code>smoke</code>	Có hút thuốc không: 0 = Không, 1 = Có	-	Số nguyên	Nhi phân (binary)
<code>alco</code>	Có uống rượu không: 0 = Không, 1 = Có	-	Số nguyên	Nhi phân (binary)
<code>active</code>	Có vận động thể chất không: 0 = Không, 1 = Có	-	Số nguyên	Nhi phân (binary)
<code>cardio</code>	<b>Nhãn (label):</b> Bệnh nhân có mắc bệnh tim mạch không: 0 = Không, 1 = Có	-	Số nguyên	Nhi phân (binary)

# Tổng quan về tập dữ liệu

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70000 entries, 0 to 69999
Data columns (total 13 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   id          70000 non-null   int64  
 1   age         70000 non-null   int64  
 2   gender      70000 non-null   int64  
 3   height      70000 non-null   int64  
 4   weight      70000 non-null   float64
 5   ap_hi       70000 non-null   int64  
 6   ap_lo       70000 non-null   int64  
 7   cholesterol 70000 non-null   int64  
 8   gluc        70000 non-null   int64  
 9   smoke       70000 non-null   int64  
 10  alco        70000 non-null   int64  
 11  active      70000 non-null   int64  
 12  cardio      70000 non-null   int64  
dtypes: float64(1), int64(12)
memory usage: 6.9 MB
```

```
df.nunique()
```

```
id               70000
age              8076
gender           2
height            109
weight            287
ap_hi             153
ap_lo             157
cholesterol       3
gluc              3
smoke             2
alco              2
active            2
cardio            2
dtype: int64
```

```
df.duplicated().sum()
```

```
0
```

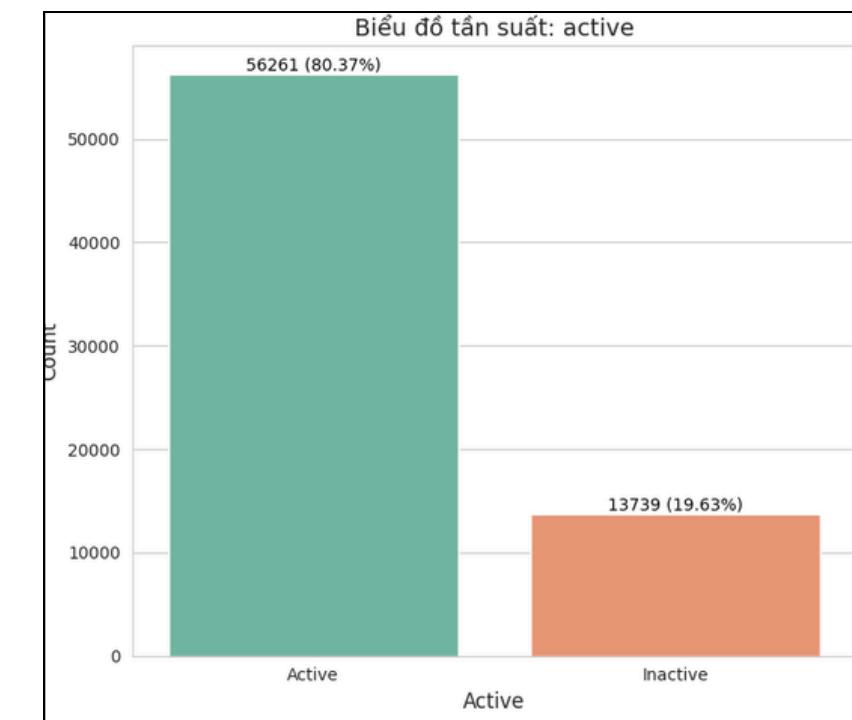
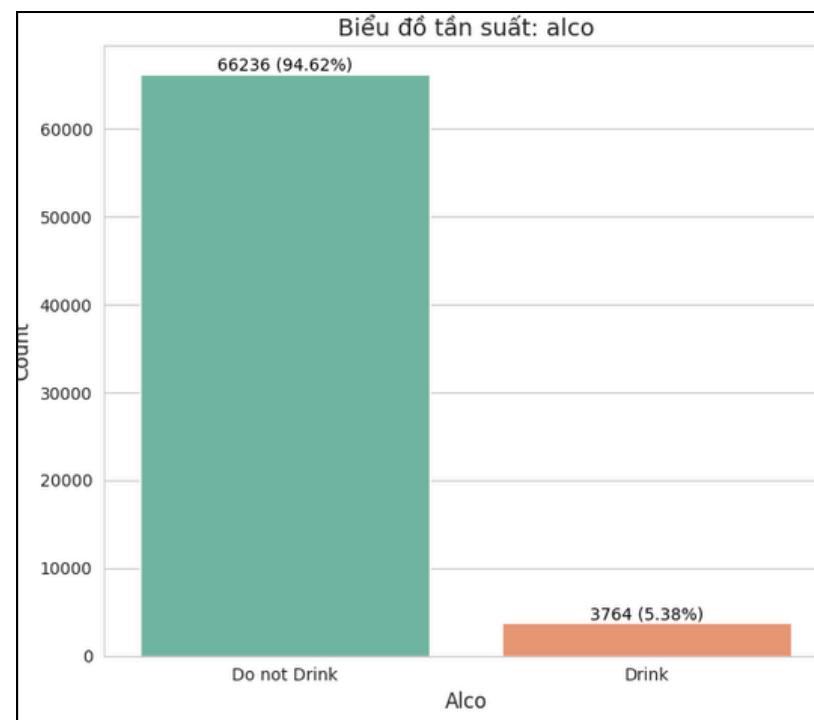
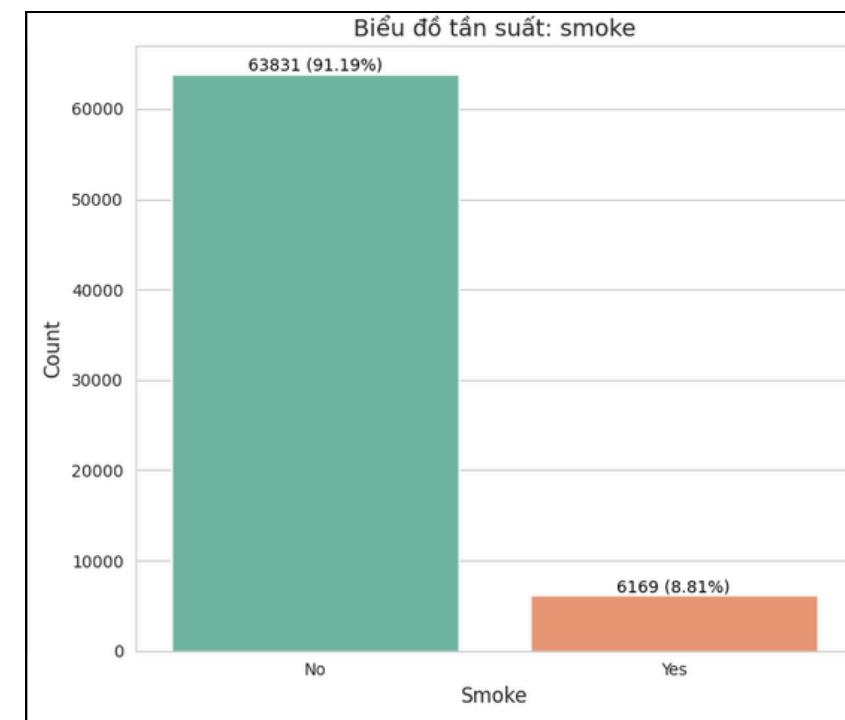
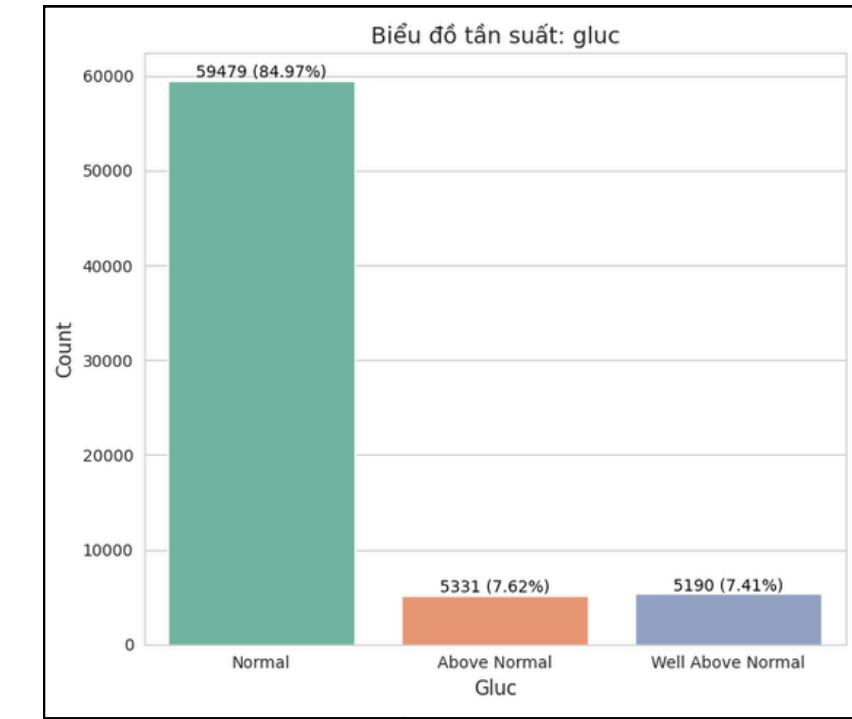
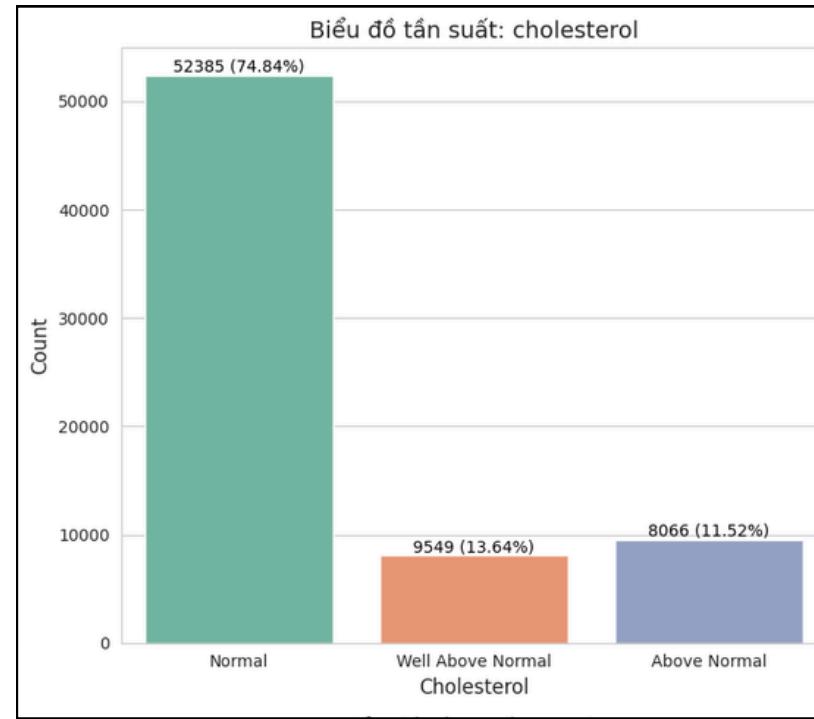
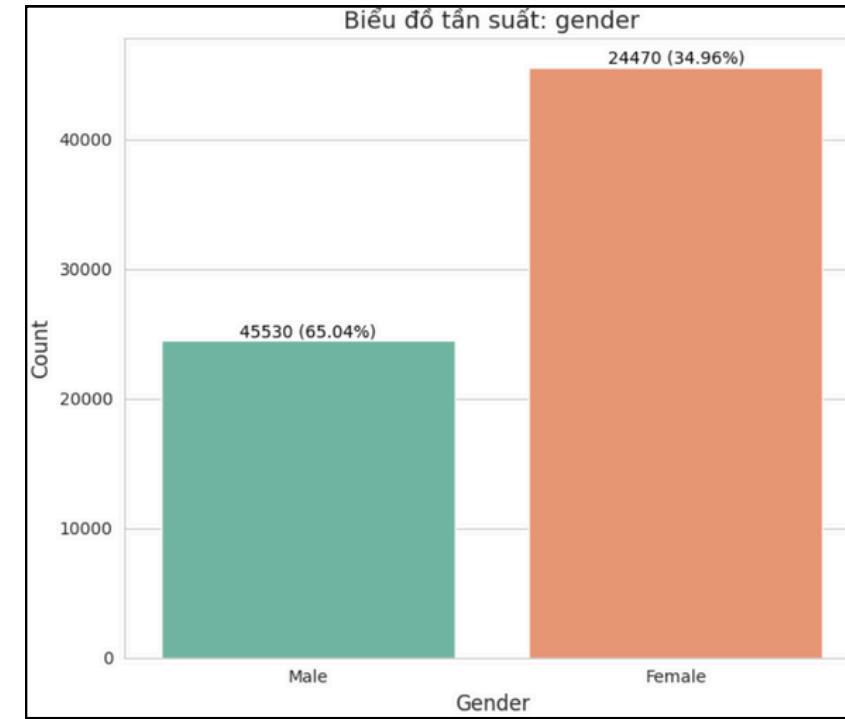
```
df.shape
```

```
(70000, 13)
```

# Thống kê mô tả các biến numerical

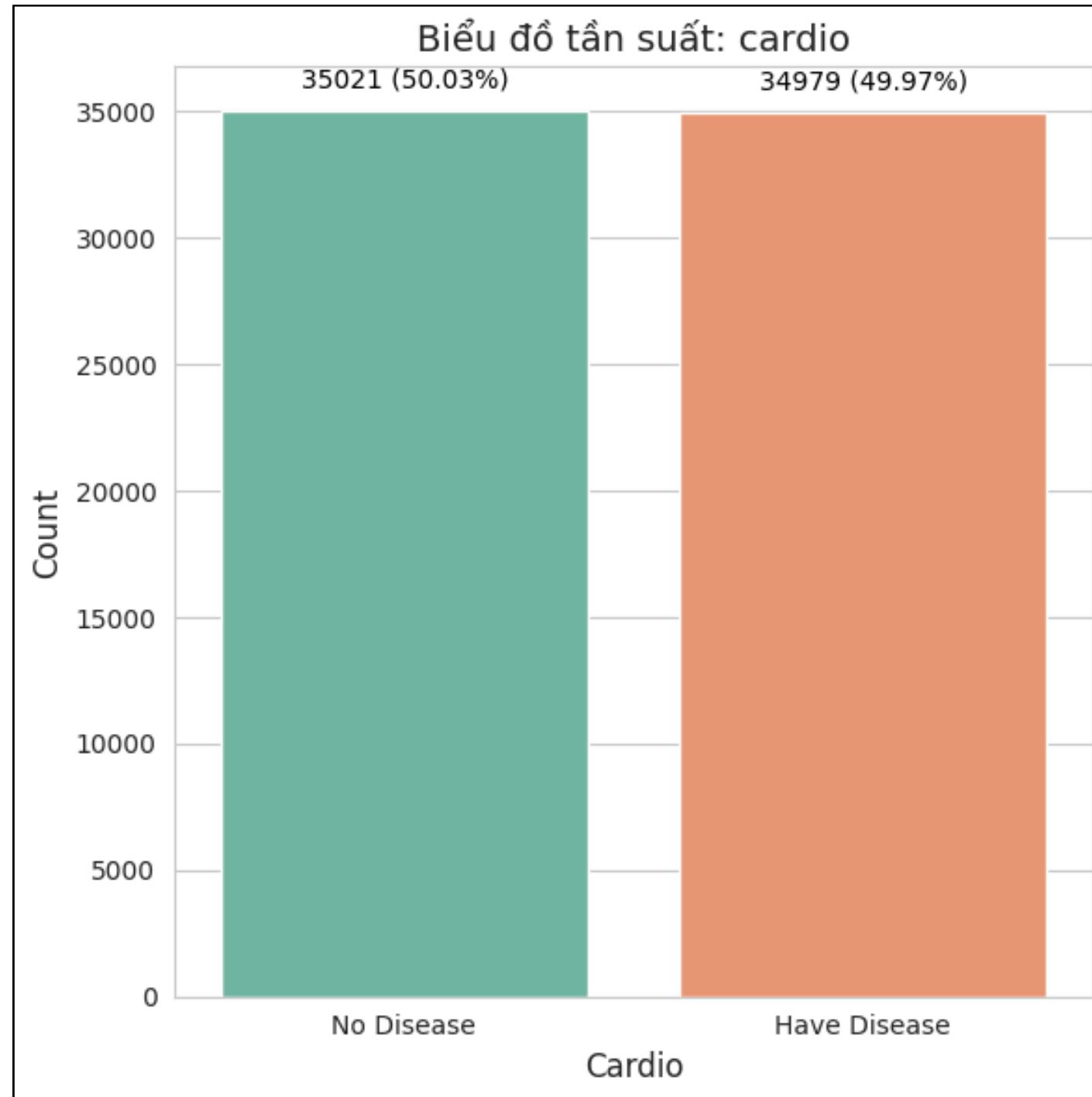
	<b>id</b>	<b>age</b>	<b>height</b>	<b>weight</b>	<b>ap_hi</b>	<b>ap_lo</b>
count	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000
mean	49972.419900	19468.865814	164.359229	74.205690	128.817286	96.630414
std	28851.302323	2467.251667	8.210126	14.395757	154.011419	188.472530
min	0.000000	10798.000000	55.000000	10.000000	-150.000000	-70.000000
25%	25006.750000	17664.000000	159.000000	65.000000	120.000000	80.000000
50%	50001.500000	19703.000000	165.000000	72.000000	120.000000	80.000000
75%	74889.250000	21327.000000	170.000000	82.000000	140.000000	90.000000
max	99999.000000	23713.000000	250.000000	200.000000	16020.000000	11000.000000

# Phân bố các biến categorical



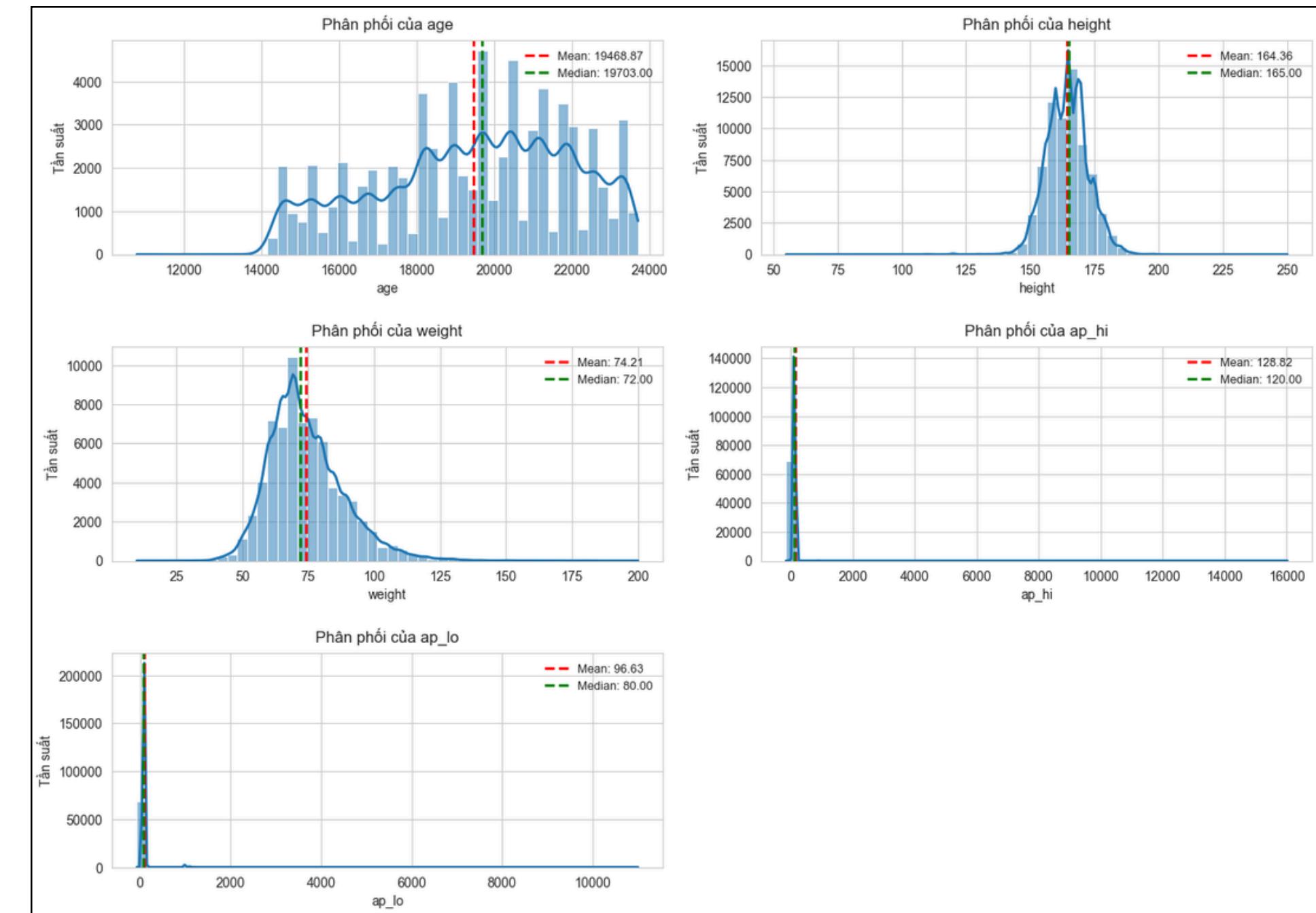
# Phân bố mục tiêu cardio

Cân bằng tốt, tỷ lệ mắc bệnh tim và không mắc bệnh tim xấp xỉ 50%

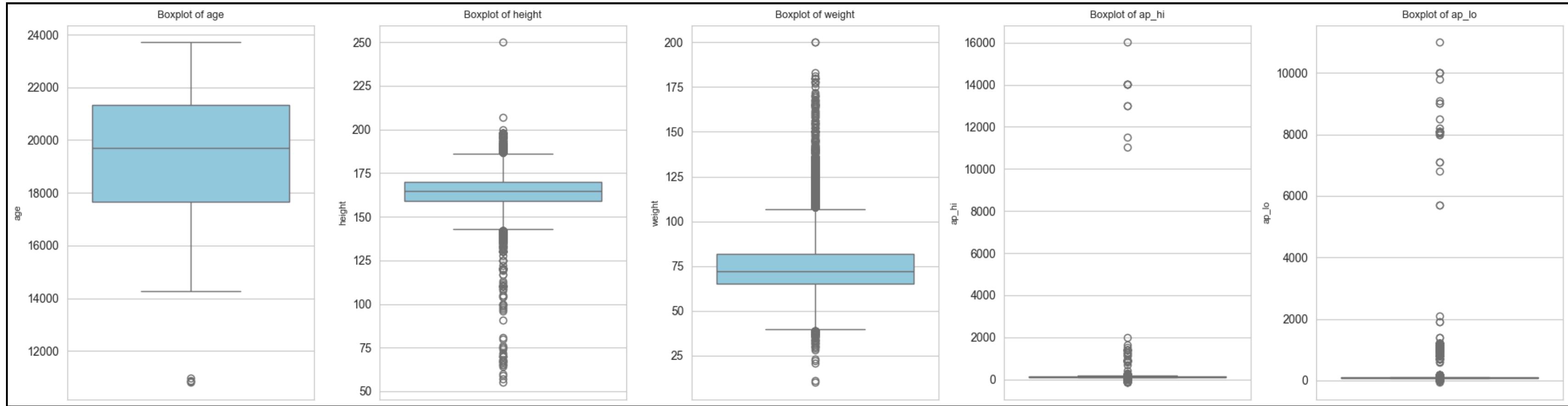


# Phân bố các biến numerical

- **Tuổi:** Trung bình khoảng 49-60 tuổi, phân phối khá cân bằng và chỉ hơi lệch trái nhẹ
- **Chiều cao và cân nặng:** Phân phối hợp lý tuy còn nhiều outliers, chiều cao trung bình 164.36cm, cân nặng trung bình 74.21kg
- **Huyết áp:** Tâm thu/tâm trương có nhiều giá trị bất thường vượt ngưỡng thực tế ( $>1000$  mmHg)



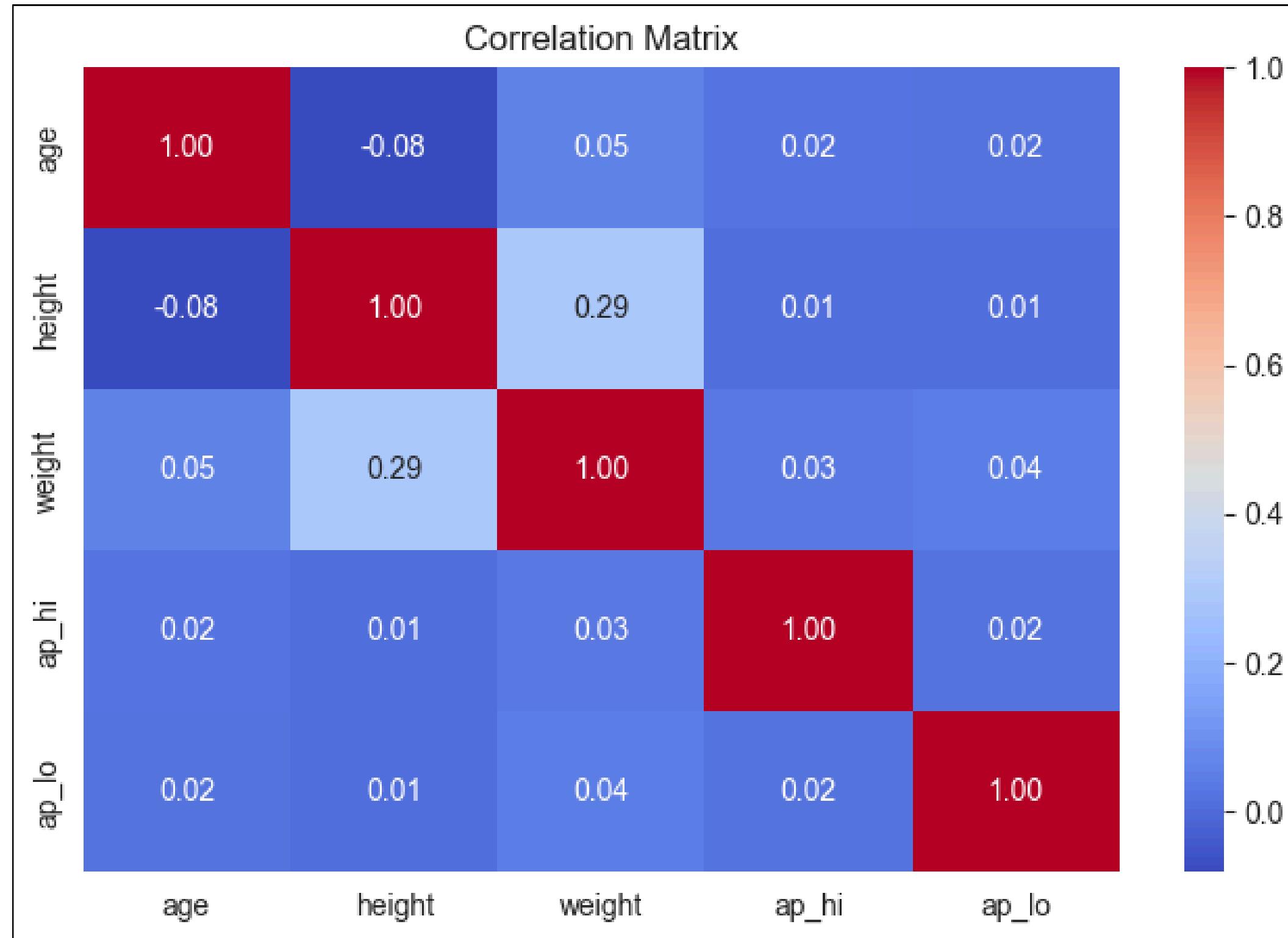
# Outliers



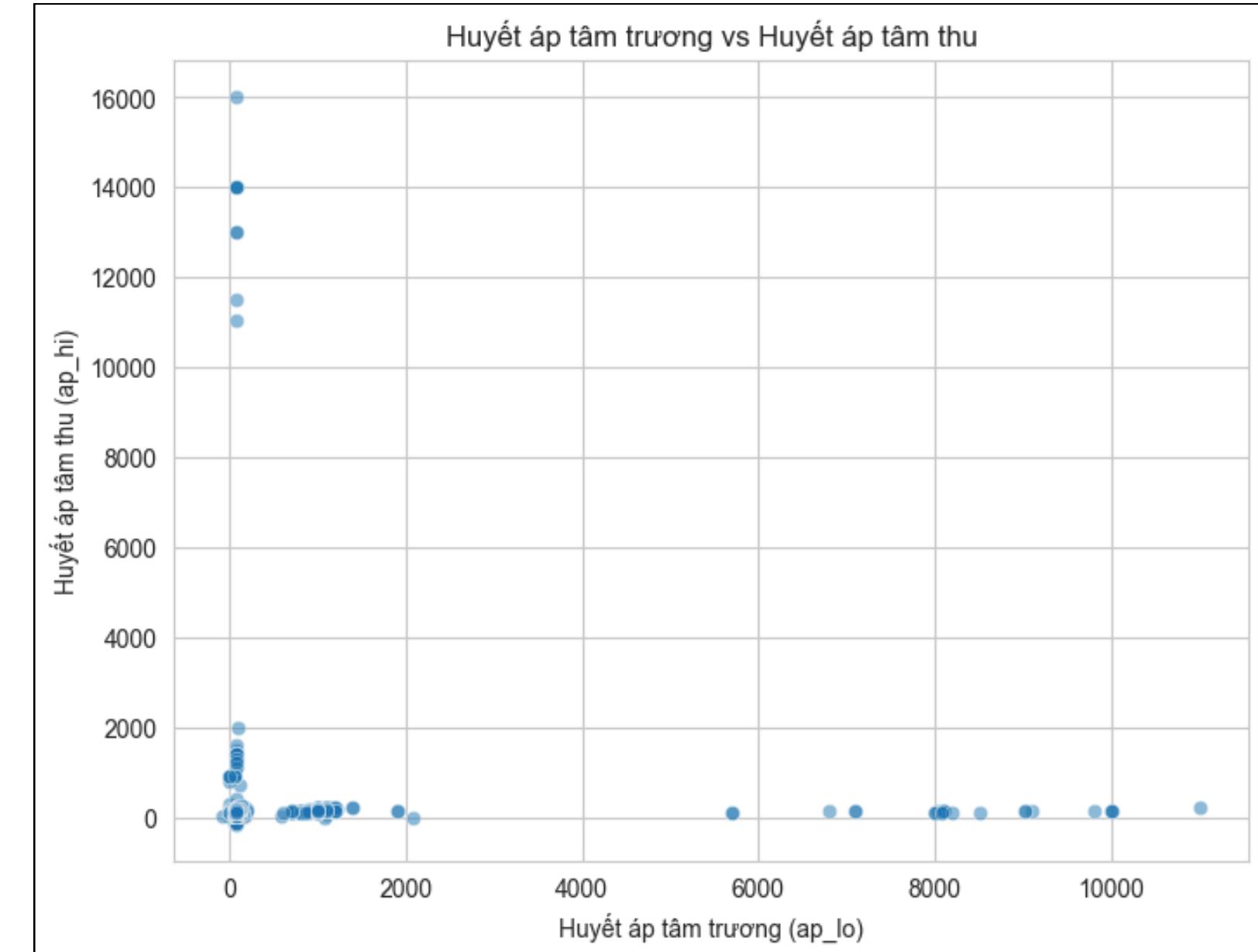
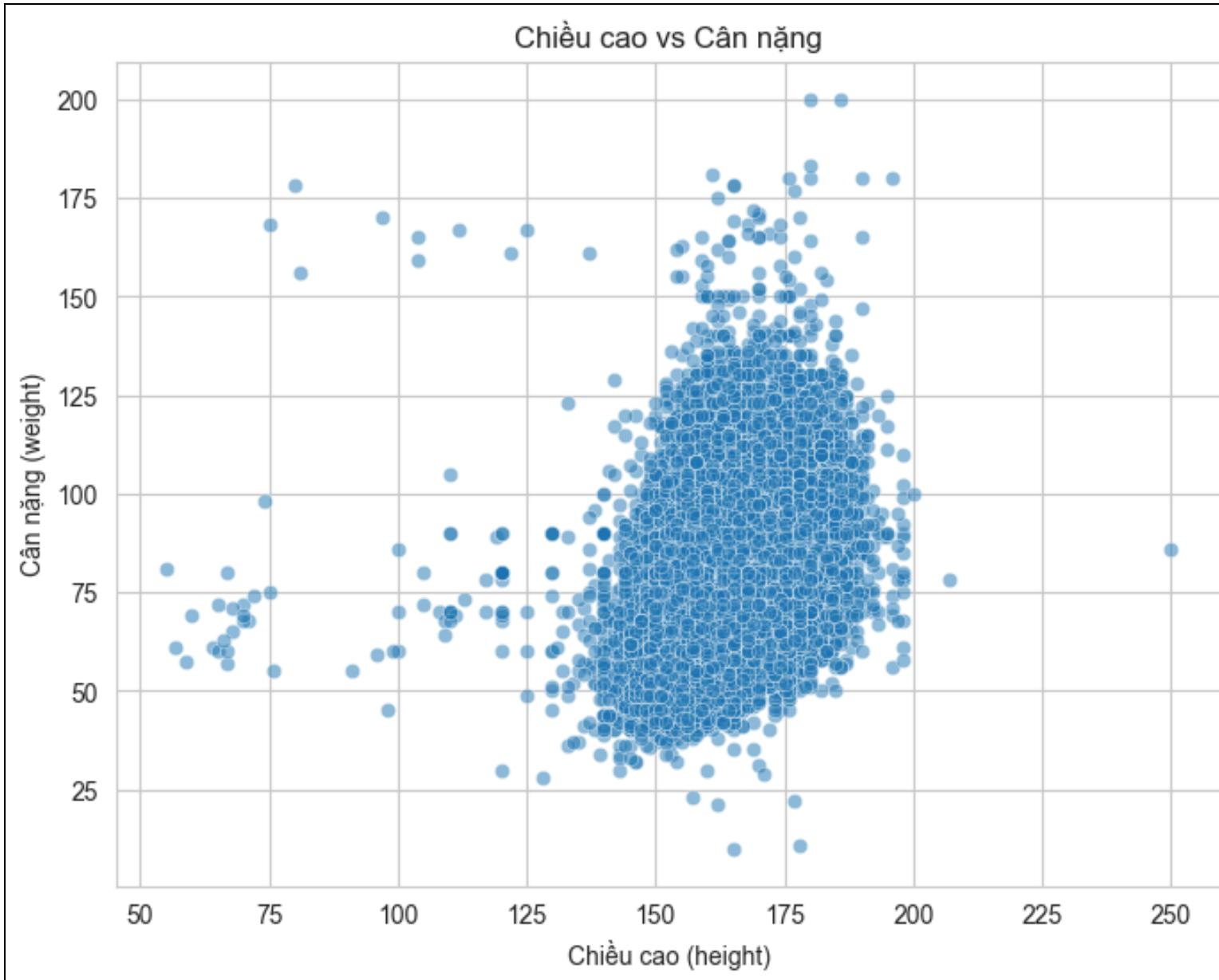
## Phương pháp xử lý:

- Loại bỏ các giá trị ngoại lệ bằng kỹ thuật cắt ngưỡng (Clip)
- Áp dụng kỹ thuật thống kê như IQR (Interquartile Range) để loại bỏ ngoại lệ.

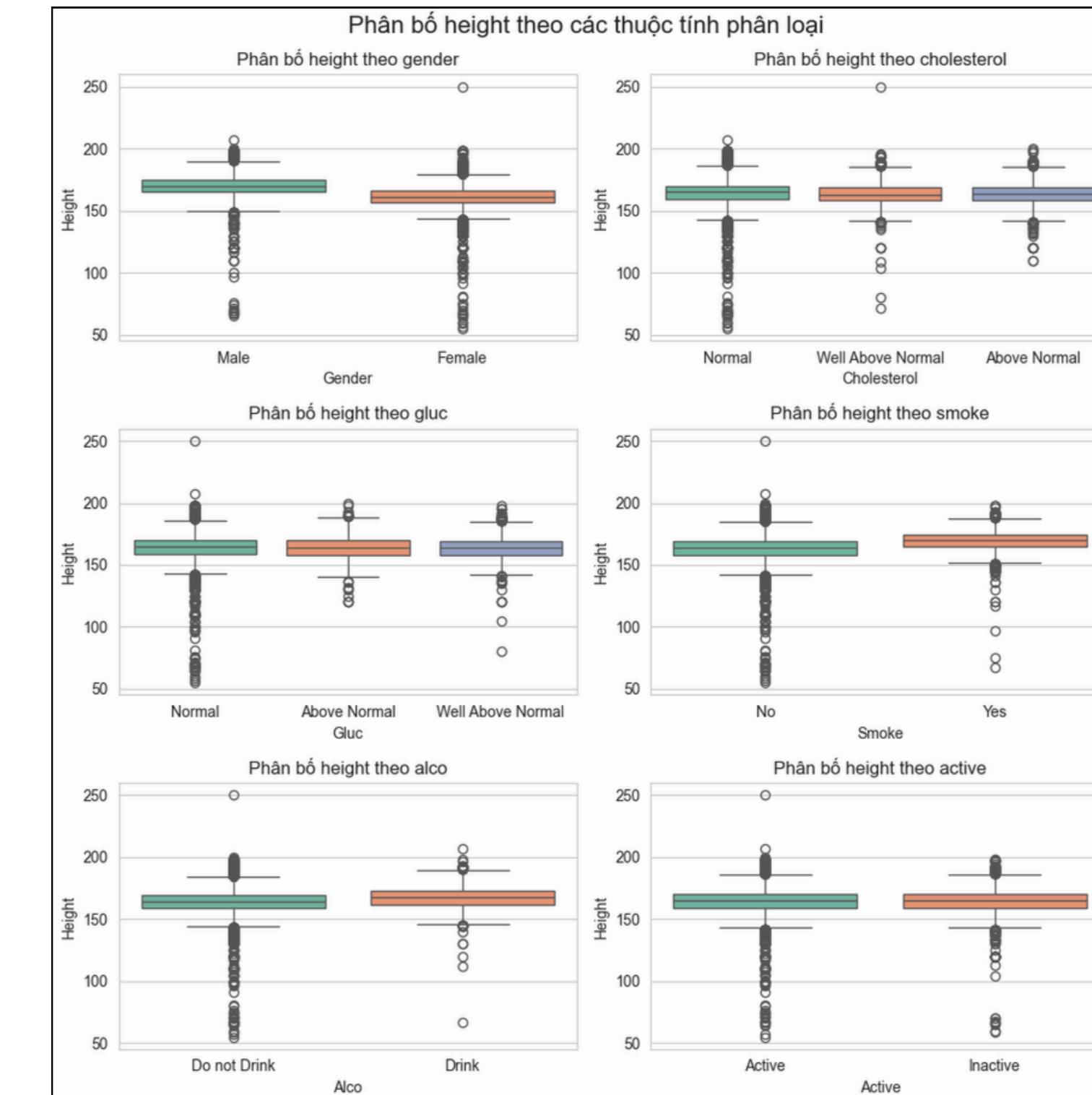
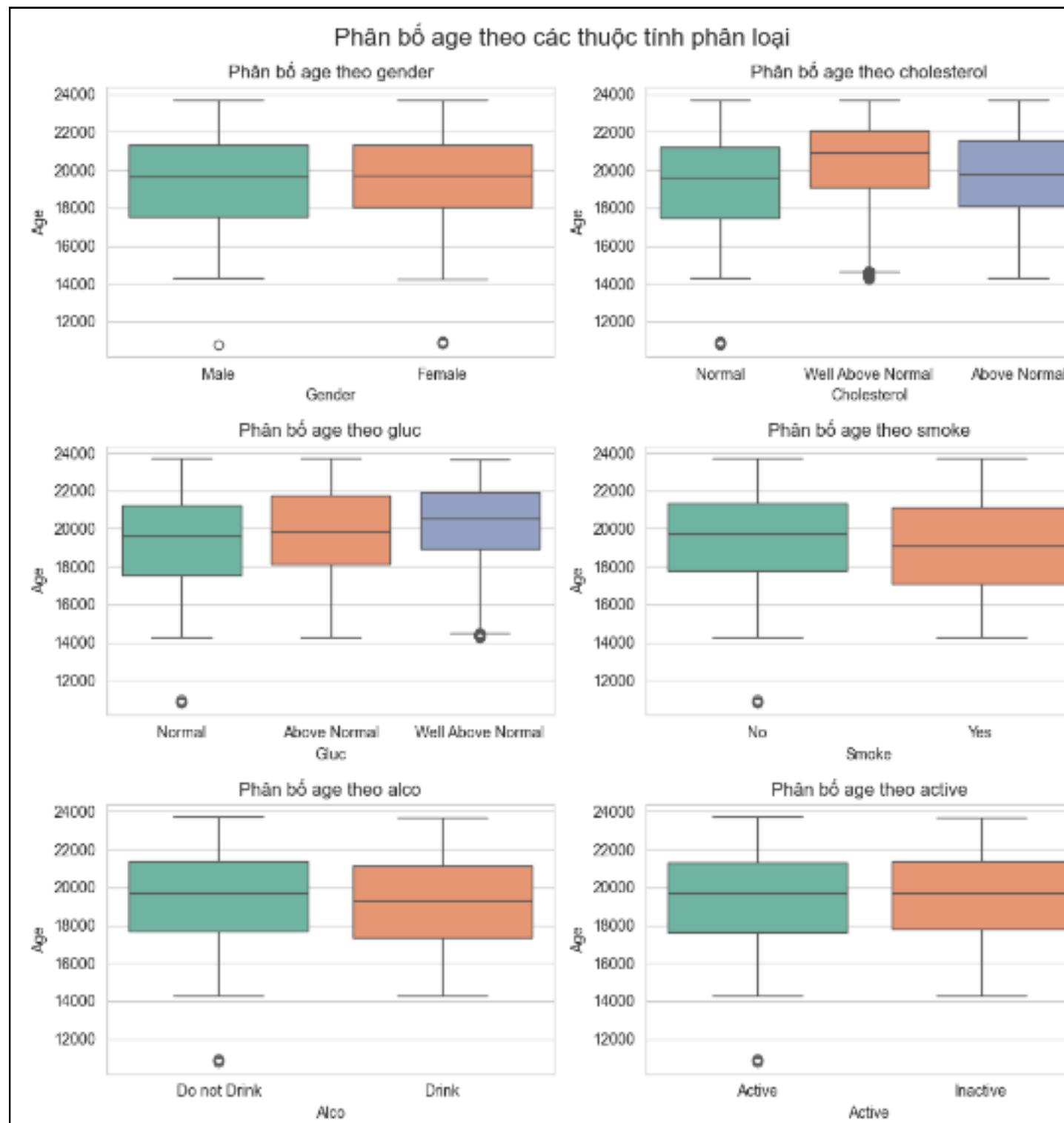
# Phân tích mối quan hệ giữa các biến numerical



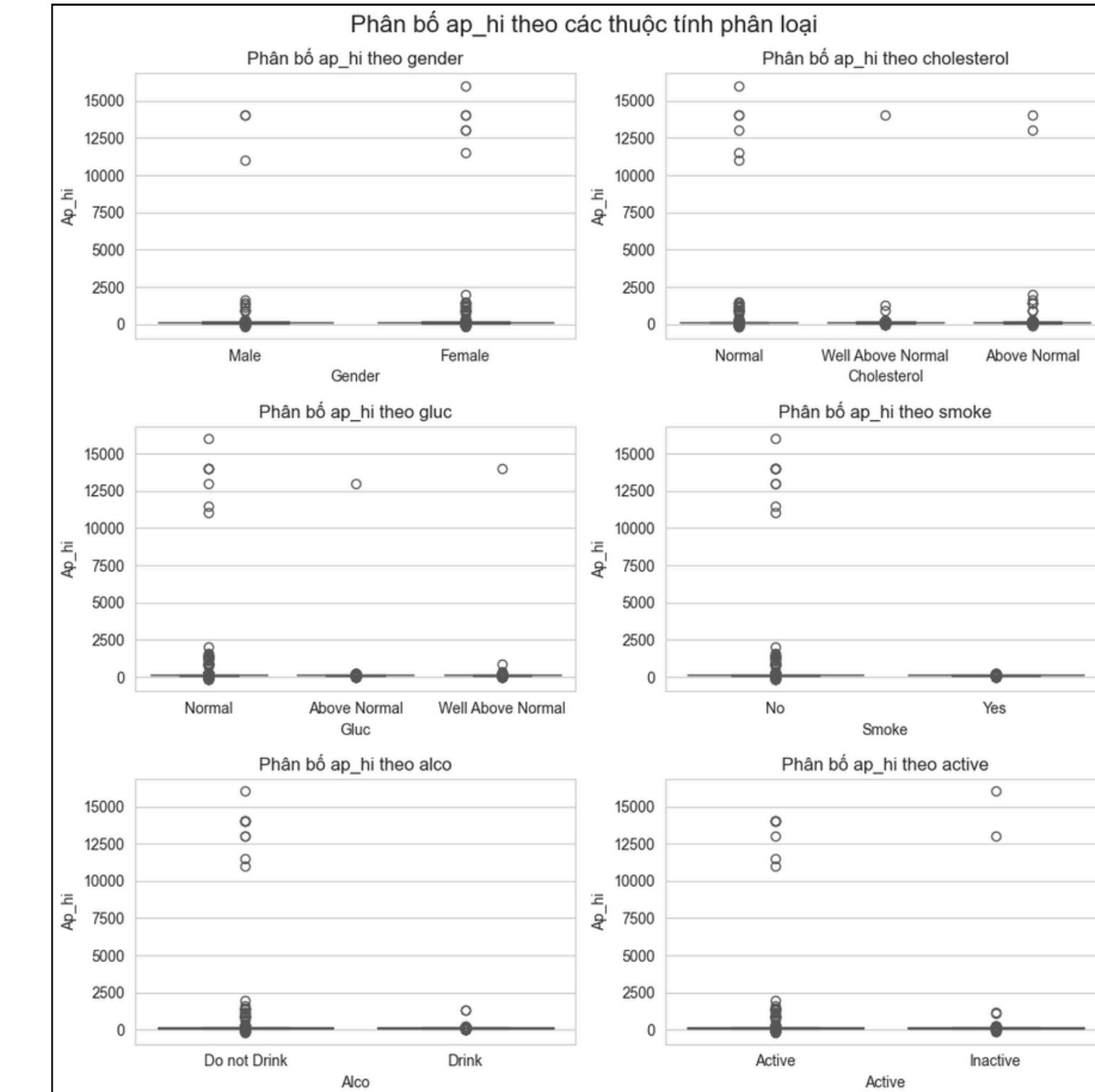
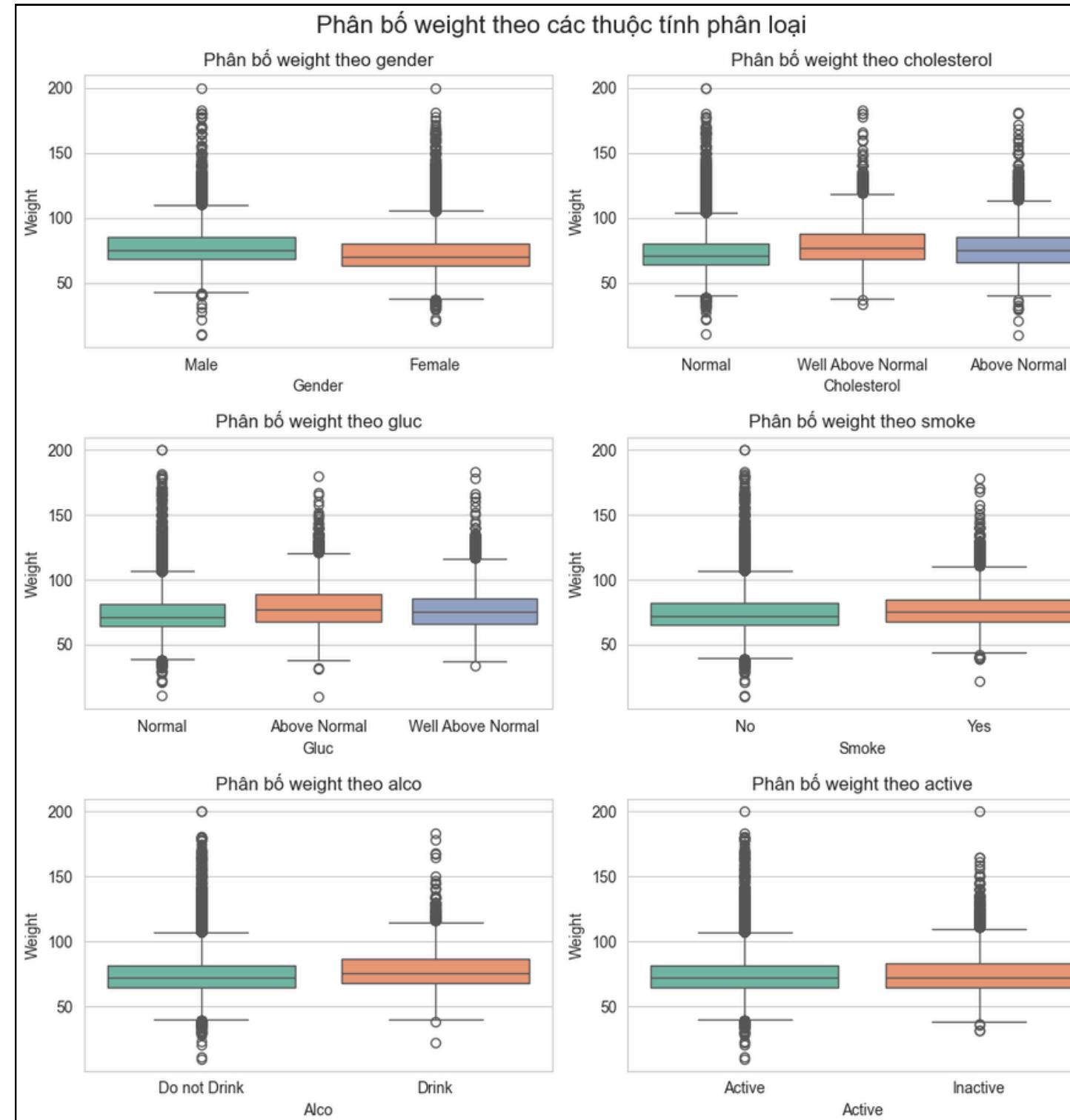
# Phân tích mối quan hệ giữa các biến numerical



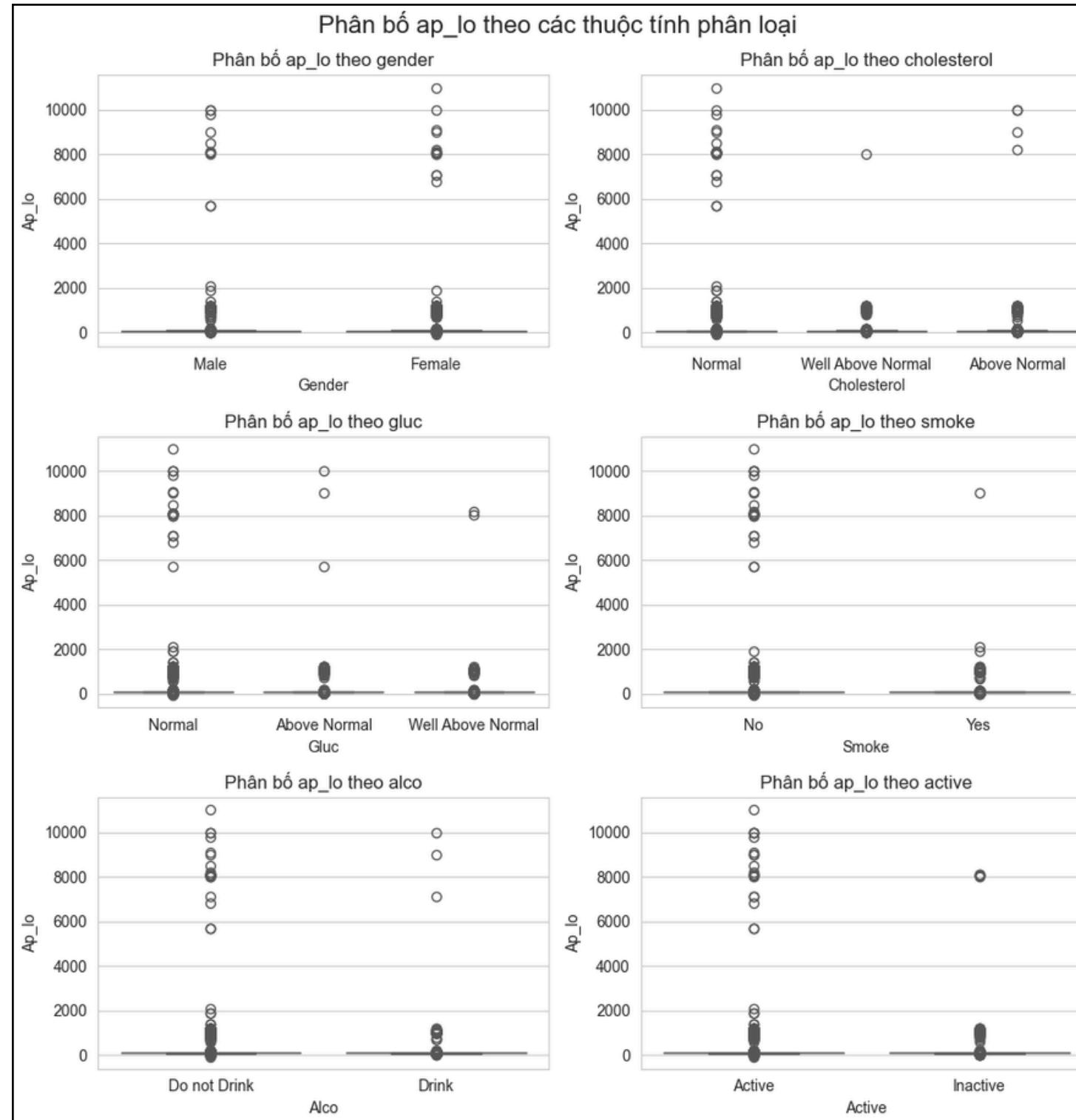
# Phân tích mối quan hệ giữa biến numerical và biến categorical



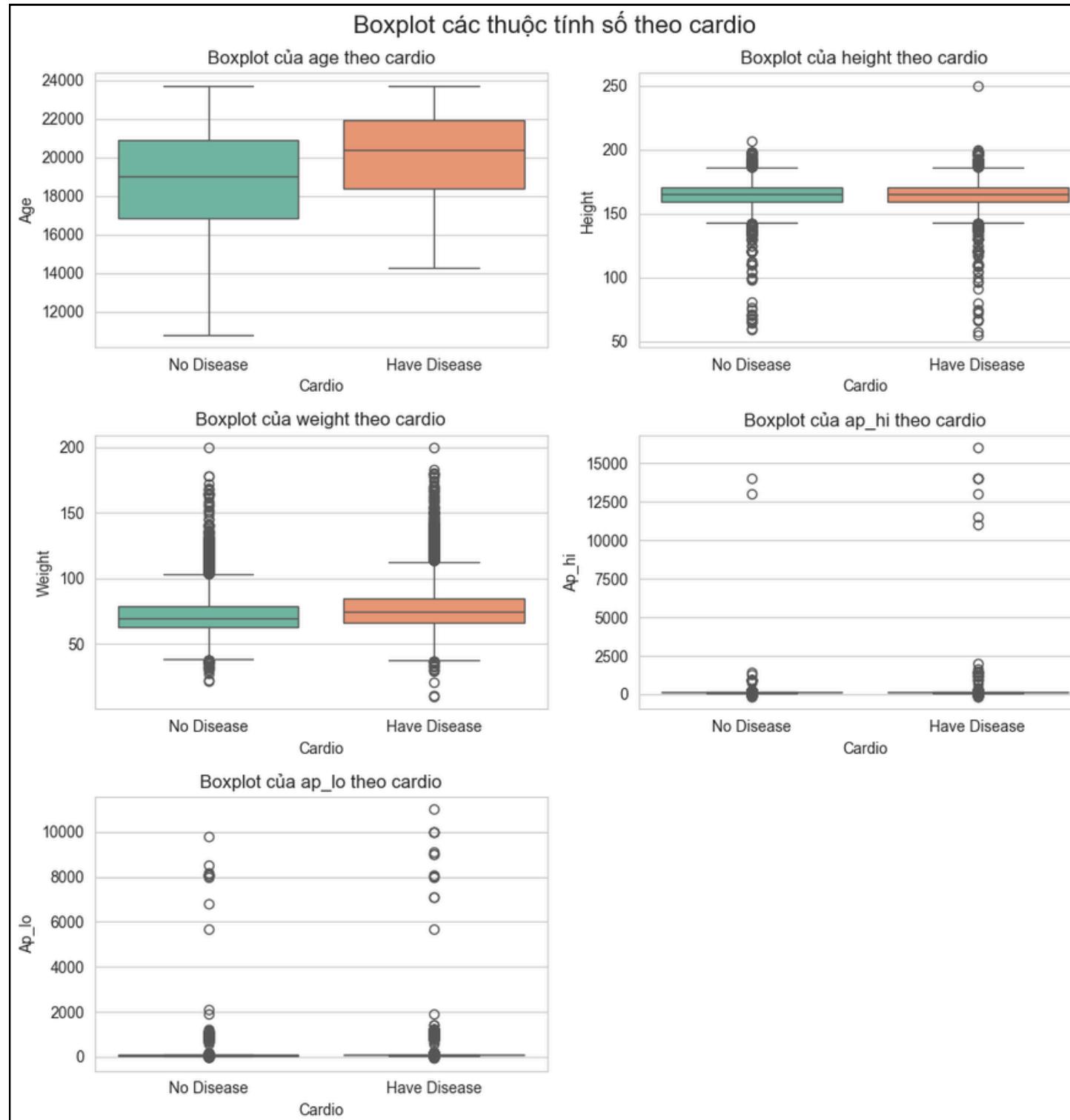
# Phân tích mối quan hệ giữa biến numerical và biến categorical



# Phân tích mối quan hệ giữa biến numerical và biến categorical



# Phân tích mối quan hệ giữa biến và mục tiêu



—

02

# Tiền xử lý dữ liệu

# Loại bỏ outliers

Áp dụng kỹ thuật thống kê IQR (Interquartile Range) để loại bỏ ngoại lệ.

```
def remove_outliers_iqr(x, y, columns):
    x_clean = x.copy()
    y_clean = y.copy()

    for col in columns:
        Q1 = x_clean[col].quantile(0.25)
        Q3 = x_clean[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        mask = (x_clean[col] >= lower_bound) & (x_clean[col] <= upper_bound)
        x_clean = x_clean[mask]
        y_clean = y_clean[mask]

    return x_clean, y_clean
```

# Loại bỏ outliers

Loại bỏ các giá trị ngoại lệ bằng kỹ thuật cắt ngưỡng (Clip)

```
def clip_outliers(df):
    df_clipped = df.copy()
    df_clipped['height'] = df_clipped['height'].clip(lower=100, upper=220)
    df_clipped['weight'] = df_clipped['weight'].clip(lower=30, upper=180)
    df_clipped['ap_hi'] = df_clipped['ap_hi'].clip(lower=60, upper=250)
    df_clipped['ap_lo'] = df_clipped['ap_lo'].clip(lower=40, upper=150)

    # Loại bỏ hàng sai logic
    df_clipped = df_clipped[df_clipped['ap_hi'] > df_clipped['ap_lo']]
    return df_clipped
```

# Thêm các cột đặc trưng mới

Thêm các đặc trưng mới như tuổi (tính theo năm), chỉ số BMI được tính từ cân nặng và chiều cao của bệnh nhân

```
# Đặc trưng mới
df_cleaned['age_years'] = (df_cleaned['age'] / 365).astype(int)
df_cleaned['BMI'] = df_cleaned['weight'] / (df_cleaned['height'] / 100)**2
```

	<b>id</b>	<b>age</b>	<b>gender</b>	<b>height</b>	<b>weight</b>	<b>ap_hi</b>	<b>ap_lo</b>	<b>cholesterol</b>	<b>gluc</b>	<b>smoke</b>	<b>alco</b>	<b>active</b>	<b>cardio</b>	<b>age_years</b>	<b>BMI</b>
0	0	18393	2	168	62.0	110	80	1	1	0	0	1	0	50	21.967120
1	1	20228	1	156	85.0	140	90	3	1	0	0	1	1	55	34.927679
2	2	18857	1	165	64.0	130	70	3	1	0	0	0	1	51	23.507805
3	3	17623	2	169	82.0	150	100	1	1	0	0	1	1	48	28.710479
4	4	17474	1	156	56.0	100	60	1	1	0	0	0	0	47	23.011177

# Chia train, test và chuẩn hóa dữ liệu

```
X = df_cleaned.drop(columns=[target, 'id', 'age'])  
y = df_cleaned[target]
```

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

# Chia train, test và chuẩn hóa dữ liệu

```
num_transformer = Pipeline(steps=[  
    ("scaler", StandardScaler())  
])  
  
cate_tranformer = Pipeline(steps=[  
    ("encoder", OneHotEncoder(drop='first'))  
])  
  
preprocessor = ColumnTransformer(transformers=[  
    ("cate", cate_tranformer, categorical_cols),  
    ("num", num_transformer, numerical_cols)  
], remainder='passthrough')
```

---

03

# Các thuật toán phân loại

# Danh sách các thuật toán

1

**Logistic Regression**

2

**Naive Bayes**

3

**K-NN**

4

**Decision Tree,  
Random Forest**

5

**XGBoost**

6

**LightGBM**

7

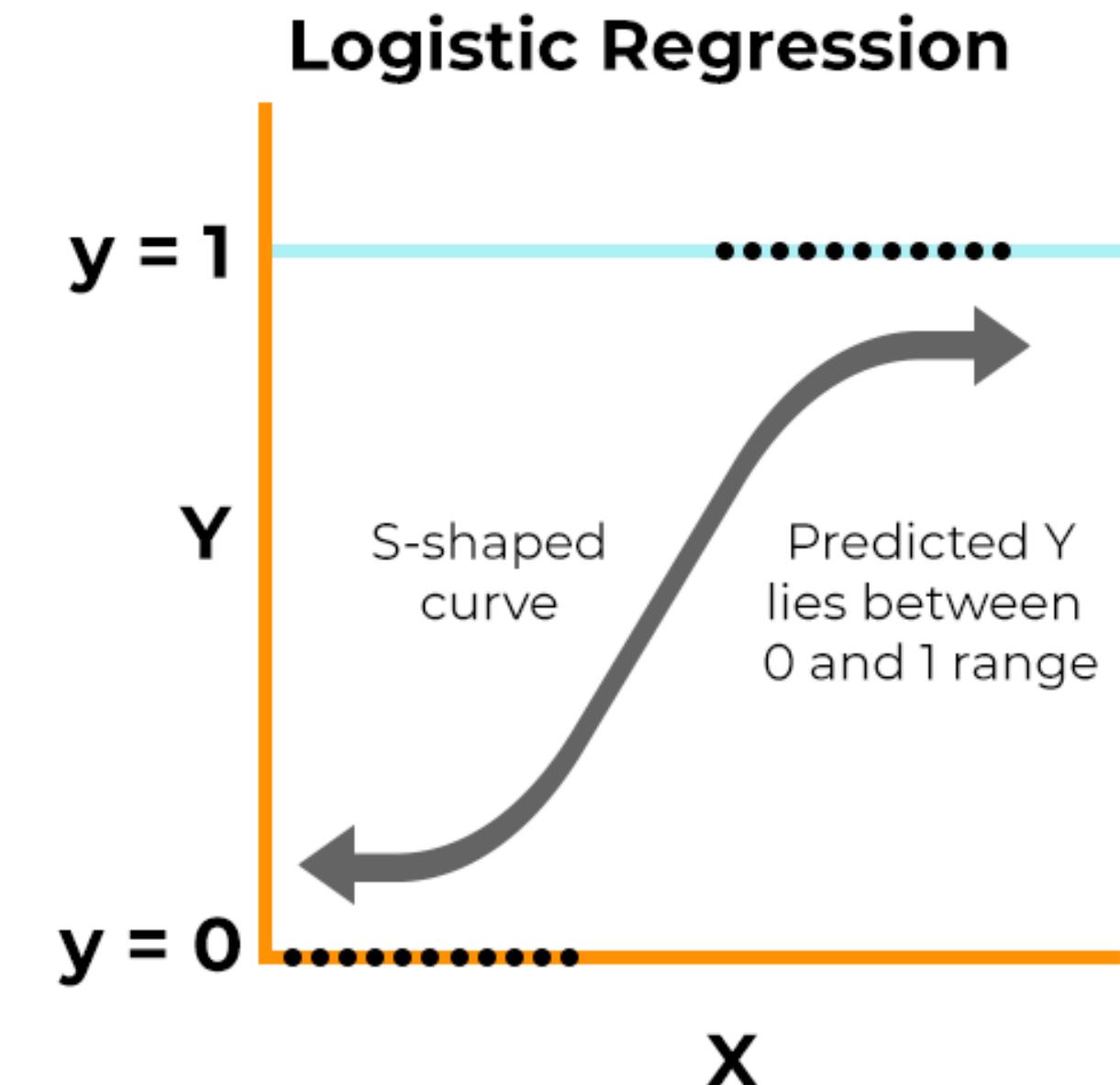
**SVM**

8

**MLP**

# Logistic Regression

- Nguyên lý: Dự đoán xác suất xảy ra một sự kiện nhị phân (0 hoặc 1) dựa trên mô hình tuyến tính kết hợp các đặc trưng
- Thuật toán đơn giản, dễ triển khai
- Có khả năng diễn giải rõ ràng, có thể hiểu được vai trò của từng biến đầu vào



# Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

log_reg = Pipeline(steps=[
    ("preprocess", preprocess),
    ("model", LogisticRegression(max_iter=1000))
])
log_reg.fit(X_train, y_train)
evaluate_model(log_reg, X_test, y_test)
```

# Logistic Regression

Accuracy : 0.7271

Precision: 0.7569

Recall : 0.6650

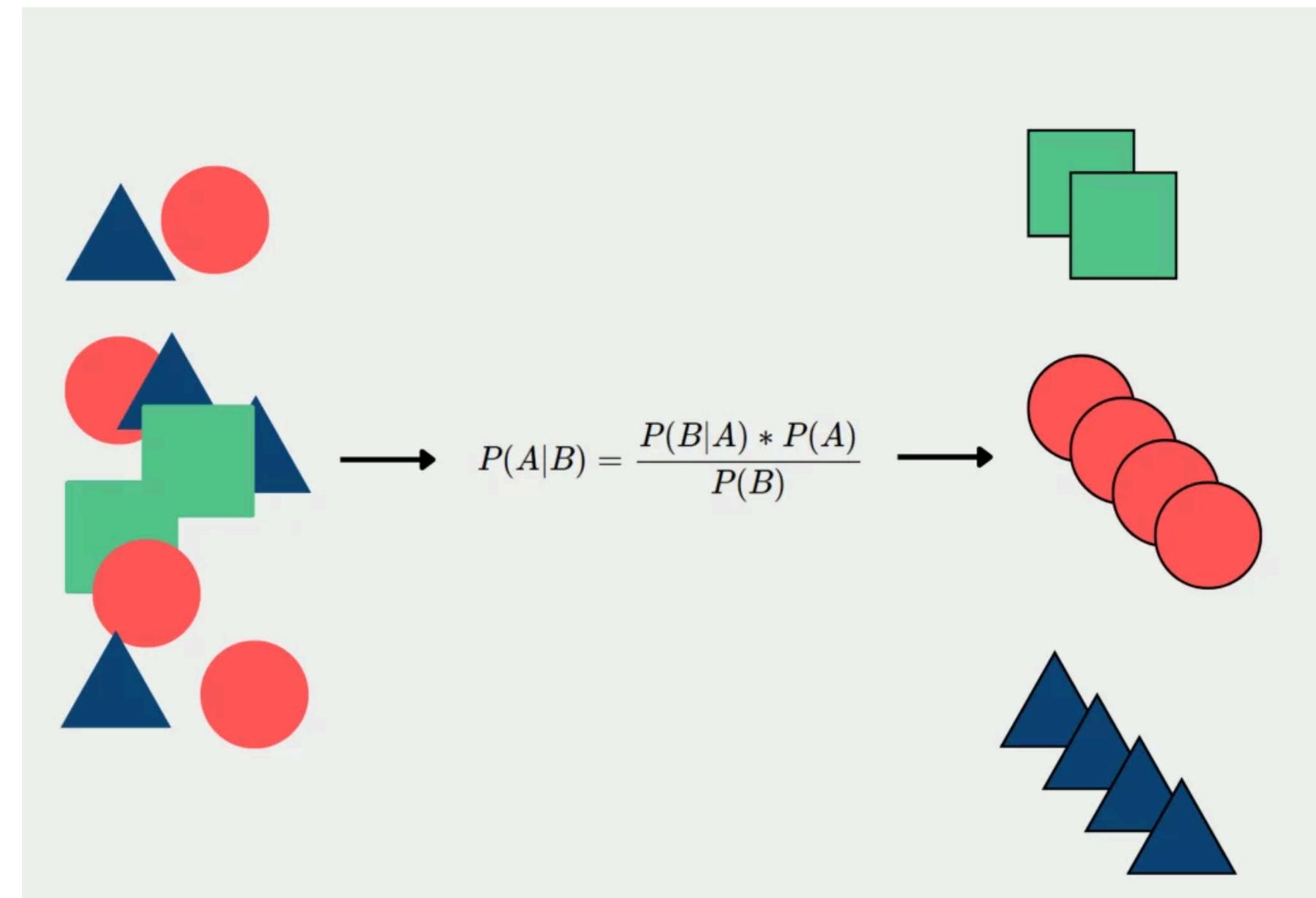
F1-score : 0.7079

Classification Report:

	precision	recall	f1-score	support
0	0.70	0.79	0.74	6957
1	0.76	0.66	0.71	6886
accuracy			0.73	13843
macro avg	0.73	0.73	0.73	13843
weighted avg	0.73	0.73	0.73	13843

# Naive Bayes

- Nguyên lý: Dựa trên định lý Bayes, giả định rằng các đặc trưng là độc lập với nhau
- Thuật toán nhanh, hiệu quả trên dữ liệu nhiều chiều
- Hoạt động tốt với dữ liệu categorical



# Naive Bayes

```
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.pipeline import Pipeline

gauss_nb = Pipeline(steps=[
    ("preprocess", preprocessor),
    ("model", GaussianNB()))
])
gauss_nb.fit(X_train, y_train)
evaluate_model(gauss_nb, X_test, y_test)
```

# Naive Bayes

Accuracy : 0.6855

Precision: 0.7417

Recall : 0.5642

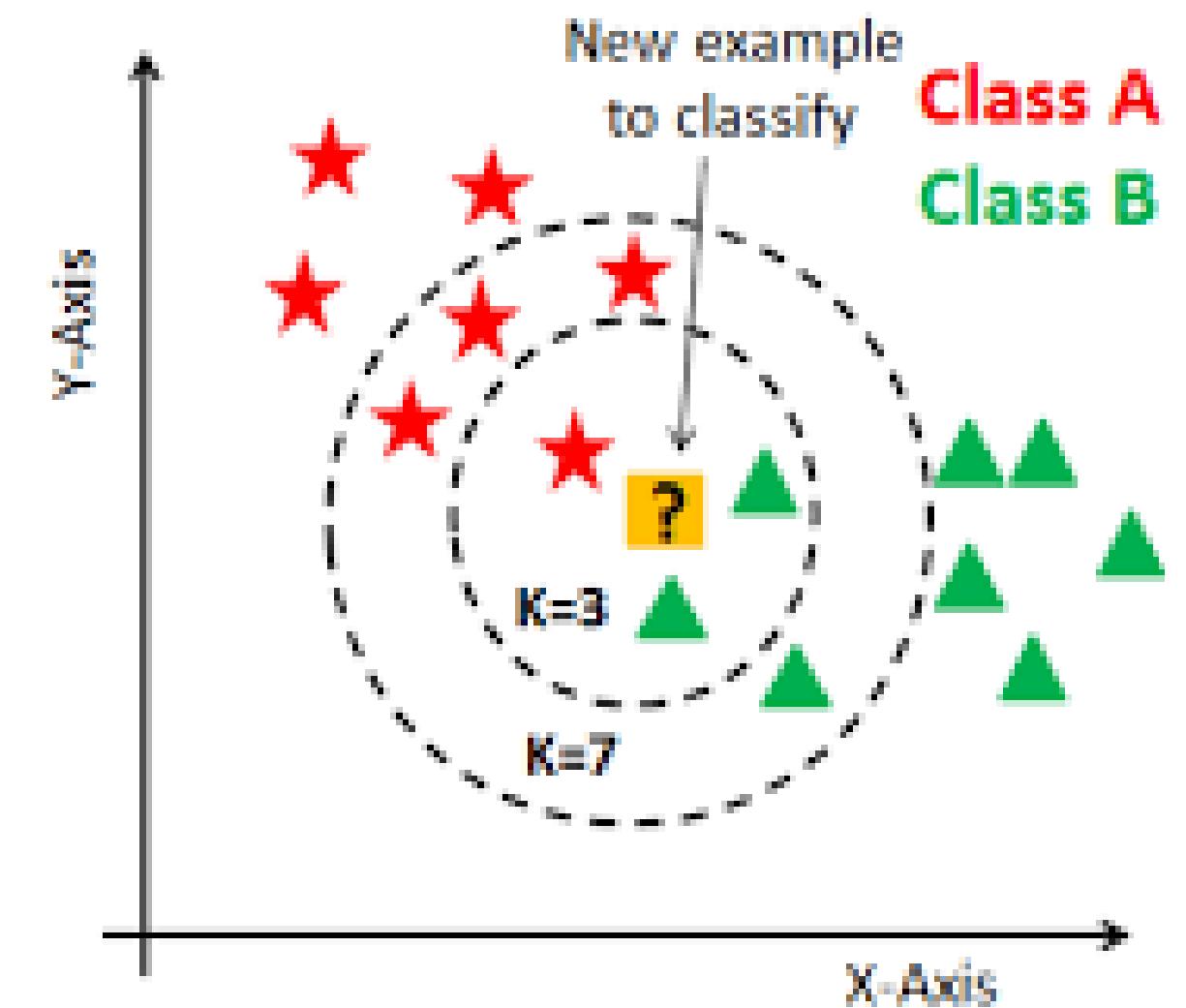
F1-score : 0.6409

Classification Report:

	precision	recall	f1-score	support
0	0.65	0.81	0.72	6957
1	0.74	0.56	0.64	6886
accuracy			0.69	13843
macro avg	0.70	0.68	0.68	13843
weighted avg	0.70	0.69	0.68	13843

# k-Nearest Neighbour

- Nguyên lý: Phân loại dựa vào nhãn của k điểm gần nhất trong không gian đặc trưng
- Thuật toán dễ hiểu, không cần huấn luyện mô hình
- Nhạy cảm với nhiễu và thang đo



# k-Nearest Neighbour

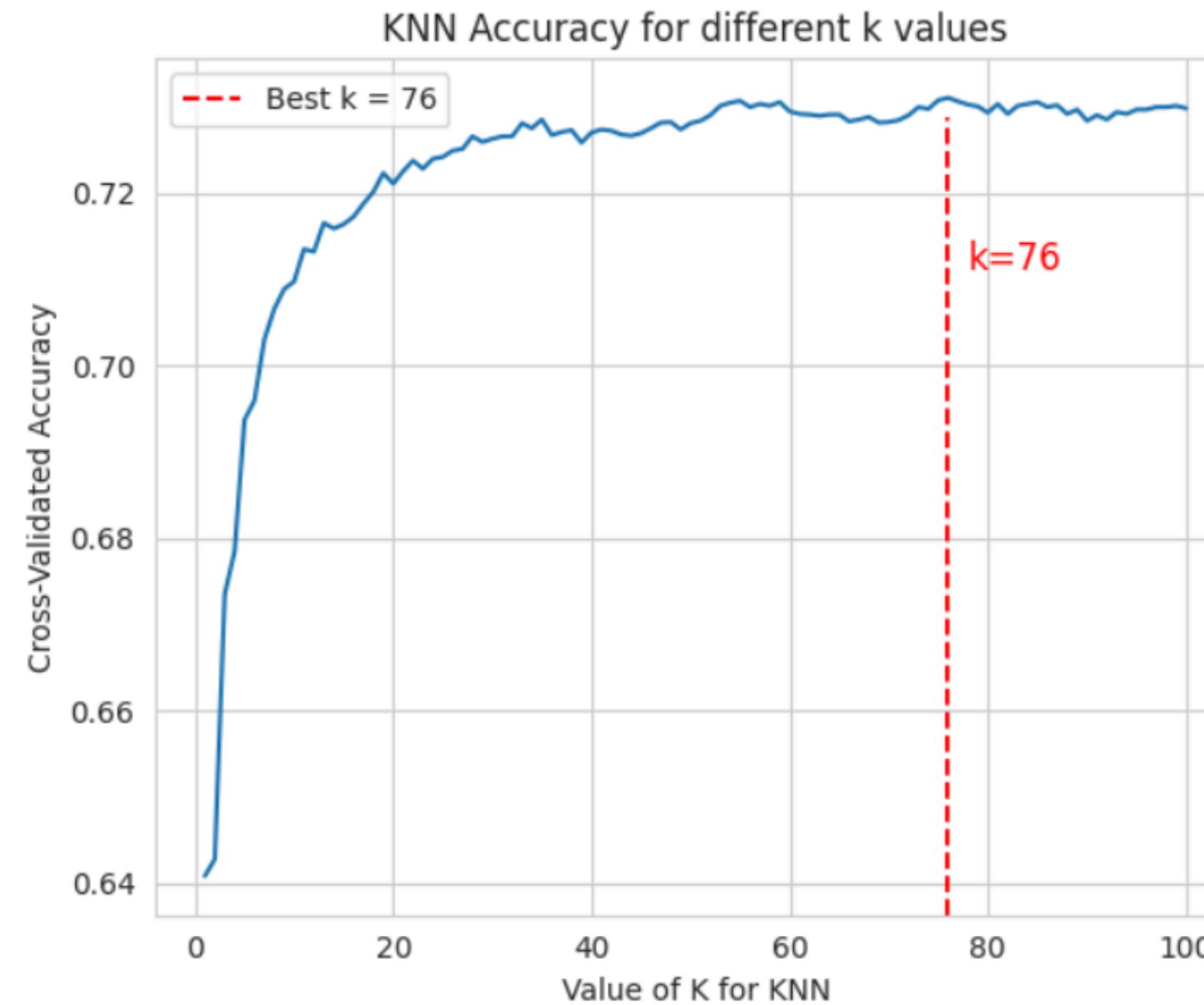
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, roc_curve

# 4. Tìm best k
from sklearn.model_selection import cross_val_score
k_range = list(range(1, 101))
scores = []

for k in k_range:
    knn_pipe = Pipeline(steps=[
        ("preprocessing", preprocessor),
        ("classifier", KNeighborsClassifier(n_neighbors=k))
    ])
    knn_pipe.fit(X_train, y_train)
    score = knn_pipe.score(X_test, y_test)
    scores.append(score)

# 5. Tìm k tốt nhất
m = scores.index(max(scores)) + 1
print(f"Best k = {m}, Cross-Validated Accuracy = {scores[m-1]:.4f}")
```

# k-Nearest Neighbour



# k-Nearest Neighbour

Accuracy : 0.7311

Precision: 0.7688

Recall : 0.6568

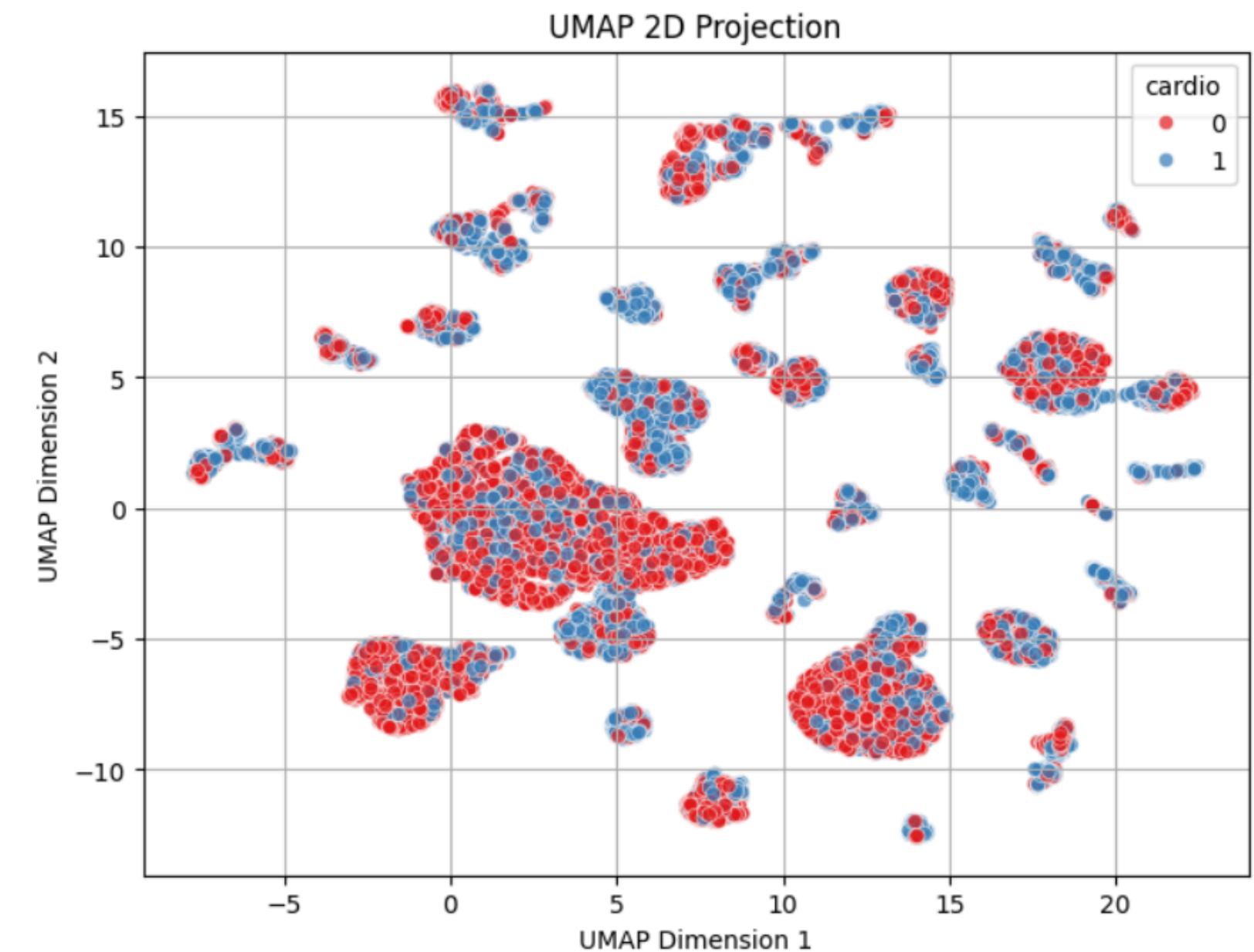
F1-score : 0.7084

Classification Report:

	precision	recall	f1-score	support
0	0.70	0.80	0.75	6957
1	0.77	0.66	0.71	6886
accuracy			0.73	13843
macro avg	0.74	0.73	0.73	13843
weighted avg	0.74	0.73	0.73	13843

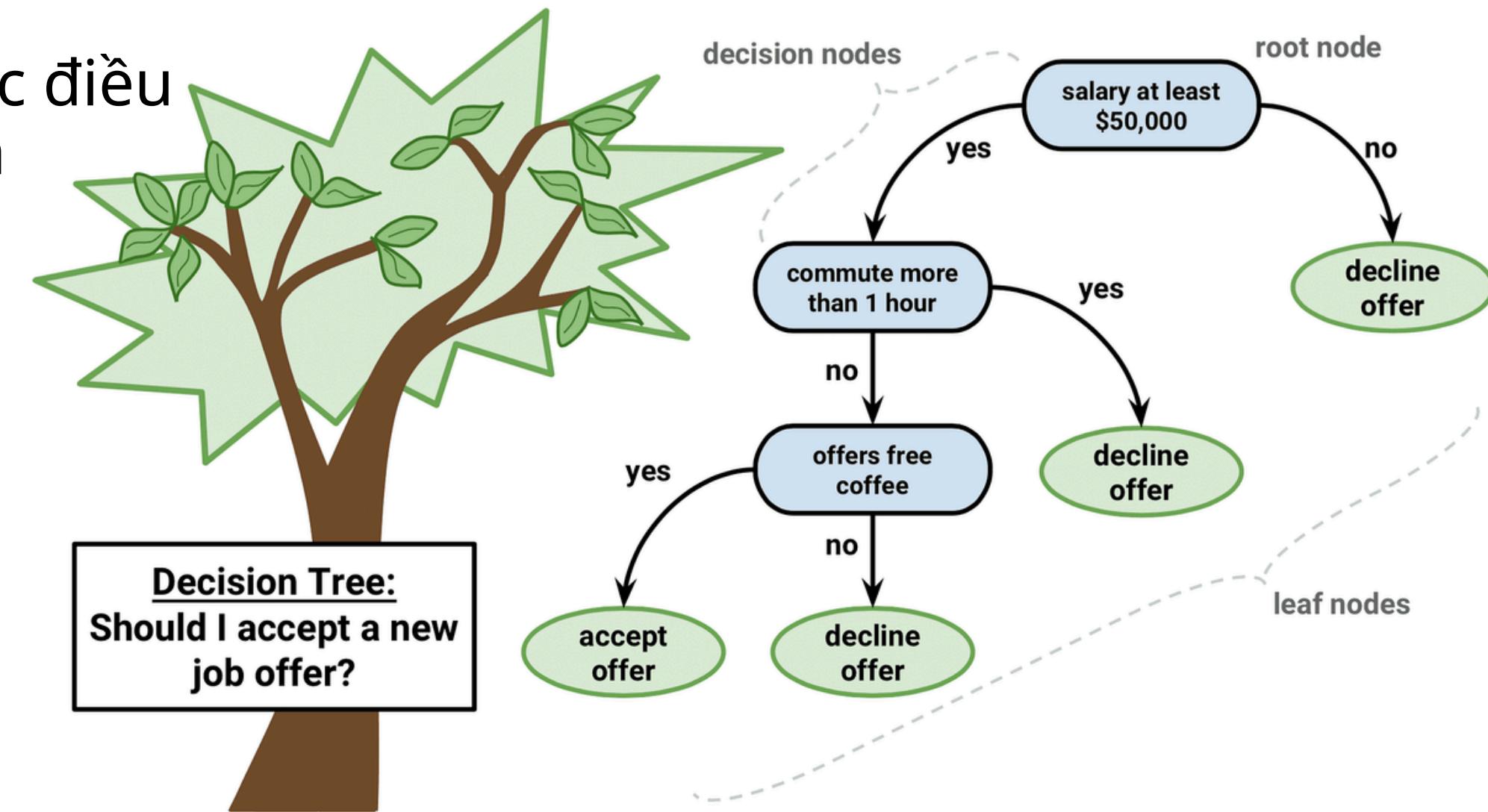
# Nhận xét

- Dữ liệu có phân cụm rõ ràng nhưng biên ranh giới không tuyến tính, gợi ý rằng mô hình Non-Linear sẽ phù hợp hơn để nắm bắt quan hệ giữa các đặc trưng gốc
- UMAP giữ được cấu trúc cục bộ, nên nếu các cụm hiện tại chồng lấn nhẹ nhưng không trộn lẫn hoàn toàn, thì các mô hình như SVM với kernel, XGBoost, hoặc Neural Network có thể khai thác tốt



# Decision Tree

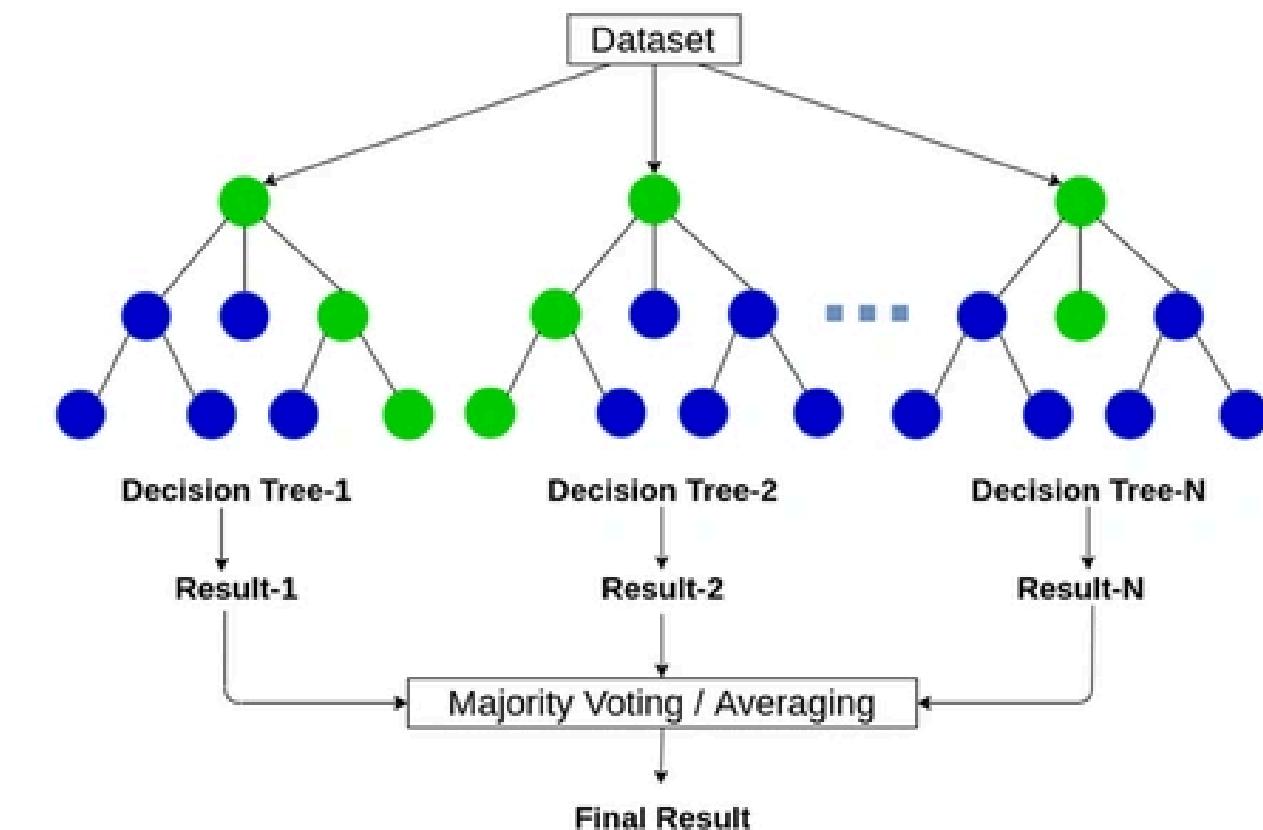
- Nguyên lý: Phân chia dữ liệu theo các điều kiện để tối đa hóa sự phân biệt nhau
- Thuật toán dễ hiểu, có thể trực quan hóa
- Variance cao(dễ overfit), không có Additive Structure



# Random Forest

- Nguyên lý: Cải tiến Decision Tree bằng Bagging - tập hợp nhiều cây quyết định, mỗi cây huấn luyện trên một phần dữ liệu khác nhau sau đó tổng hợp kết quả
- Giảm variance và tăng Bias(giải quyết vấn đề của Decision Tree)
- Không dễ giải thích như Decision Tree

# Random Forest



# Decision Tree and Random Forest

```
param_grid_tree = {  
    'max_depth': [3, 5, 10, None],  
    'min_samples_split': [2, 5, 10]  
}  
  
param_grid_rf = {  
    'n_estimators': [100, 200],  
    'max_depth': [5, 10, None],  
    'min_samples_split': [2, 5]  
}
```

```
models = {  
    "Decision Tree": (DecisionTreeClassifier(random_state=42), param_grid_tree),  
    "Random Forest": (RandomForestClassifier(random_state=42), param_grid_rf),  
}
```

```
for name, (model, param_grid) in models.items():  
    print(f"{name}")  
    grid = GridSearchCV(model, param_grid, cv=3, scoring='accuracy', n_jobs=-1, verbose=0)  
    grid.fit(X_train, y_train)  
    best_model = grid.best_estimator_  
    print(f"{name} - Best Params: {grid.best_params_}")  
    evaluate_model(best_model, X_test, y_test)
```

# Decision Tree and Random Forest

```
results = {}

for name, (model, param_grid) in models.items():
    print(f"{name}")
    grid = GridSearchCV(model, param_grid, cv=3, scoring='accuracy', n_jobs=-1, verbose=0)
    grid.fit(X_train, y_train)

    best_model = grid.best_estimator_
    y_pred = best_model.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, average='weighted', zero_division=0)
    rec = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')

    results[name] = {"Accuracy": acc, "Precision": prec, "Recall": rec, "F1-score": f1}

    print(f"{name} - Best Params: {grid.best_params_}")
    print(f"  Accuracy  = {acc:.4f}")
    print(f"  Precision = {prec:.4f}")
    print(f"  Recall    = {rec:.4f}")
    print(f"  F1-score  = {f1:.4f}\n")
```

# Decision Tree and Random Forest

Decision Tree

Decision Tree - Best Params: {'max\_depth': 5, 'min\_samples\_split': 2}

Accuracy : 0.7326

Precision: 0.7882

Recall : 0.6323

F1-score : 0.7017

Random Forest

Random Forest - Best Params: {'max\_depth': 10, 'min\_samples\_split': 2, 'n\_estimators': 200}

Accuracy : 0.7354

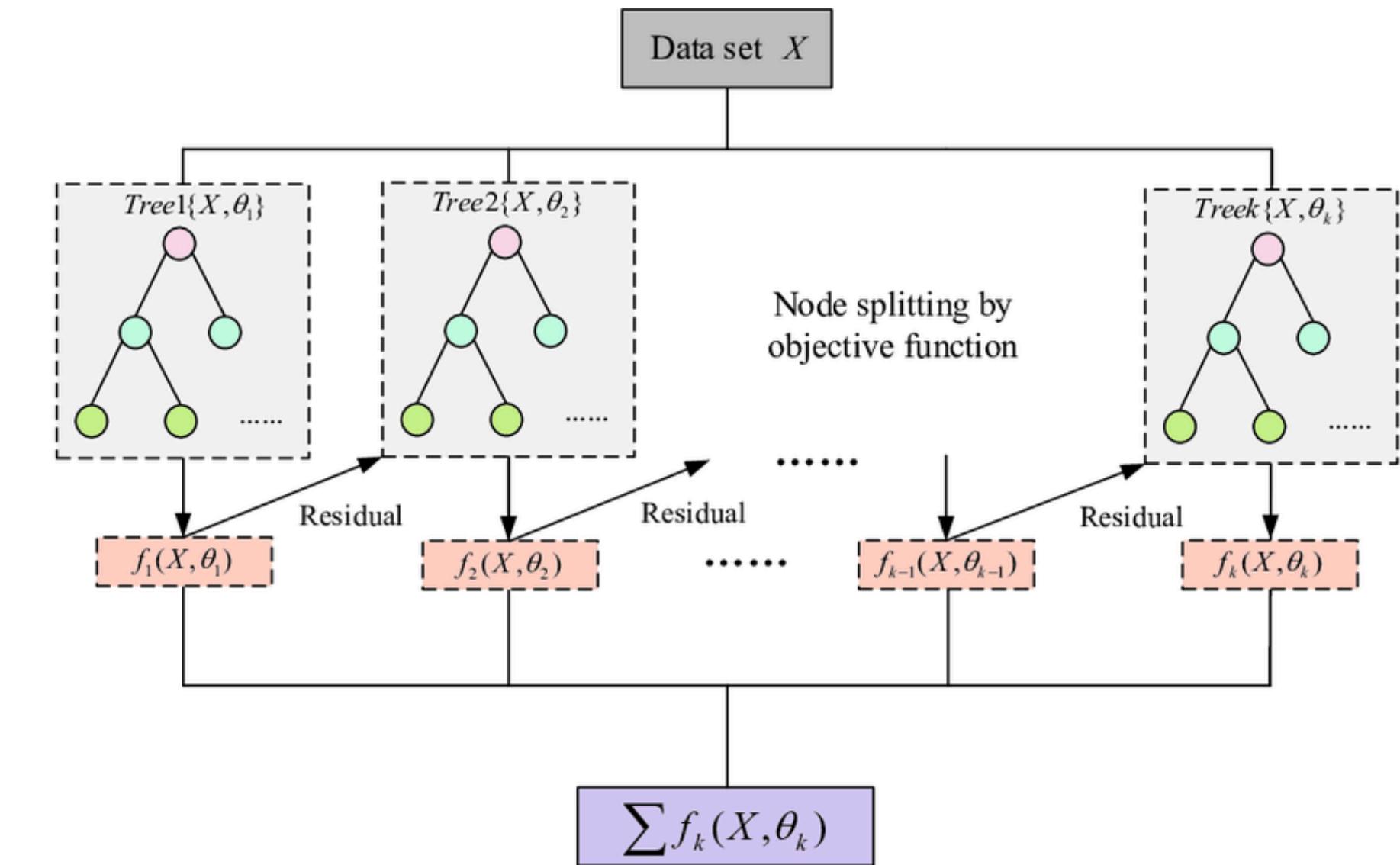
Precision: 0.7660

Recall : 0.6740

F1-score : 0.7170

# XGBoost

- Nguyên lý: Cải tiến Decision Tree bằng Boosting - cây sau học từ lỗi của cây trước
- Thuật toán chính xác cao, xử lý outlier và missing tốt
- Giảm Bias, không giải quyết được vấn đề Variance cao của Decision Tree nên vẫn dễ overfit



# XGBoost

```
xgboost = Pipeline(steps=[  
    ("preprocessing", preprocessor),  
    ("classifier", xgb.XGBClassifier(  
        n_estimators=100,  
        learning_rate=0.1,  
        max_depth=5,  
        random_state=42,  
        use_label_encoder=False,  
        eval_metric='logloss'  
    ))  
])  
  
# 5. Huấn luyện  
xgboost.fit(X_train, y_train)  
  
evaluate_model(xgboost, X_test, y_test)
```

# XGBoost

Accuracy : 0.7358

Precision: 0.7607

Recall : 0.6840

F1-score : 0.7203

Classification Report:

	precision	recall	f1-score	support
0	0.72	0.79	0.75	6957
1	0.76	0.68	0.72	6886
accuracy			0.74	13843
macro avg	0.74	0.74	0.73	13843
weighted avg	0.74	0.74	0.74	13843

# XGBoost - Hyperparameter tuning

```
param_dist = {  
    'n_estimators': randint(100, 300),  
    'max_depth': randint(3, 10),  
    'learning_rate': uniform(0.01, 0.2),  
    'subsample': uniform(0.7, 0.3),  
    'colsample_bytree': uniform(0.7, 0.3)  
}  
  
search = RandomizedSearchCV(  
    estimator=xgb_clf,  
    param_distributions=param_dist,  
    n_iter=30,           # thử 30 tổ hợp ngẫu nhiên  
    scoring='f1',  
    cv=3,  
    random_state=42,  
    n_jobs=-1,  
    verbose=1  
)
```

# XGBoost - Hyperparameter tuning

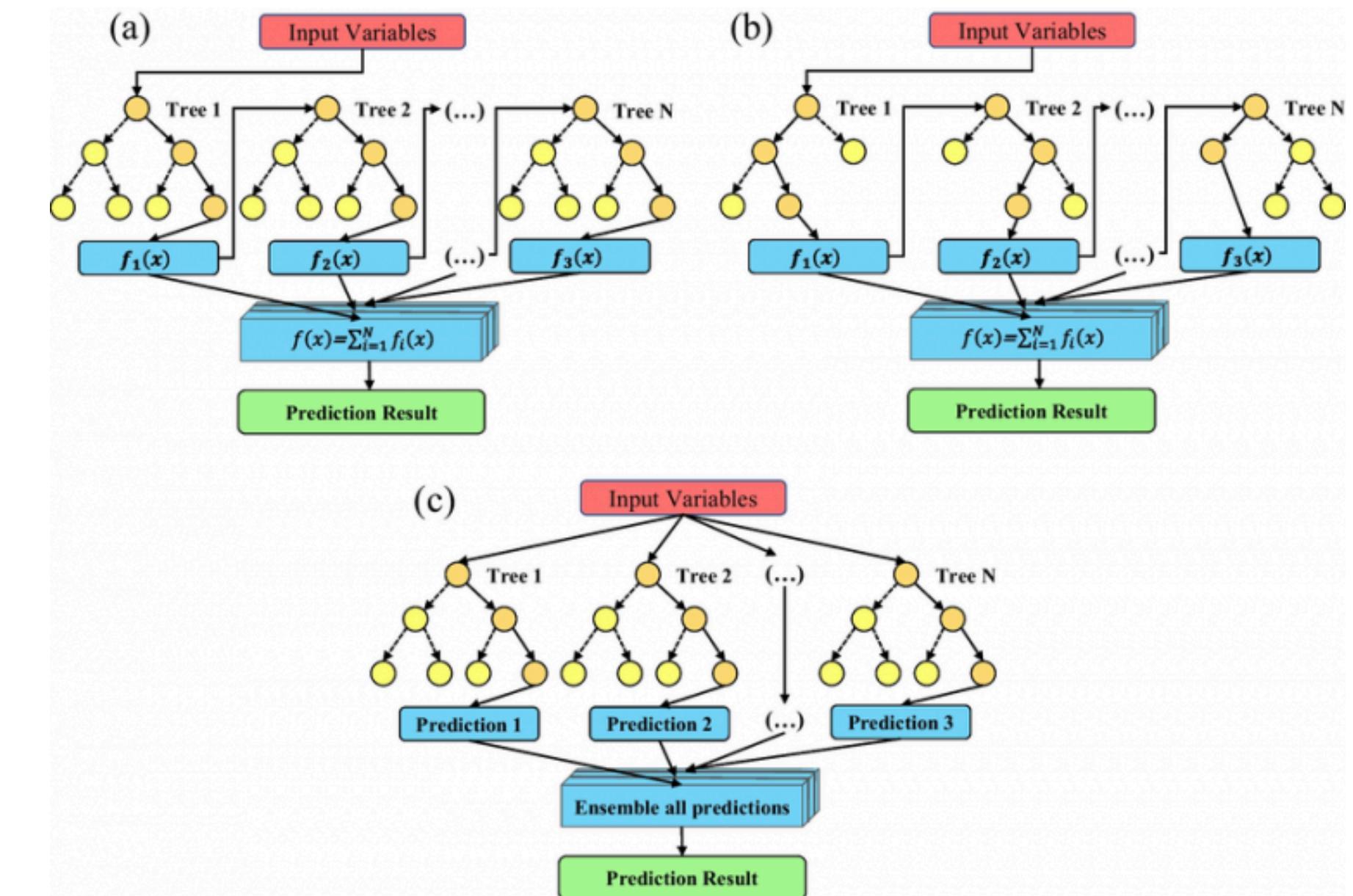
```
Fitting 3 folds for each of 30 candidates, totalling 90 fits
Best parameters found:
{'colsample_bytree': 0.7468055921327309, 'learning_rate': 0.041198904067240534, 'max_depth':
5, 'n_estimators': 187, 'subsample': 0.8001125833417065}
Accuracy : 0.7427
Precision: 0.7597
Recall   : 0.7025
F1-score : 0.7300

Classification Report:
precision    recall  f1-score   support
          0       0.73      0.78      0.75     6989
          1       0.76      0.70      0.73     6854

accuracy                           0.74     13843
macro avg       0.74      0.74      0.74     13843
weighted avg    0.74      0.74      0.74     13843
```

# LightGBM

- Nguyên lý: Boosting giống XGBoost, nhưng sử dụng leaf-wise tree growth
- Thuật toán tăng tốc độ đáng kể so với XGBoost
- Vẫn dễ overfit do Variance cao



# LightGBM

```
params = {
    'learning_rate': 0.05,
    'max_depth': -1,
    'n_estimators': 100,
    'num_leaves': 50
}
lgbm_clf = LGBMClassifier(**params, random_state=42, verbosity=-1)

lgbm = Pipeline(steps=[
    ("preprocessing", preprocessor),
    ("classifier", lgbm_clf)
])

lgbm.fit(X_train, y_train)

evaluate_model(lgbm, X_test, y_test)
```

# LightGBM

Accuracy : 0.7373

Precision: 0.7600

Recall : 0.6895

F1-score : 0.7231

Classification Report:

	precision	recall	f1-score	support
0	0.72	0.78	0.75	6957
1	0.76	0.69	0.72	6886
accuracy			0.74	13843
macro avg	0.74	0.74	0.74	13843
weighted avg	0.74	0.74	0.74	13843

# LightGBM - Hyperparameter tuning

```
param_grid = {  
    'num_leaves': [31, 50, 70],  
    'max_depth': [-1, 10, 20],  
    'learning_rate': [0.05, 0.1, 0.2],  
    'n_estimators': [100, 200]  
}  
  
grid_search = GridSearchCV(  
    estimator=lgbm_clf,  
    param_grid=param_grid,  
    scoring='f1', # hoặc 'accuracy', 'recall', tùy bài toán  
    cv=5,  
    verbose=1,  
    n_jobs=-1  
)  
  
grid_search.fit(X_train, y_train)
```

# LightGBM - Hyperparameter tuning

Accuracy : 0.7380

Precision: 0.7620

Recall : 0.6882

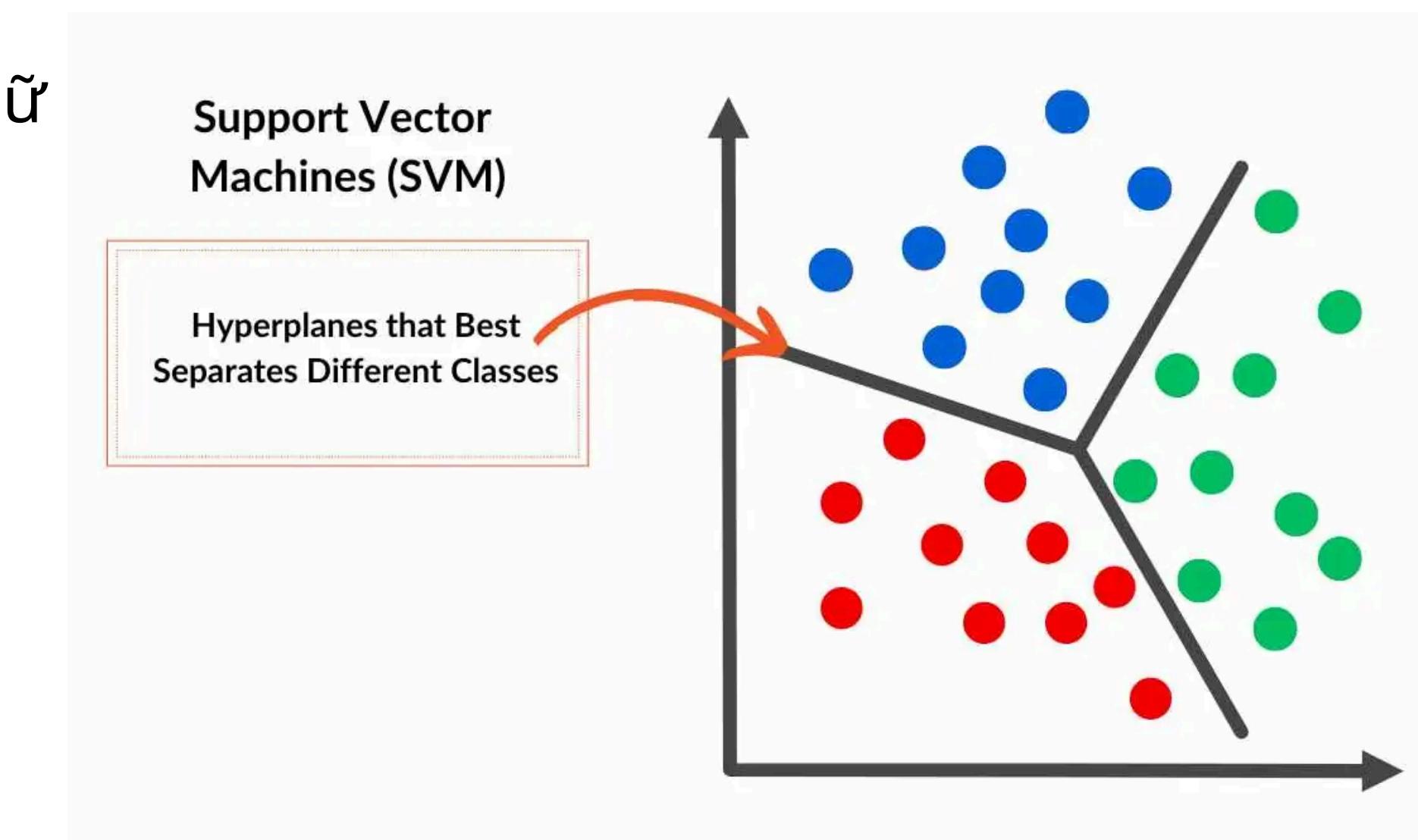
F1-score : 0.7232

Classification Report:

	precision	recall	f1-score	support
0	0.72	0.79	0.75	6957
1	0.76	0.69	0.72	6886
accuracy			0.74	13843
macro avg	0.74	0.74	0.74	13843
weighted avg	0.74	0.74	0.74	13843

# Support Vector Machine

- Nguyên lý: Tìm hyperplane phân chia dữ liệu có margin tối đa
- Thuật toán tốt với không gian đặc trưng cao, phân lớp rõ ràng
- Có thể xử lý quan hệ phi tuyến qua kernel trick.



# Support Vector Machine

```
from sklearn.svm import SVC

svm_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('svm', SVC(kernel='rbf', C=1, gamma='scale', class_weight='balanced'))
])

svm_pipeline.fit(X_train, y_train)
evaluate_model(svm_pipeline, X_test, y_test)
```

# Support Vector Machine

Accuracy : 0.7357

Precision: 0.7659

Recall : 0.6750

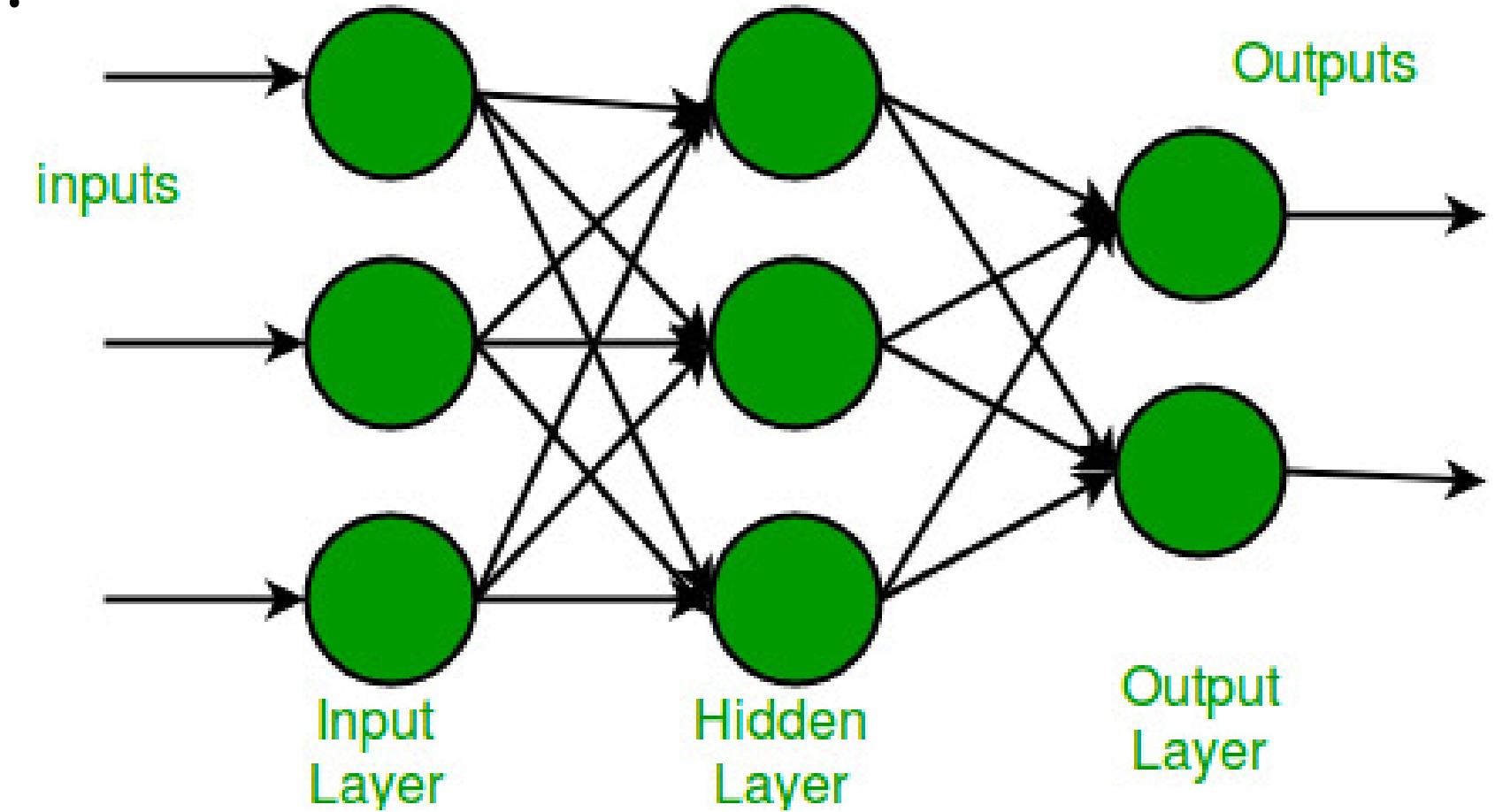
F1-score : 0.7176

Classification Report:

	precision	recall	f1-score	support
0	0.71	0.80	0.75	6957
1	0.77	0.67	0.72	6886
accuracy			0.74	13843
macro avg	0.74	0.74	0.73	13843
weighted avg	0.74	0.74	0.73	13843

# Multi-Layer Perceptron

- Mạng nơ-ron nhiều tầng (feedforward): MLP gồm input layer- nhiều hidden layers – output layer, mỗi tầng là các perceptron kết nối đầy đủ.
- Activation functions: sử dụng các hàm ReLU, sigmoid, tanh, ... để học các mối quan hệ phức tạp
- Học bằng backpropagation: Trọng số được tối ưu bằng lan truyền ngược + gradient descent, giảm sai số trên tập huấn luyện



# Multi-Layer Perceptron

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

X_train = preprocessor.fit_transform(X_train).astype(np.float32)
X_test = preprocessor.fit_transform(X_test).astype(np.float32)
# TensorDataset
train_dataset = TensorDataset(torch.tensor(X_train), torch.tensor(y_train.values, dtype=torch.float32))
test_dataset = TensorDataset(torch.tensor(X_test), torch.tensor(y_test.values, dtype=torch.float32))

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64)
```

# Multi-Layer Perceptron

```
class MLP(nn.Module):
    def __init__(self, input_dim):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(input_dim, 64),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(64, 32),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(32, 1),
            nn.Sigmoid()
        )

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

# Multi-Layer Perceptron

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = MLP(input_dim=X_train.shape[1]).to(device)

criterion = nn.BCELoss()
optimizer = Adam(model.parameters(), lr=1e-3)

for epoch in range(1, 21):
    model.train()
    train_loss = 0
    for xb, yb in train_loader:
        xb, yb = xb.to(device), yb.to(device).unsqueeze(1)
        optimizer.zero_grad()
        preds = model(xb)
        loss = criterion(preds, yb)
        loss.backward()
        optimizer.step()
        train_loss += loss.item() * xb.size(0)

    avg_loss = train_loss / len(train_loader.dataset)
    print(f"Epoch {epoch}, Train Loss: {avg_loss:.4f}")
```

# Multi-Layer Perceptron

```
X_test = []
y_test = []

for xb, yb in test_loader:
    xb = xb.to(device)
    with torch.no_grad():
        preds = model(xb)
    X_test.append(preds.cpu().numpy())
    y_test.append(yb.cpu().numpy())

X_test = np.concatenate(X_test, axis=0)
y_test = np.concatenate(y_test, axis=0)

class WrapperModel:
    def __init__(self, raw_preds):
        self.raw_preds = raw_preds

    def predict(self, X):
        return (self.raw_preds > 0.5).astype(int).flatten()

mock_model = WrapperModel(X_test)

evaluate_model(mock_model, X_test, y_test)
```

# Multi-Layer Perceptron

Accuracy : 0.7415

Precision: 0.7622

Recall : 0.6948

F1-score : 0.7269

Classification Report:

	precision	recall	f1-score	support
0.0	0.72	0.79	0.75	6989
1.0	0.76	0.69	0.73	6854
accuracy			0.74	13843
macro avg	0.74	0.74	0.74	13843
weighted avg	0.74	0.74	0.74	13843

—

04

# Kết luận

Mô hình	Accuracy	Precision	Recall	F1-score
Logistic Regression	0.7271	0.7596	0.6650	0.7079
Naive Bayes	0.6855	0.7417	0.5642	0.6409
KNN	0.7311	0.7688	0.6568	0.7084
Decision Tree	0.7326	<b>0.7882</b>	0.6323	0.7017
Random Forest	0.7354	0.7660	0.6740	0.7170
<b>XGBoost</b>	<b>0.7427</b>	0.7597	<b>0.7025</b>	<b>0.7300</b>
LightGBM	0.7380	0.7620	0.6882	0.7232
SVM	0.7357	0.7659	0.6750	0.7176
MLP	0.7415	0.7622	0.6948	0.7269

---

05

# Định hướng phát triển

- Tối ưu XGBoost hơn nữa bằng cách tinh chỉnh siêu tham số như learning rate, max\_depth, n\_estimators và áp dụng kỹ thuật early stopping để cải thiện hiệu suất và tránh overfitting.
- Thử TabNet, TabTransformer
- Xây dựng một ứng dụng giao diện đơn giản bằng Streamlit, cho phép người dùng nhập thông tin sức khỏe và nhận kết quả dự đoán nguy cơ mắc bệnh tim một cách trực quan, thân thiện.
- Thử nghiệm mô hình đã huấn luyện với các bộ dữ liệu bệnh lý khác hoặc dữ liệu từ hệ thống y tế địa phương để kiểm tra khả năng khái quát (generalization) và độ tin cậy của mô hình trong các bối cảnh thực tế khác nhau.



# The End

CẢM ƠN THẦY VÀ CÁC BẠN ĐÃ LẮNG NGHE