ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

BÁO CÁO CUỐI KÌ

Môn: Python cho Khoa học Dữ liệu

Nhóm 26

Đề tài:

Dự đoán Bệnh Tim Mạch (Heart Disease Prediction)

Giảng viên phụ trách:

Hà Minh Tuấn

Thành viên thực hiện:

Đỗ Lê Nguyên Đan (MSSV: 23280044)
Lương Lê Công Hạnh (MSSV: 23280057)
Âu Dương Khả (MSSV: 23280063)

Mục lục

1 Giới thiệu bài toán và dữ liệu.

1.1 Bài toán

Mục tiêu bài toán. Mục tiêu của bài báo cáo nghiên cứu này là xây dựng một mô hình phân loại nhằm dự đoán khả năng mắc bệnh tim mạch của bệnh nhân dựa trên các đặc

trưng sức khoẻ (như tuổi, huyết áp, mức cholesterol, điện tâm đồ, v.v.). Kết quả dự đoán có thể giúp phát hiện sớm nguy cơ bệnh

1.2 Dữ liệu

Nguồn dữ liệu. Báo cáo sử dụng bộ dữ liệu *Heart Disease Data* từ Kaggle[1].

1.3 Phân tích dữ liệu của data gốc

Tên biến	**Kiểu**	**Mô tả**
id	Số nguyên	Mã định danh duy nhất của mỗi bệnh nhân.
age	Số nguyên	Tuổi của bệnh nhân (tính theo năm).
dataset	Chuỗi	Nguồn gốc hoặc nơi thu thập dữ liệu.
sex	Chuỗi	Giới tính của bệnh nhân (*Male/Female*).
cp	Chuỗi	Loại đau ngực, gồm 4 nhóm: *typical angina, atypical angina, non-anginal, asymptomatic.*
trestbps	Số	Huyết áp lúc nghỉ (mmHg) khi nhập viện.
chol	Số	Nồng độ cholesterol trong huyết thanh (mg/dl).
fbs	Boolean	Đường huyết lúc đói $>$ 120 mg/dl hay không (*True/False*).
restecg	Chuỗi	Kết quả điện tâm đồ lúc nghỉ, gồm: *normal, ST-T abnormality, LV hypertrophy.*
thalach	Số	Nhịp tim tối đa đạt được.
exang	Boolean	Đau thắt ngực do vận động gây ra (*True/False*).
oldpeak	Số	Mức độ chênh lệch ST do gắng sức so với lúc nghỉ.
slope	Số	Độ dốc đoạn ST khi gắng sức.
ca	Số nguyên	Số lượng mạch máu lớn (0–3) được nhuộm màu bằng phương pháp soi huỳnh quang.
thal	Chuỗi	Tình trạng thallium: *normal, fixed defect, reversible defect.*
num	Số nguyên	Nhãn dự đoán: 0 (không mắc bệnh) hoặc $>$0 (có bệnh tim).

Bảng 1: Bảng mô tả các biến trong bộ dữ liệu bệnh tim

[1]Nguồn dữ liệu: `https://www.kaggle.com/datasets/redwankarimsony/heart-disease-data`

2 Phân tích các hàm sử dụng

2.1 Tiền xử lý dữ liệu

Lớp `DataPreprocessor` được thiết kế để tự động hóa toàn bộ quy trình tiền xử lý dữ liệu, từ đọc file đến chuẩn hóa và mã hóa. Lớp này đóng vai trò trung tâm trong việc chuẩn bị dữ liệu cho các mô hình học máy, đảm bảo dữ liệu được làm sạch và biến đổi một cách nhất quán.

2.1.1 Tổng quan về lớp DataPreprocessor

Lớp `DataPreprocessor` hỗ trợ xử lý đa dạng các kiểu dữ liệu (numeric, categorical, datetime) với các chức năng chính:

- Đọc dữ liệu từ nhiều định dạng (CSV, Excel, JSON)
- Tự động phát hiện và phân loại kiểu dữ liệu
- Xử lý giá trị thiếu (missing values) thông minh
- Phát hiện và xử lý ngoại lai (outliers) tự động
- Tạo đặc trưng mới (feature engineering)
- Mã hóa biến phân loại (encoding)
- Chuẩn hóa dữ liệu số (normalization/standardization)
- Pipeline xử lý tự động hoàn chỉnh

2.1.2 Các phương thức khởi tạo và tiện ích

1. **__init__:** Phương thức khởi tạo nhận vào tên cột target (tùy chọn) và giá trị seed ngẫu nhiên (mặc định 42). Khởi tạo các biến lưu trữ dữ liệu gốc (`original_data`) và dữ liệu đang xử lý (`data`), cùng với các danh sách phân loại cột theo kiểu (`numeric_columns`, `categorical_columns`, `datetime_columns`). Ngoài ra, khởi tạo các dictionary để lưu trữ scalers và encoders đã được fit (`self.scalers`, `self.encoders`). Cuối cùng, thiết lập hệ thống logging để theo dõi quá trình xử lý.

2. **__repr__:** Trả về chuỗi mô tả đối tượng, hiển thị shape của dữ liệu và tên cột target nếu đã load dữ liệu, hoặc thông báo chưa có dữ liệu.

3. **_setup_logging:** Thiết lập hệ thống logging sử dụng logger chung của module. Nếu chưa có handler, thiết lập cấu hình cơ bản với format bao gồm timestamp, tên module, mức độ (INFO/ERROR/WARNING) và thông điệp.

2.1.3 Đọc và phát hiện dữ liệu

1. **load_data:** Đọc dữ liệu từ file với hỗ trợ đa định dạng (CSV, Excel, JSON). Tự động phát hiện định dạng thông qua phần mở rộng file. Xóa cột 'id' nếu có vì không cần thiết cho modeling. Lưu bản copy vào `original_data` để có thể reset về sau. Tự động gọi `_detect_column_types()` để phân loại các cột. Raise `FileNotFoundError` nếu file không tồn tại hoặc `ValueError` nếu định dạng không được hỗ trợ.

2. **_detect_column_types:** Tự động phát hiện và phân loại kiểu dữ liệu của các cột thành 3 nhóm: `numeric_columns` (int, float), `categorical_columns` (object, string), và `datetime_columns` (datetime hoặc string có thể convert thành datetime). Đối với cột datetime, thử convert với sample 100 giá trị đầu và chỉ coi là datetime nếu tỷ lệ thành công $\geq 80\%$. Bỏ qua cột target khi phân loại.

2.1.4 Xử lý giá trị thiếu

1. **handle_missing_values:** Xử lý giá trị thiếu sử dụng `SimpleImputer` từ sklearn. Cho cột số: hỗ trợ các chiến lược 'median' (trung vị), 'mean' (trung bình), 'most_frequent' (mode), hoặc 'constant' (giá trị cố định). Cho cột phân loại: hỗ trợ 'most_frequent' (mode) hoặc 'constant' (mặc định 'missing_category'). Fit và transform riêng từng cột có giá trị thiếu, sử dụng `.ravel()` để chuyển từ mảng 2D về 1D cho Series. Ghi log số lượng missing còn lại sau xử lý.

2.1.5 Phát hiện và xử lý ngoại lai (Outliers)

1. **detect_outliers_smart:** Phát hiện outliers thông minh dựa trên phân bố dữ liệu. Tự động chọn phương pháp phù hợp:

   - Phân bố lệch nhiều ($|\text{skewness}| > 2$): sử dụng *Isolation Forest*
   - Phân bố gần chuẩn ($|\text{skewness}| < 0.5$): sử dụng *Z-score* với ngưỡng mặc định 3
   - Mặc định: sử dụng *IQR (Interquartile Range)* với multiplier 1.5

   Nhận tham số tùy chỉnh qua kwargs như `z_score_threshold`, `iqr_multiplier`, `isolation_contamination`. Trả về danh sách index của các outliers. Bỏ qua nếu cột có ít hơn 10 giá trị.

2. **handle_outliers:** Xử lý các outliers đã phát hiện theo 3 chiến lược:

   - **drop**: Loại bỏ toàn bộ các hàng chứa outliers
   - **clip**: Clip giá trị outlier về ngưỡng upper/lower được tính từ IQR của dữ liệu non-outlier ($Q1 - 1.5 \times IQR$, $Q3 + 1.5 \times IQR$)
   - **impute**: Thay thế outliers bằng median của dữ liệu non-outlier hoặc giá trị cố định

   Nhận `outliers_dict` dạng {`'tên_cột': [list_index_outlier]`}. Warning nếu không đủ dữ liệu non-outlier để tính ngưỡng.

3. **handle_all_outliers:** Tự động phát hiện và xử lý outliers cho TẤT CẢ các cột số. Quy trình 2 bước: (1) Dùng `detect_outliers_smart()` để phát hiện outliers cho từng cột số, lưu vào `outliers_map`; (2) Dùng `handle_outliers()` để xử lý tất cả outliers đã phát hiện bằng chiến lược đã chọn (clip/drop/impute). Nhận `detection_params` để tùy chỉnh tham số phát hiện. Ghi log tổng số outliers tìm thấy và số cột có outliers.

2.1.6 Tạo đặc trưng mới (Feature Engineering)

1. **create_polynomial_features:** Tạo đặc trưng đa thức (polynomial features) từ các cột số để capture quan hệ phi tuyến. Từ cột x, tạo ra $x^2, x^3, \ldots, x^{\text{degree}}$. Mặc định tạo cho 5 cột numeric đầu tiên với degree=2 để tránh tạo quá nhiều features. Tên cột mới theo format `{tên_cột}_pow{bậc}`. Tự động cập nhật lại danh sách column types sau khi tạo features.

2. **create_interaction_features:** Tạo đặc trưng tương tác (interaction features) giữa các cặp cột để capture tương tác giữa biến. Từ cặp (x, y), tạo 4 features:

   - $x \times y$ (phép nhân): `{col1}_x_{col2}`
   - $x \div y$ (phép chia, tránh chia 0 bằng replace $0 \to 10^{-6}$): `{col1}_div_{col2}`
   - $x + y$ (phép cộng): `{col1}_plus_{col2}`
   - $x - y$ (phép trừ): `{col1}_minus_{col2}`

   Mặc định auto chọn 3 cặp đầu từ `numeric_columns`. Tự động cập nhật lại column types sau khi tạo.

2.1.7 Mã hóa biến phân loại (Encoding)

1. **encode_categorical:** Mã hóa biến phân loại tự động với chiến lược thông minh dựa vào cardinality:

   - Binary hoặc High cardinality (≤ 2 hoặc > 10 unique values): sử dụng **Label Encoding**
   - Low cardinality (3-10 unique values): sử dụng **One-Hot Encoding** với `drop='first'` để tránh dummy variable trap và `handle_unknown='ignore'` để xử lý giá trị mới khi inference

   Lưu encoders vào `self.encoders` để dùng cho dữ liệu mới. Với OneHot, tạo tên cột mới theo format `{tên_cột}_{category}` và xóa cột gốc. Tự động cập nhật lại column types sau encoding.

2.1.8 Chuẩn hóa dữ liệu (Normalization/Standardization)

1. **normalize:** Chuẩn hóa dữ liệu số với 3 phương pháp:

   - **standard**: StandardScaler - chuẩn hóa về mean=0, std=1 (Z-score normalization)
   - **minmax**: MinMaxScaler - scale về khoảng $[0, 1]$
   - **robust**: RobustScaler - sử dụng median và IQR, tốt cho dữ liệu có outliers vì ít bị ảnh hưởng bởi giá trị cực trị

   Fit và transform riêng từng cột numeric. Lưu tất cả scalers vào `self.scalers` để dùng cho dữ liệu mới khi inference.

2.1.9 Pipeline tự động

1. **auto_process_by_data_types:** Pipeline tự động xử lý dữ liệu hoàn chỉnh với quy trình 5 bước:

   (a) Xử lý missing values cho cột số (median/mean/mode) và cột phân loại (mode/constant)

   (b) Tạo datetime features nếu có cột datetime

   (c) Phát hiện và xử lý outliers cho numeric columns bằng `detect_outliers_smart` + `handle_all_outliers` với chiến lược clip/drop/impute

   (d) Mã hóa categorical columns tự động (Label/OneHot)

   (e) Chuẩn hóa numeric columns bằng StandardScaler

   Tất cả bước đều có thể cấu hình qua tham số. Trả về DataFrame sẵn sàng cho modeling. *Lưu ý*: chiến lược 'drop' outliers sẽ thay đổi index của data.

2.1.10 Hàm tiện ích

1. **from_dataframe (classmethod):** Tạo instance DataPreprocessor từ DataFrame có sẵn thay vì load từ file. Nhận DataFrame và tên cột target, tạo instance mới, copy DataFrame vào `data` và `original_data`, sau đó tự động phát hiện kiểu cột. Phương thức này hữu ích khi làm việc với data đã có trong memory.

2. **get_processed_data:** Trả về bản copy của DataFrame đã được xử lý. Dùng `copy()` để tránh thay đổi dữ liệu gốc ngoài ý muốn. Raise `ValueError` nếu chưa có dữ liệu.

3. **save_processed_data:** Ghi dữ liệu đã xử lý ra file với 3 định dạng hỗ trợ: 'csv' (`to_csv`), 'excel' (`to_excel`), 'json' (`to_json` với `orient='records'`). Raise `ValueError` nếu chưa có dữ liệu hoặc format không được hỗ trợ.

4. **preprocess_new_patient:** Xử lý dữ liệu bệnh nhân mới để dự đoán, sử dụng encoders và scalers đã được fit từ training data để đảm bảo consistency. Nhận dữ liệu mới dạng list (theo thứ tự columns) hoặc dict (key=tên cột). Quy trình:

   (a) Convert sang DataFrame

   (b) Apply tất cả encoders đã fit (Label/OneHot), xử lý unknown category bằng gán=0 hoặc `handle_unknown='ignore'`

   (c) Apply tất cả scalers đã fit

   (d) Thêm missing columns với giá trị 0 (scaled nếu có scaler)

   (e) Sắp xếp columns theo đúng thứ tự training data

   Raise `ValueError` nếu preprocessor chưa được fit. Trả về DataFrame 1 row sẵn sàng cho prediction.

2.2 Mô hình học máy

Mô hình học máy được thiết kế bao gồm một Class cha (BaseModel). Lớp này đóng vai trò là khung sườn (skeleton), đóng gói toàn bộ quy trình huấn luyện, kiểm thử và đánh giá mô hình học máy. Việc thiết kế này giúp chuẩn hóa quy trình, giảm thiểu việc lặp lại mã nguồn và dễ dàng mở rộng cho các thuật toán khác nhau. Các mô hình con được dùng trong bài toán này bao gồm:

1. Logistic Regression
2. SVM
3. Random Forest
4. XGBoost

2.2.1 Các hàm hỗ trợ dùng trong bài toán

1. **_to_numpy:** Hàm dùng để chuyển kiểu dữ liệu về kiểu numpy. Dùng để đồng bộ dữ liệu bài toán.
2. **_ensure_dir:** Hàm dùng để kiểm tra folder khi lưu mô hình. Nếu không tìm thấy thì hàm sẽ tự tạo folder mới để lưu trữ mô hình.

2.2.2 Các phương thức trong class BaseModel

1. **__init__:** Phương thức khởi tạo model, lưu dữ liệu X-y ban đầu, tỉ lệ train/test, seed random và tên mô hình. Nó cũng tạo các biến lưu tập train/test, mô hình ML thực tế, trạng thái đã huấn luyện hay chưa, và metadata như thời điểm tạo/huấn luyện mô hình.
2. **split_data:** Chia dữ liệu thành tập train/test. Nếu X và y không được truyền vào, hàm sẽ tự lấy từ đối tượng. Dữ liệu được chuyển về dạng numpy và chia bằng train_test_split. Kết quả được lưu vào các biến private để dùng cho training.
3. **build_model:** Là hàm trừu tượng dùng làm khung cho các lớp con (như Logistic, SVM, RF...). BaseModel không có mô hình cụ thể, nên hàm này chỉ báo lỗi nếu gọi nhầm từ lớp cha.
4. **fit:** Huấn luyện mô hình. Hàm kiểm tra X–y truyền vào, tự động chia train/test nếu chưa có, sau đó gọi build_model() để tạo mô hình thực tế. Khi mô hình _model đã sẵn sàng, nó thực hiện .fit() trên dữ liệu huấn luyện và đánh dấu mô hình đã được huấn luyện. Nếu có đưa thêm tập validation, hàm sẽ tự tính điểm và lưu lại trong metadata.
5. **predict:** Phương thức dùng để dự đoán nhãn y dựa trên dữ liệu đầu vào X sau khi mô hình đã được huấn luyện. Hàm kiểm tra mô hình đã fitted hay chưa, nếu chưa sẽ báo lỗi. Sau đó X được chuyển sang dạng numpy và mô hình nội bộ _model sẽ tạo ra dự đoán cuối cùng.

6. **predict_proba:** Phương thức trả về xác suất dự đoán của từng lớp cho dữ liệu X. Hàm này đảm bảo mô hình đã được train và hỗ trợ predict_proba(), nếu không sẽ báo lỗi. Sau khi chuyển X sang numpy, mô hình sẽ trả về ma trận xác suất, thường dùng trong bài toán phân loại nhị phân hoặc đa lớp.

7. **evaluate:** Phương thức tính toán các chỉ số đánh giá mô hình như accuracy, precision, recall, F1, và ROC-AUC (nếu mô hình hỗ trợ). Nếu không truyền X-y, phương thức sẽ dùng các thuộc tính có sẵn. y_pred sẽ được dự đoán nếu mô hình đã huấn luyện. Sau đó, các chỉ số được tính toán theo yêu cầu người dùng và trả về dưới dạng Dict. Đây là phương thức tổng hợp dùng để đánh giá toàn diện hiệu suất mô hình.

8. **optimize_params:** Phương thức tối ưu siêu tham số bằng GridSearchCV hoặc RandomizedSearchCV. Phương thức yêu cầu cung cấp đủ dữ liệu yêu cầu. Tùy vào lựa chọn của người dùng, phương thức sẽ giúp tìm ra bộ tham số tốt nhất, sau đó cập nhật lại mô hình.

9. **cross_validate:** Phương thức giúp kiểm chứng độ tin cậy của mô hình. Được xét từ tập X, y truyền vào hoặc dùng từ thuộc tính của chính các mô hình. Sau đó chuyển dữ liệu sang numpy để tính điểm trên từng fold. Cuối cùng trả về danh sách điểm từng fold cùng với điểm trung bình và độ lệch chuẩn

10. **feature_importances:** Phương thức đánh giá mức độ quan trọng của từng đặc trưng trong mô hình. Phương thức chỉ hỗ trợ nếu mô hình đã được huấn luyện và có thuộc tính feature_importances_. Kết quả trả về sẽ là Series chứa giá trị biểu thị độ quan trọng của đặc trưng. Nếu mô hình không hỗ trợ kết quả sẽ là None

11. **save_model:** Phương thức này lưu lại các thuộc tính như tên, mô hình, test_size,... bằng joblib. Việc này giúp ta tái sử dụng mô hình mà không cần phải huấn luyện lại.

12. **load_model:** Phương thức tải mô hình đã lưu từ file vào một đối tượng trong class. Sau khi tải xong ta có thể dùng lại mô hình để thực hiện các phương thức mong muốn.

13. **_format_results_for_saving:** Phương thức tạo ra kết quả là một Dict chứa các thông tin và giá trị tương ứng của mô hình. Hàm muốn lưu phải được huấn luyện

14. **save_experiment_results:** Phương thức hỗ trợ lưu thông tin về mô hình thành file csv hoặc json (do người dùng chọn). Không hỗ trợ lưu thông tin hàm chưa huấn luyện

15. **_save_to_csv / _save_to_json:** Hai phương thức hỗ trợ lưu file vào thư mục được truyền vào. Nếu thư mục không tồn tại thì tạo 1 cái

16. **is_fitted:** Phương thức kiểm tra mô hình đã được huấn luyện hay chưa. Phương thức trả về giá trị thuộc tính is_fitted.

17. **get_params:** Phương thức giúp lấy ra các tham số được dùng khi huấn luyện mô hình

18. **set_params:** Phương thức dùng để cài đặt các tham số được dùng khi huấn luyện mô hình

19. **summary/ print_summary:** Phương thức summary cho ta biết các thông tin chi tiết về mô hình, còn print_summary hỗ trợ in các thông tin ấy một cách khoa học.

20. **plot_confusion_matrix:** Phương thức vẽ confusion matrix dựa trên dữ liệu test. Nó kiểm tra mô hình đã được huấn luyện chưa, dự đoán y_pred, sau đó dùng ConfusionMatrixDisplay để vẽ. Đây là công cụ để đánh giá phân loại.

21. **plot_roc_pr:** Hàm vẽ hai biểu đồ quan trọng: ROC Curve và Precision–Recall Curve. Nó lấy điểm dự đoán (proba hoặc decision function), tính các đường cong và vẽ chúng lên hai subplot. Đây là cách trực quan để đánh giá mô hình phân loại nhị phân.

22. **_get_model_scores:** Phương thưc nội bộ trả về score dự đoán cho positive class. Nếu model hỗ trợ predict_proba, nó trả về xác suất cột 1; nếu hỗ trợ decision_function, nó trả về giá trị decision. Nếu không hỗ trợ cả hai thì báo lỗi. Đây là phương thức dùng cho ROC và PR.

2.2.3 Các phương thức dùng trong class con

Vì đa số các phương thức đã được định nghĩa trong class cha. Nên ở class con là các mô hình cụ thể. Chúng đều có chung các phương thức sau:

1. **__init__:** Kế thừa các thuộc tính từ lớp cha và bổ sung các tham số cần thiết để huấn luyện mô hình.

2. **build_model:** Phương thức này để xây dựng mô hình cụ thể.

3. **plot_feature_important:** Phương thức giúp vẽ đồ thị biểu diễn độ quan trọng của từng đặc trưng đối với nhãn. các giá trị có thể được tìm thông qua phương thức feature_importances ở trên. Nếu mô hình không được hỗ trợ, ta truy cập thủ công vào thuộc tính model để tìm giá trị cần thiết.

2.2.4 Class ModelSelector

Class với mục đích chọn ra mô hình tốt nhất theo chỉ số truyền vào và so sánh các mô hình. Đây là một class độc lập, không kế thừa từ BaseModel. Các phương thức có trong mô hình:

1. **__init__:** Phương thức khởi tạo các thuộc tính cần thiết gồm danh sách mô hình cần so sánh, chỉ số người dùng muốn so sánh, bảng kết quả và mô hình tốt nhất.

2. **check_fitted:** Phương thức kiểm tra xem mô hình đã được huấn luyện hay chưa.

3. **summary:** Phương thức thực hiện tính các chỉ số được yêu cầu và trả về danh sách các mô hình cùng với chỉ số mong muốn đã được sắp xếp. Các mô hình phải được huấn luyện trước và có chỉ số người dùng yêu cầu.

4. **set_best_model:** Phương thức dùng để cài mô hình tốt nhất vào lớp. Phương thức có hỗ trợ dùng phương thức summary nếu chưa được dùng trước đó.

5. **get_metric / get_best_model**: Các phương thức dùng để get thuộc tính thay vì truy cập trực tiếp.

6. **print_summary:** Phương thức hỗ trợ in ra kết quả của phương thức summary và set_best_model.

2.2.5 Thư viện hỗ trợ mô hình máy học

1. **Mô hình Logistic Regression (Hồi quy Logistic)**

   - Là mô hình cơ sở (baseline) cho các bài toán phân loại, dùng hàm sigmoid/softmax để mô hình hóa xác suất một điểm về một lớp cụ thể.
   - Đặc điểm:
     + Rất nhanh.
     + Phù hợp với dữ liệu vừa và nhỏ.
     + Hoạt động tốt với dữ liệu được chuẩn hóa.
   - Nguồn: Thư viện sklearn.linear_model.

2. **Mô hình SVC (Support Vector Classification)**

   - Thuật toán này hoạt động bằng cách tìm một siêu phẳng (hyperplane) tối ưu trong không gian nhiều chiều để phân tách các lớp dữ liệu với lề (margin) lớn nhất.
   - Đặc điểm:
     + Hoạt động tốt với dữ liệu nhiều chiều, phi tuyến.
     + Tham số ảnh hưởng lớn đến kết quả.
     + Cần chuẩn hóa dữ liệu khi huấn luyện mô hình.
   - Nguồn: Thư viện sklearn.svm.

3. **Mô hình Random Forest Classifier**

   - Mô hình xây dựng nhiều cây quyết định (Decision Trees) trong quá trình huấn luyện và đưa ra kết quả cuối cùng dựa trên cơ chế bỏ phiếu (voting) từ các cây con.
   - Đặc điểm:
     + : Tham số dễ hiểu.
     + : Hoạt động tốt mà không cần chuẩn hóa dữ liệu.
     + : Ổn định, phù hợp với dữ liệu vừa.
   - Nguồn: Thư viện sklearn.ensemble.

4. **Mô hình XGBClassifier**

   - Khác với Random Forest xây dựng các cây song song, XGBoost xây dựng các cây tuần tự (sequential), trong đó mỗi cây mới được tạo ra để sửa lỗi cho cây trước đó.

- Đặc điểm:
  + Mô hình hạn chế overfitting tốt
  + Hiệu suất cao
  + Linh hoạt(có thể xử lý dữ liệu thiếu)
- Nguồn: Thư viện xgboost

2.2.6 Thư viện hỗ trợ tối ưu tham số

1. **Kỹ thuật GridSearchCV (Tìm kiếm lưới)**
   - Thuật toán sẽ thử nghiệm tất cả các tổ hợp tham số có thể có trong một "lưới"(grid) do người dùng định nghĩa trước để tìm ra bộ tham số mang lại hiệu năng cao nhất.
   - Đặc điểm:
     + Đảm bảo tìm ra kết quả tối ưu nhất.
     + Phù hợp khi tham số cần tối ưu ít.
   - Nguồn: thư viện sklearn.model_selection.
2. **Kỹ thuật RandomizedSearchCV**
   - Thay vì thử tất cả, phương pháp này thực hiện lấy mẫu ngẫu nhiên một số lượng cố định các tổ hợp tham số từ các phân phối giá trị cho trước.
   - Đặc điểm:
     + Thời gian xử lý nhanh
     + Phù hợp với không gian tham số lớn
     + Kết quả trả về là "đủ tốt" thay vì tốt nhất
   - Nguồn: Thư viện sklearn.model_selection.

2.3 Cấu trúc và luồng xử lý của tệp main.py

Tệp tin `main.py` đóng vai trò là điểm khởi chạy chính của chương trình, điều phối luồng dữ liệu từ khâu cấu hình đến dự đoán. Quy trình xử lý được thực hiện tuần tự qua các bước sau:

1. **Thiết lập môi trường hiển thị:** Sử dụng `warnings.filterwarnings('ignore')` và `logging.disable(logging.CRITICAL)` để loại bỏ các cảnh báo hệ thống không cần thiết, giúp kết quả in ra màn hình (console) gọn gàng và dễ theo dõi.
2. **Tải cấu hình hệ thống:** Hàm `load_config` được gọi để thiết lập các tham số chạy. Hàm này hỗ trợ cơ chế tự động: sử dụng cấu hình mặc định (default config) hoặc nhận đầu vào tùy chỉnh (input) từ người dùng nếu có.
3. **Khởi tạo dữ liệu:** Dữ liệu được khởi tạo và xử lý bước đầu dựa trên các tham số cấu hình đã đọc ở bước trước.
4. **Thực thi Pipeline Tiền xử lý:** Hệ thống chạy Pipeline Tiền xử lý Tự động (Automated Preprocessing Pipeline) với các tham số đã thiết lập để làm sạch và chuẩn hóa dữ liệu.

5. **Huấn luyện và lưu trữ mô hình:** Các mô hình được chạy trên tập dữ liệu đã qua xử lý. Sau khi hoàn tất, mô hình được lưu trữ cục bộ dưới dạng tệp `.pkl` để phục vụ việc tái sử dụng.

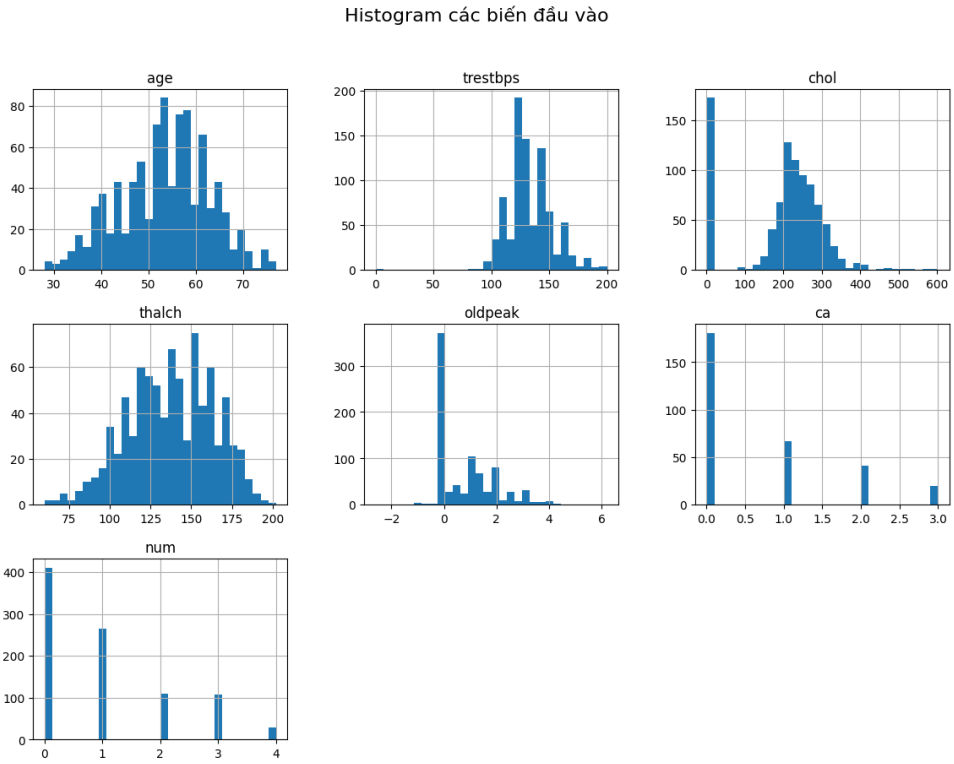
6. **Lựa chọn mô hình tối ưu:** Sử dụng lớp `ModelSelector` để so sánh và chọn ra mô hình tốt nhất dựa trên độ đo (metric) đánh giá. Giá trị mặc định được sử dụng là `recall`.

7. **Dự đoán:** Sử dụng phương thức `predict` của mô hình tốt nhất vừa chọn để thực hiện dự đoán trên dữ liệu của bệnh nhân.

3 Phân tích kết quả dựa trên plot sau khi chạy model

3.1 Trực quan hóa dữ liệu đầu vào (Input Data Visualization)

3.1.1 Histogram cho các biến numeric



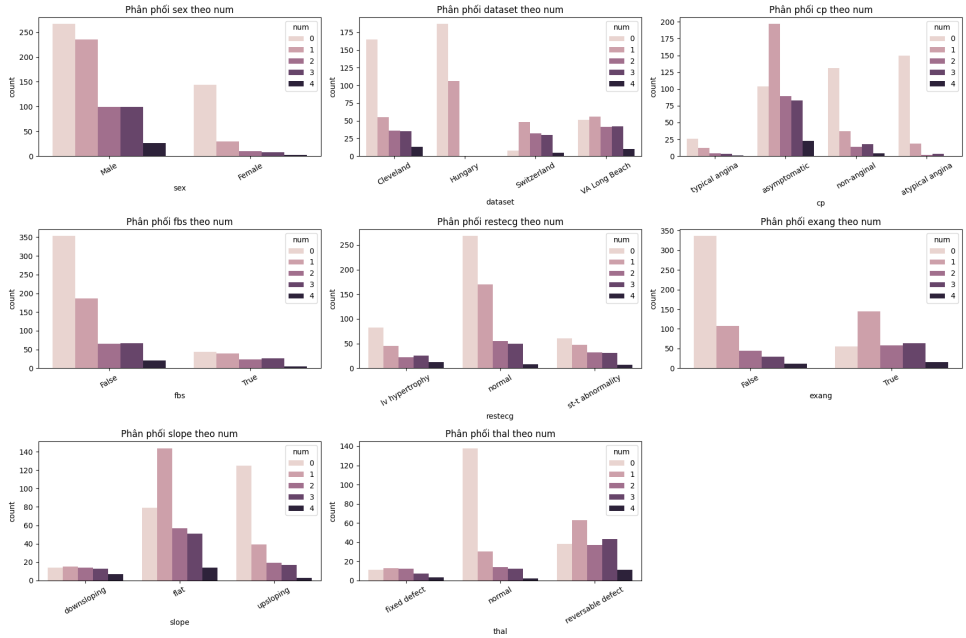
Hình 1: Histogram cho các biến numeric

Biến hiển thị: age, trestbps, chol, thalch (max heart rate), oldpeak, ca, num (target).
Những điểm quan trọng rút ra:

- **age:** Phân bố gần Gaussian, trung bình $\sim 50-60$; có một số người cao tuổi (60+).
- **trestbps (huyết áp khi nghỉ):** Phân bố dồn vào khoảng $110-160$; một số giá trị cao tới ~ 180.
- **chol:** Phân bố lệch phải, có nhiều giá trị NA/0 (một cột có cột bar tại 0 — tức dữ liệu cholesterol có giá trị 0 hoặc missing được giữ lại). Điều này cần xử lý vì 0 không hợp lý cho cholesterol.
- **oldpeak (ST depression):** Có spike lớn ở 0 (nhiều bệnh nhân không có ST depression) và dải giá trị dương nhỏ $(0-4)$.
- **ca (số mạch bị nhuộm):** Có các giá trị rời rạc $\{0,1,2,3\}$ — biểu đồ cho thấy nhiều giá trị 0.
- **num (target):** Hiển thị dưới dạng $0,1,2,3,4$ — dataset gốc có nhiều lớp; ở preprocessing bạn có thể đã chuyển thành binary (≥ 1 là bệnh). Histogram cho thấy phân phối theo lớp gốc.

Ý nghĩa tác vụ: Những histogram này cho biết cần chuẩn hoá/scale với các biến lệch (chol, oldpeak), cần xử lý 0/NA ở chol, và ca cần được xử lý như categorical/discrete.

3.1.2 Countplots theo num cho nhiều biến categorical



Hình 2: Countplots theo num cho nhiều biến categorical

Các plot: Phân phối theo num cho sex, dataset, cp, fbs, restecg, exang, slope, thal.
Những phát hiện cụ thể:

- **sex vs num:** Nam (Male) nhiều hơn nữ, và tỷ lệ num > 0 (các lớp bệnh) ở nam cũng cao — tức dataset bị lệch giới tính.
- **dataset:** Có 4 nguồn dữ liệu (Cleveland, Hungary, Switzerland, VA Long Beach) — Cleveland và Hungary là nguồn lớn; distribution khác nhau giữa các nguồn (một nguồn có nhiều bệnh hơn). *Quan trọng:* dataset là feature có ảnh hưởng lớn (ta sẽ thấy ở hệ số).
- **cp (chest pain):** Giá trị asymptomatic và non-anginal và atypical phân bố khác nhau theo num. Ở hình, asymptomatic nhiều bệnh hơn so với typical angina.
- **fbs (fasting blood sugar):** Phần lớn là False; số bệnh nhân True ít hơn.
- **restecg & exang & slope & thal:** Các category như normal, reversable defect, exang True đều cho thấy khác biệt rõ rệt giữa các giá trị num. Ví dụ thal = normal nhiều ở lớp 0 nhưng reversable defect tăng dần theo các lớp bệnh.

Ý nghĩa tác vụ: Những plot này giúp thấy các biến categorical phân bố khác nhau giữa các lớp — tức là có tín hiệu phân lớp trong categorical features. Đồng thời nhắc ta phải xử lý dummy encoding / one-hot đúng cách (vì dataset cực kỳ khác biệt).

3.1.3 Heatmap tương quan numeric (Ma trận tương quan)

Ma trận hiển thị hệ số tương quan Pearson giữa các biến numeric: age, trestbps, chol, thalach, oldpeak, ca, num.

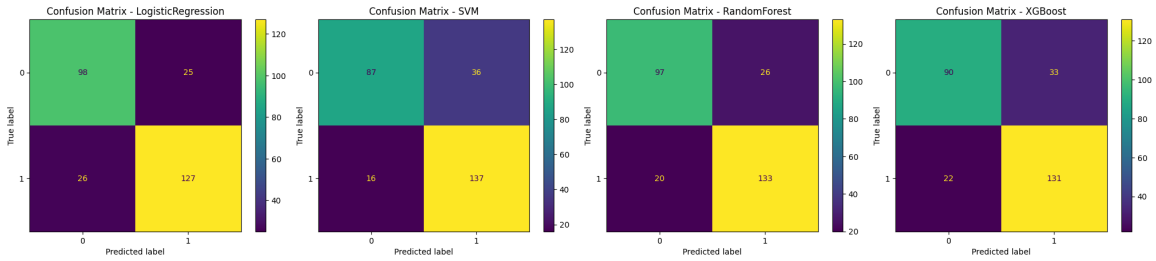
Các số nổi bật:

- **num (target)** tương quan dương đáng kể với **ca** (0.52) và **oldpeak** (0.44) → tức số mạch bị nhuộm và depression ST liên quan mạnh tới tình trạng bệnh tim.
- **num** tương quan âm với **thalch** (~ -0.37) (max heart rate) → heart rate tối đa thấp hơn liên quan tới bệnh.
- **age** có tương quan dương nhẹ với **num** (~ 0.34) → tuổi càng cao nguy cơ càng lớn.
- Một vài cặp khác: **chol** không tương quan mạnh với target ($\sim 0.05 - 0.06$), nghĩa là cholesterol đơn độc không quá mạnh ở dataset này.

Ý nghĩa: ca, oldpeak, thalch là các numeric predictor mạnh; có thể dùng làm features chính hoặc kiểm tra đa cộng tuyến (nhưng trên heatmap không thấy một cặp numeric có tương quan quá lớn với nhau để lo đa cộng tuyến nghiêm trọng).

3.2 Vẽ plot và phân tích kết quả nhận được khi chạy model

3.2.1 Phân tích confusion matrix



Hình 3: Confusion matrix của các model

Dựa trên Ma trận nhầm lẫn (Confusion Matrix), chúng ta có bảng tổng hợp các chỉ số cơ bản (TP, TN, FP, FN) cho bốn mô hình như sau:

Bảng 2: Tổng hợp thông số từ Confusion Matrix của các mô hình

Mô hình	TN	FP	FN	TP
Random Forest	97	26	20	133
Logistic Regression	**98**	**25**	26	127
SVM	87	36	**16**	**137**
XGBoost	90	33	22	131

Dưới đây là đánh giá chi tiết cho từng mô hình:

Random Forest
Mô hình Random Forest thể hiện hiệu suất **cân bằng và ổn định nhất** trong bốn thuật toán được thử nghiệm.

- Số lượng dự đoán đúng ở cả hai nhãn (TN = 97, TP = 133) đều ở mức cao.
- Tỷ lệ lỗi (FP và FN) được kiểm soát đồng đều. Điều này dẫn đến việc mô hình có chỉ số F1-Score tốt, phù hợp cho bài toán yêu cầu sự hài hòa giữa độ chính xác (Precision) và độ phủ (Recall).

Logistic Regression
Logistic Regression cho thấy xu hướng **thận trọng** trong việc dự báo nhãn dương tính (Positive).

- **Ưu điểm:** Mô hình đạt giá trị TN cao nhất (98) và FP thấp nhất (25). Điều này cho thấy mô hình có độ đặc hiệu (Specificity) cao, rất tốt trong việc giảm thiểu tỷ lệ báo động giả (False Alarm).
- **Nhược điểm:** Sự thận trọng này dẫn đến việc bỏ sót nhiều mẫu dương tính thực tế, thể hiện qua chỉ số FN cao nhất (26 trường hợp). Mô hình này không phù hợp nếu chi phí cho việc bỏ sót lỗi (Type II Error) là quá lớn.

Support Vector Machine (SVM)
Trái ngược với Logistic Regression, SVM thể hiện khả năng **nhạy bén cao** với nhãn dương tính.

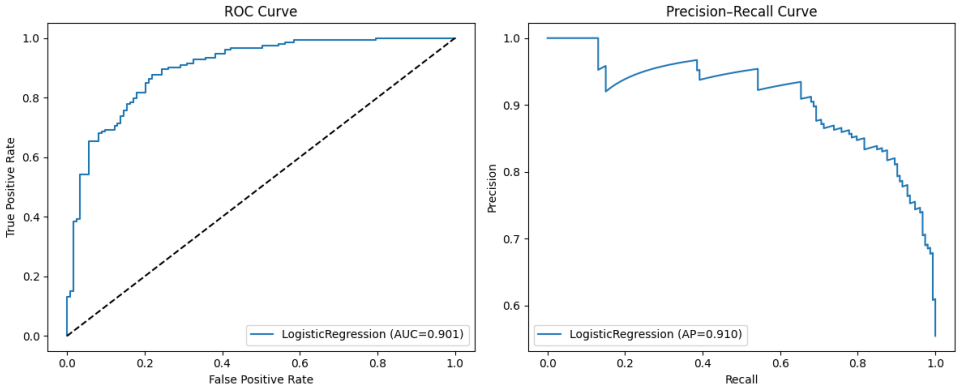
- **Ưu điểm:** SVM đạt TP cao nhất (137) và FN thấp nhất (chỉ 16). Đây là mô hình tối ưu nhất về chỉ số Recall (Độ nhạy), phù hợp cho các bài toán yêu cầu không được bỏ sót đối tượng mục tiêu (ví dụ: chẩn đoán bệnh, phát hiện gian lận).
- **Nhược điểm:** Đổi lại, mô hình chấp nhận đánh đổi bằng tỷ lệ báo dương tính giả (FP) cao nhất (36 trường hợp).

XGBoost
Trong thử nghiệm này, XGBoost chưa thể hiện được ưu thế vượt trội so với các mô hình cổ điển khác.

- Các chỉ số TP và TN của XGBoost đều thấp hơn so với mô hình tốt nhất ở từng hạng mục (thua SVM về TP và thua Logistic Regression về TN).
- Với $FP = 33$ và $FN = 22$, mô hình có tỷ lệ lỗi tổng thể cao hơn Random Forest. Kết quả này gợi ý rằng mô hình có thể cần được tinh chỉnh thêm về tham số (Hyperparameter tuning) để đạt hiệu suất tối ưu hơn.

3.2.2 Đánh giá chi tiết hiệu suất các mô hình

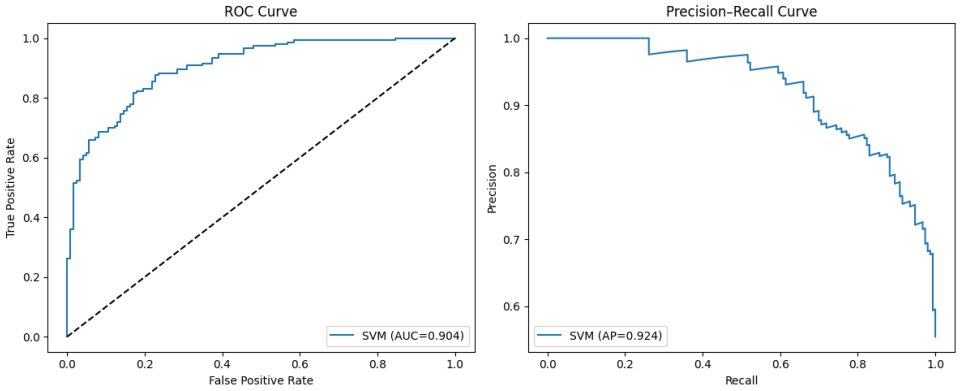
Phân tích chi tiết dựa trên biểu đồ ROC (Receiver Operating Characteristic) và Precision-Recall (PR) cho bốn mô hình: Logistic Regression, SVM, Random Forest và XGBoost.



Hình 4: ROC Curve + Precision-Recall Curve của Logistic Regression

Logistic Regression

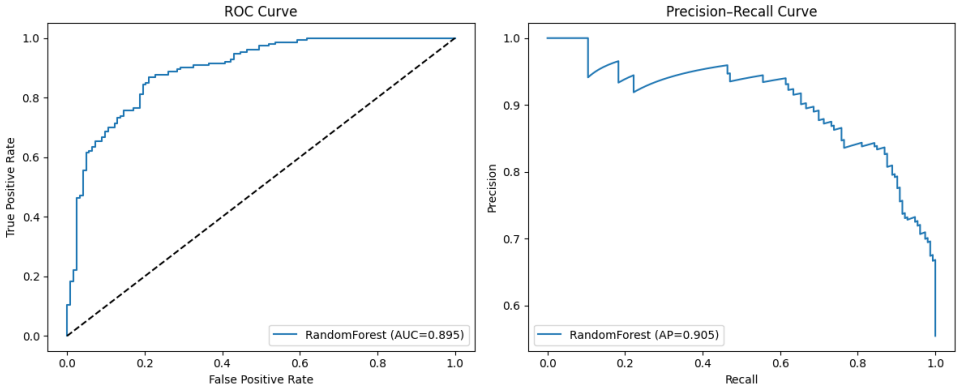
- **ROC Curve:** Mô hình đạt diện tích dưới đường cong (AUC) là 0.901. Đường cong tiệm cận tốt về góc trên bên trái, cho thấy khả năng phân loại tốt giữa các lớp dương và âm với tỷ lệ dương tính thật (TPR) cao ngay cả khi tỷ lệ dương tính giả (FPR) thấp.
- **Precision-Recall Curve:** Độ chính xác trung bình (Average Precision - AP) đạt 0.910. Đường cong giữ được độ chính xác (Precision) cao ở mức 1.0 cho đến khi độ phủ (Recall) đạt khoảng 0.2, sau đó giảm dần nhưng vẫn duy trì sự ổn định. Đây là một kết quả rất tốt cho các bài toán mất cân bằng dữ liệu.



Hình 5: ROC Curve + Precision-Recall Curve của SVM

Support Vector Machine (SVM)

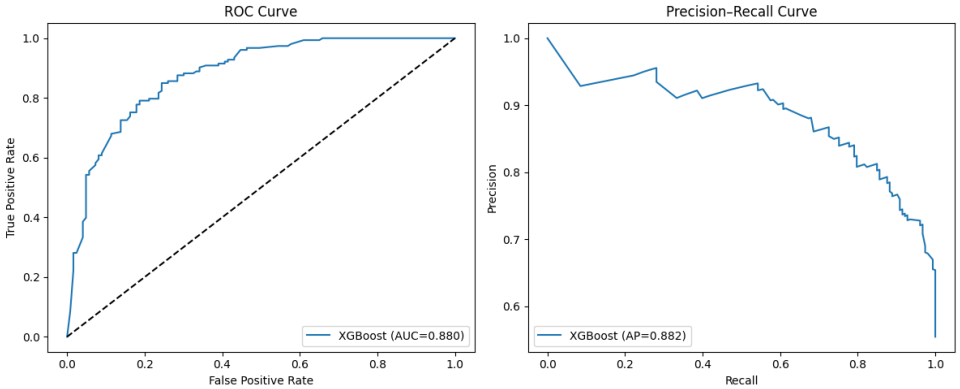
- **ROC Curve:** SVM đạt chỉ số AUC = 0.904, đây là chỉ số cao nhất trong số 4 mô hình được thử nghiệm. Điều này chỉ ra rằng SVM có khả năng tổng quát hóa và phân tách ranh giới quyết định tốt nhất.
- **Precision-Recall Curve:** Chỉ số AP = 0.924 cũng là cao nhất trong các mô hình. Đường cong PR của SVM bao phủ diện tích lớn nhất phía trên bên phải, chứng tỏ mô hình duy trì được độ chính xác rất cao ngay cả khi độ phủ tăng lên.



Hình 6: ROC Curve + Precision-Recall Curve của Random Forest

Random Forest

- **ROC Curve:** Mô hình đạt AUC = 0.895. Mặc dù thấp hơn một chút so với SVM và Logistic Regression, nhưng đường cong vẫn cho thấy hiệu suất mạnh mẽ. Đường cong có dạng bậc thang đặc trưng của các mô hình dựa trên cây quyết định.
- **Precision-Recall Curve:** Đạt AP = 0.905. Tại các ngưỡng Recall cao (> 0.8), Precision của Random Forest có xu hướng giảm nhanh hơn so với SVM, nhưng nhìn chung mô hình vẫn hoạt động ổn định.



Hình 7: ROC Curve + Precision-Recall Curve của XGBoost

XGBoost

- **ROC Curve:** XGBoost đạt AUC = 0.880, thấp nhất trong 4 mô hình. Đường cong không tiệm cận góc trên trái tốt như SVM hay Logistic Regression.
- **Precision-Recall Curve:** Chỉ số AP = 0.882. Biểu đồ cho thấy sự sụt giảm đáng kể về Precision khi Recall tăng lên mức 0.9. Kết quả này gợi ý rằng mô hình XGBoost có thể cần được tinh chỉnh thêm các siêu tham số (hyperparameters) hoặc xử lý dữ liệu kỹ hơn để cải thiện hiệu suất so với các mô hình khác.

Tổng kết so sánh

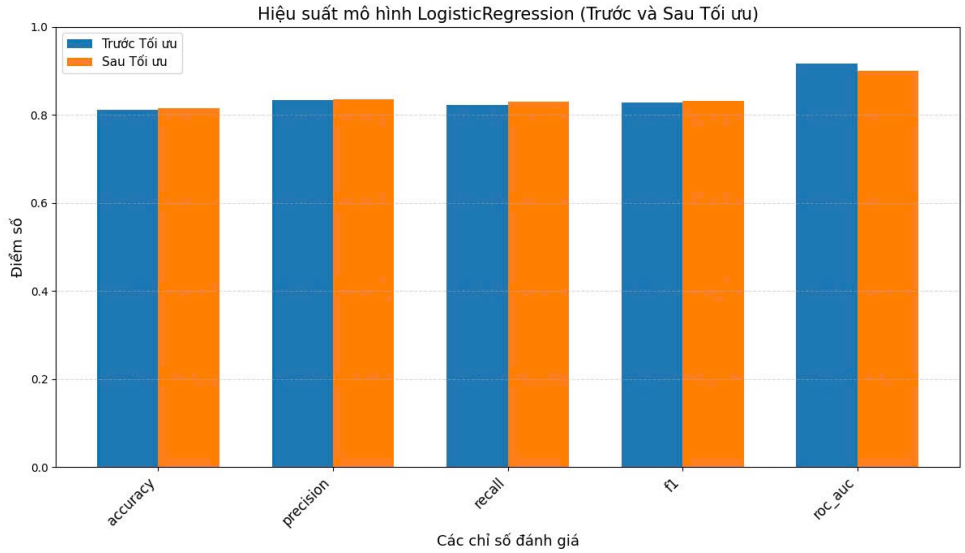
Dựa trên các chỉ số AUC và AP, thứ tự xếp hạng hiệu suất (từ cao xuống thấp) như sau:

1. **SVM:** Tốt nhất (AUC = 0.904, AP = 0.924).
2. **Logistic Regression:** Ổn định (AUC = 0.901, AP = 0.910).
3. **Random Forest:** Khá (AUC = 0.895, AP = 0.905).
4. **XGBoost:** Cần cải thiện (AUC = 0.880, AP = 0.882).

3.2.3 So sánh mô hình tối ưu siêu tham số

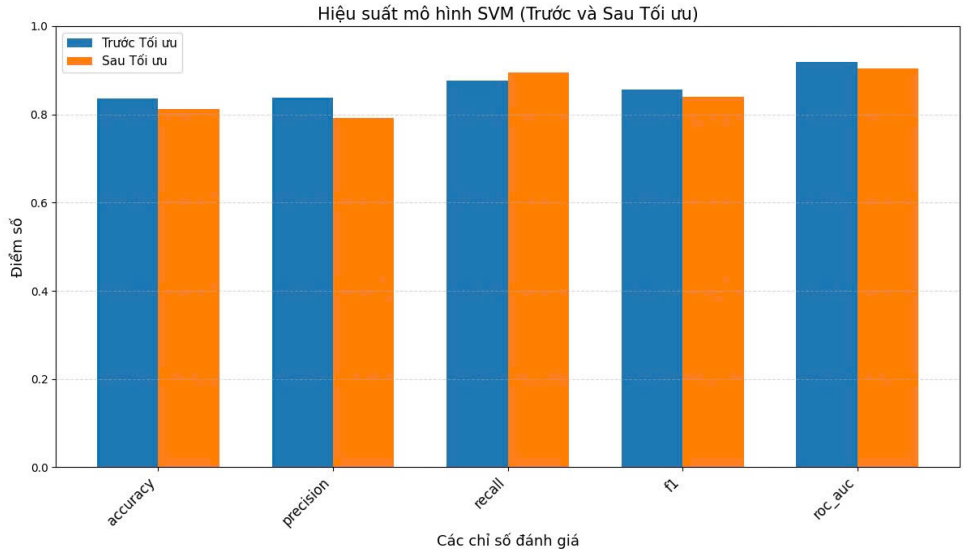
Các biểu đồ ở phần 3.2.1 thuộc các mô hình đã tối ưu siêu tham số theo scoring là **Recall**. Do đó ta cần so sánh thử xem ảnh hưởng của chúng đối với các chỉ số còn lại. Dưới đây là đồ thị so sánh giữa tham số tối ưu và tham số thường dùng với mô hình:

- **Logistic Regression:** Sau khi tối ưu, chỉ số Recall tăng nhẹ. Tuy nhiên, Roc_auc lại giảm đáng kể



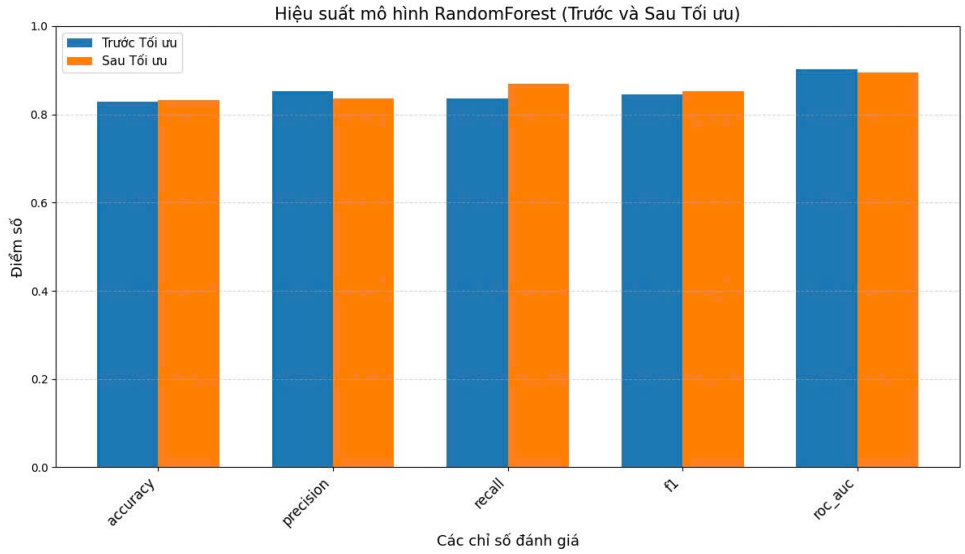
Hình 8: Mô hình Logistic Regression

- **SVM:** Sau khi tối ưu, chỉ số Recall tăng đáng kể. Bù lại thì 4 chỉ số đều giảm sâu.



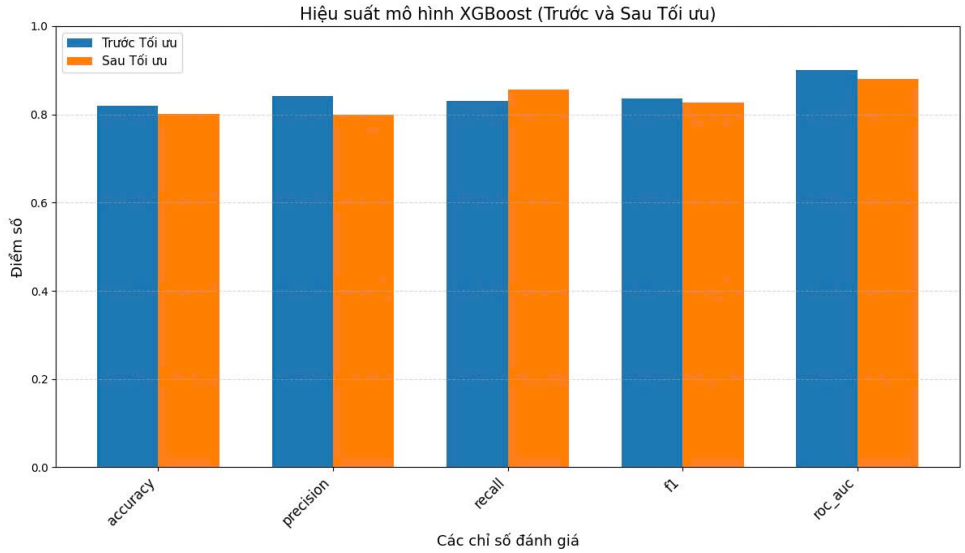
Hình 9: Mô hình SVM

- **Random Forest:** Chỉ số Recall tăng mạnh khi tối ưu. Các chỉ số còn lại chỉ có sự chênh lệch nhẹ



Hình 10: Mô hình Random Forest

- **XGBoost:** Chỉ số Recall tăng đáng kể sau khi tối ưu. Nhưng tương tự như SVM, các chỉ số còn lại đều giảm so với bình thường.



Hình 11: Mô hình XGBoost

Nhận xét:

- Ta có thể thấy ngoại trừ Random Forest. Các mô hình còn lại đều phải đánh đổi lớn để có chỉ số recall tốt hơn. Điều này là dễ hiểu vì các tham số thường dùng sẽ hướng mô hình đến chỉ số tốt nhất ở cả 5 chỉ số.

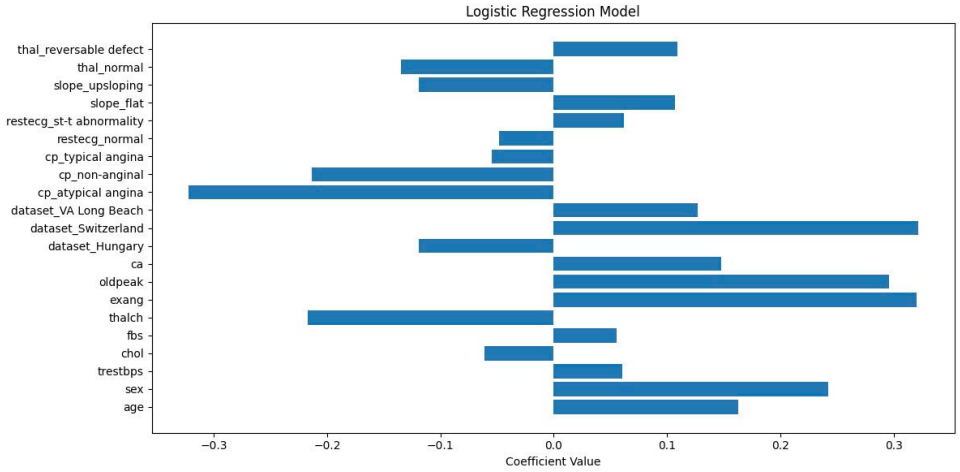
- Tuy nhiên với tính chất bài toán là dự đoán bệnh của bệnh nhân, ta cần phải tìm cách tối thiểu hóa trường hợp **Âm tính giả(FN)** để tránh bỏ sót các bệnh nhân thực sự mắc bệnh. Do đó ta phải ưu tiên chỉ số Recall hơn mặc dù việc tối ưu tham số này ảnh hưởng xấu đến các chỉ số còn lại.

- Một vài mô hình với việc tối ưu dẫn đến đa phần chỉ số bị thấp hơn, ta có thể xem xét mở rộng tổ hợp bộ tham số truyền vào.

3.2.4 Đồ thị biểu hiện feature importances

Dồ thị biểu diễn mức độ quan trọng của các đặc trưng (Feature Importances) đối với quá trình dự đoán của mô hình. Các thanh càng dài (giá trị càng lớn) thể hiện đặc trưng đó đóng góp càng nhiều thông tin giúp mô hình phân loại chính xác nhãn mục tiêu. Những đặc trưng có giá trị thấp đóng vai trò ít quan trọng hơn hoặc có thể được coi là nhiễu. Dưới đây là chi tiết các đồ thị theo từng mô hình:

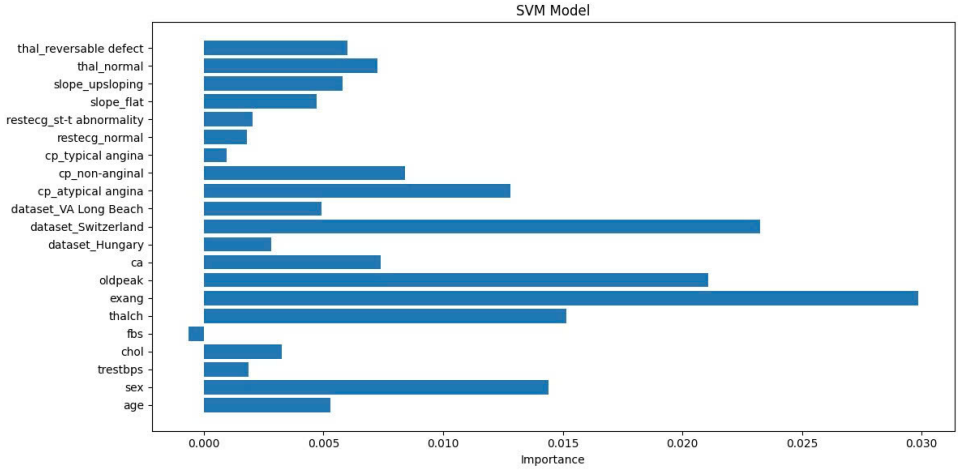
- **Logistic Regression:**



Hình 12: Feature importances cho mô hình Logistic Regression

- Đây là mô hình tuyến tính nên feature importances là các trọng số w. nếu trọng số $w < 0$, thì thuộc tính này có xu hướng đẩy nhãn về 0. Ngược lại, nếu $w > 0$, thuộc tính này có xu hướng tăng xác suất nhãn.
- Từ đồ thị ta có thể thấy cái đặc trưng như **exang, oldpeak, dataset_Switzerland** có trọng số dương cao. Điều này cho thấy các bệnh nhân đau thắt ngực khi vận động, hoặc các bệnh nhân có chênh lệch ST lớn khi vận động, hoặc bệnh nhân ở thụy Sỹ có nguy cơ mắc bệnh cao hơn
- Ngược lại các trọng số âm lớn bao gồm **thalch, cp**. Điều này cho thấy nhịp tim tối đa đạt được cao là dấu hiệu cho tim khỏe. Ngoài ra, **cp** biểu thị cho loại đau ngực có trọng số thấp vì ảnh hưởng từ đặc trưng **exang** cùng biểu thị sự đau thắt ngực

- **SVM**



Hình 13: Feature importances cho mô hình SVM

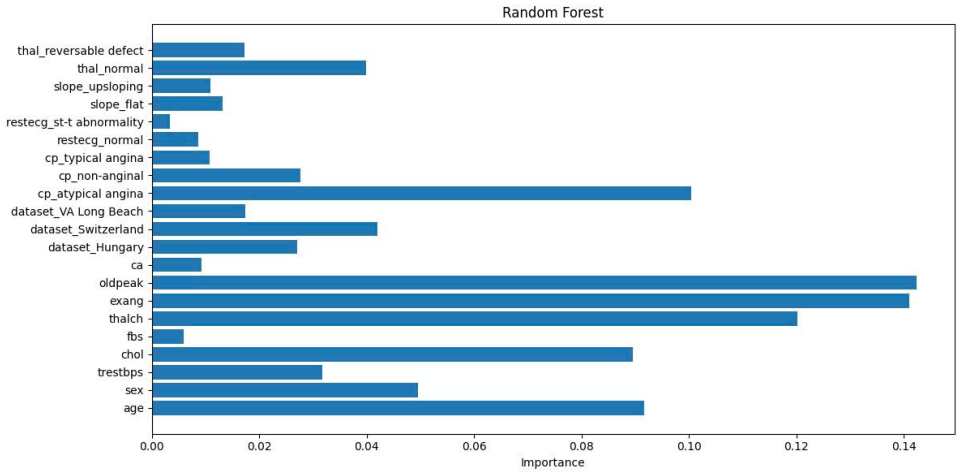
- Đây là mô hình sử dụng kernel phi tuyến nên, feature importances sẽ biểu thị theo mức độ ảnh hưởng(Permutation Importance) của từng đặc trưng. Giá trị càng cao thì ảnh hưởng càng lớn.

– Đặc trưng **fbs** mang giá trị âm và độ lớn nhỏ. Ta có thể xem đây là một đặc trưng gây nhiễu cho mô hình.

– Các đặc trưng có ảnh hưởng lớn nhất đến kết quả là **exang, dataset_Switzerland, oldpeak**. Tuy nhiên ta không thể biết ảnh hưởng của những đặc trưng này là ảnh hưởng tích cực hay tiêu cực nên ta phải phân tích thêm từ kiến thức thực tế.

– một vài đặc trưng khác có mức độ quan trọng trên trung bình như **cp, thalch, sex** cũng ảnh hưởng đáng kể đến kết quả dự đoán.

- **Random Forest**



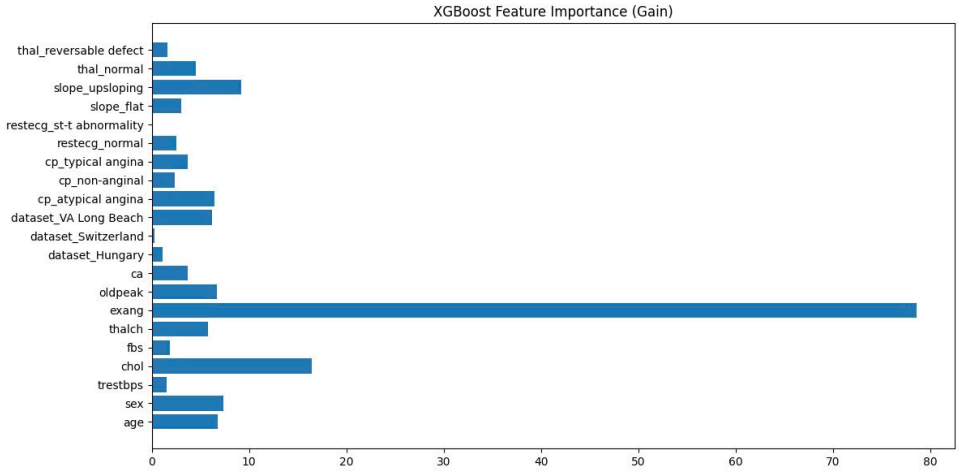
Hình 14: Feature importances cho mô hình Random Forest

– Mô hình này feature important được tính từ thuộc tính feature_importances_ trong mô hình. Các giá trị nằm trong khoảng $[0, 1]$ biểu thị mức độ ảnh hưởng quan trọng của đặc trưng đối với nhãn.

– Mô hình Random Forest có các đặc trưng quan trọng nhất **oldpeak, exang** các đặc trưng ảnh hưởng lớn nhưng thấp hơn là **cp_atypical angina, thalch**. Mô hình này là phi tuyến nên không thể biểu thị được mối tương quan với 2 nhãn.

– đặc trưng **dataset_Switzerland** không nằm trong top ảnh hưởng ở mô hình này vì Random Forest ưu tiên các chỉ số sinh học hơn. Ví dụ các bệnh nhân ở Thụy Sĩ có chỉ số **oldpeak** cao, mô hình sẽ có xu hướng giảm mức quan trọng của đặc trưng không phải chỉ số sinh học này đi.

- **XGBoost**



Hình 15: Feature importances cho mô hình XGBoost

- Đối với mô hình XGBoost, mức độ quan trọng của đặc trưng được xác định dựa trên chỉ số "Gain" (Average Gain - Lợi ích thông tin trung bình). Các giá trị biểu thị mức độ ảnh hưởng quan trọng của đặc trưng đối với nhãn.
- Đặc trưng **exang** có điểm số vượt trội. Điều này xảy ra do mô hình XGBoost tập trung tối đa vào các đặc trưng mang lại hiệu quả thực sự.
- Điều này dẫn đến hiện tượng loại bỏ các đặc trưng dư thừa tương tự **exang** như **dataset_Switzerland** bị đẩy xuống 0 như một đặc trưng nhiễu. Tuy nhiên đặc trưng này đã được **exang** giải thích rồi nên mô hình sẽ loại bỏ ra khỏi quá trình tính điểm.

- **Nhận xét:**
  - Mô hình Logistics Regression là mô hình duy nhất biểu thị được mối tương quan của đặc trưng đối với từng nhãn.
  - Hai mô hình Random Forest và XGBoost cho thấy khả năng kháng nhiễu và chọn lọc đặc trưng tốt hơn.

Phân công công việc

Bảng 3: Bảng phân công nhiệm vụ

Họ và Tên	**MSSV**	**Nội dung công việc**
Đỗ Lê Nguyên Đan	23280044	Viết main.py, notebook, sửa code đồng thời thêm các hàm vẽ plot và phân tích dữ liệu kết quả
Lương Lê Công Hạnh	23280057	Thực hiện phần tiền xử lý dữ liệu và hệ thống logging
Âu Dương Khả	23280063	Viết các Class liên quan về model, vẽ đồ thị so sánh trước và sau tối ưu siêu tham số