BỘ NÔNG NGHIỆP VÀ PHÁT TRIỂN NÔNG THÔN

PHÂN HIỆU TRƯỜNG ĐẠI HỌC THỦY LỢI

BỘ MÔN CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN MÔN HỌC HỌC MÁY

Tên đề tài:

PHÂN TÍCH DỰ ĐOÁN
TRANSACTIONS FEATURE ENGINEERED

Giảng viên hướng dẫn:	*ThS. Vũ Thị Hạnh*
Sinh viên thực hiện:	*Nguyễn Huỳnh Anh Tuấn* *Trần Tất Phát* *Đỗ Xuân Hương*
Mssv:	*2351067119* *2351067106* *2351067096*
Lớp:	*S26-65CNTT*

2025

LỜI CẢM ƠN

Để hoàn thành được bài tiểu luận này, chúng em xin chân thành cảm ơn Ban Giám hiệu, các khoa, phòng và quý thầy, cô Phân hiệu Đại Học Thủy Lợi , những người đã tận tình giúp đỡ và tạo điều kiện cho chúng em trong quá trình học tập. Đặc biệt, chúng em xin gửi lời cảm ơn sâu sắc đến cô Vũ Thị Hạnh - người đã trực tiếp giảng dạy và hướng dẫn em thực hiện bài tiểu luận này bằng tất cả lòng nhiệt tình và sự quan tâm sâu sắc.

Trong quá trình thực hiện bài tiểu luận này, do hiểu biết còn nhiều hạn chế nên bài làm khó tránh khỏi những thiếu sót. Chúng em rất mong nhận được những lời góp ý của quý thầy cô để bài tiểu luận ngày càng hoàn thiện hơn.

Chúng em xin chân thành cảm ơn!

MỤC LỤC

NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

TPHCM, ngày tháng 12 năm 2025

Ký và ghi rõ họ tên

Vũ Thị Hạnh

PHẦN 1: GIỚI THIỆU ĐỀ TÀI

Trong bối cảnh chuyển đổi số mạnh mẽ hiện nay, các hệ thống giao dịch điện tử (ngân hàng số, ví điện tử, thanh toán trực tuyến…) ngày càng phát triển và đóng vai trò quan trọng trong đời sống kinh tế – xã hội. Song song với sự phát triển đó là sự gia tăng của các hành vi gian lận trong giao dịch, gây thiệt hại lớn về tài chính và làm giảm niềm tin của người dùng.

Dự án này tập trung vào phân tích dữ liệu giao dịch thông qua một tập dữ liệu đã được xử lý và trích xuất đặc trưng (feature engineering). Dữ liệu bao gồm nhiều thuộc tính quan trọng phản ánh hành vi giao dịch như đặc điểm số tiền, tần suất giao dịch, mối quan hệ giữa các giao dịch, cũng như các đặc trưng tổng hợp giúp mô hình học máy dễ dàng nhận diện các mẫu (patterns) bất thường.

Mục tiêu chính của dự án là:

1. Khám phá và phân tích dữ liệu giao dịch sau khi đã được làm sạch và xây dựng đặc trưng.
2. Ứng dụng các thuật toán học máy (đặc biệt là các mô hình phân loại như Naive Bayes, Decision Tree, Random Forest, …) để dự đoán và so sánh hiệu quả phát hiện giao dịch bất thường/gian lận.
3. Đánh giá mô hình dựa trên các chỉ số phù hợp nhằm lựa chọn phương pháp tối ưu cho bài toán thực tế.
4. Thông qua đó, dự án giúp minh họa rõ ràng quy trình xử lý dữ liệu và xây dựng mô hình học máy trong một bài toán mang tính ứng dụng cao.

PHẦN 2: MỤC TIÊU MÀ BÀI TOÁN ĐẶT RA

1. Mục tiêu và phạm vi đề tài

a) Mục tiêu của đề tài

Mục tiêu chính của đề tài là nghiên cứu và ứng dụng các kỹ thuật học máy trên tập dữ liệu Transactions Feature Engineered nhằm phân tích hành vi giao dịch và phát hiện các giao dịch bất thường/gian lận.

Cụ thể, đề tài hướng đến các mục tiêu sau:

- Khám phá và hiểu rõ cấu trúc tập dữ liệu giao dịch sau khi đã được tiền xử lý và trích xuất đặc trưng.
- Đánh giá vai trò của feature engineering trong việc nâng cao hiệu quả của các mô hình học máy.
- Xây dựng và huấn luyện các mô hình phân loại giao dịch, tiêu biểu như Naive Bayes và một số mô hình phổ biến khác.
- So sánh hiệu quả của các mô hình dựa trên các chỉ số đánh giá phù hợp (Accuracy, Precision, Recall, F1-score, …).
- Xác định mô hình phù hợp nhất cho bài toán phân loại giao dịch trên tập dữ liệu đã cho.
- Thông qua đề tài, người thực hiện có cơ hội củng cố kiến thức về xử lý dữ liệu, xây dựng mô hình học máy và đánh giá mô hình, đồng thời hiểu rõ hơn tính ứng dụng thực tế của Machine Learning trong lĩnh vực tài chính.

b) Phạm vi của đề tài :

phạm vi nghiên cứu của đề tài được giới hạn như sau:

- Dữ liệu nghiên cứu: Sử dụng tập dữ liệu Transactions Feature Engineered, là dữ liệu giao dịch đã được làm sạch và trích xuất đặc trưng, không đi sâu vào xử lý dữ liệu thô ban đầu.
- Phạm vi kỹ thuật: Tập trung vào các thuật toán học máy giám sát (Supervised Learning) cho bài toán phân loại, đặc biệt là Naive Bayes và một số mô hình cơ bản khác để so sánh. Không triển khai các mô hình học sâu (Deep Learning).
- Phạm vi nội dung: Đề tài chỉ tập trung vào việc phân tích dữ liệu và xây dựng mô hình dự đoán giao dịch, không đề cập đến việc triển khai hệ thống thực tế hoặc tích hợp vào hệ thống ngân hàng/ứng dụng thương mại.
- Phạm vi đánh giá: Việc đánh giá mô hình dựa trên tập dữ liệu sẵn có và các chỉ số thống kê, không xem xét yếu tố thời gian thực (real-time) hay dữ liệu phát sinh mới.

PHẦN 3: MÔ TẢ DỮ LIỆU VÀ CÁC BƯỚC TIỀN XỬ LÝ

3.1 Giới thiệu toàn bộ dữ liệu Transactions Feature Engineerd

Tập dữ liệu Transactions Feature Engineered là dữ liệu giao dịch tài chính đã được xử lý, làm sạch và trích xuất đặc trưng (feature engineering) nhằm phục vụ cho các bài toán phân tích dữ liệu và học máy, đặc biệt là bài toán phân loại giao dịch bất thường hoặc gian lận.

Ban đầu, dữ liệu giao dịch thô thường chứa nhiều thông tin rời rạc, nhiễu hoặc khó khai thác trực tiếp cho mô hình học máy. Do đó, tập dữ liệu này đã trải qua quá trình tiền xử lý dữ liệu và xây dựng các đặc trưng mới, giúp biểu diễn rõ hơn hành vi và đặc điểm của mỗi giao dịch.

a) Cấu trúc dữ liệu

Mỗi dòng trong tập dữ liệu tương ứng với một giao dịch, mỗi cột là một đặc trưng (feature) mô tả giao dịch đó. Các đặc trưng trong tập dữ liệu có thể được chia thành các nhóm chính như sau:

- Nhóm đặc trưng giao dịch cơ bản:Phản ánh các thông tin trực tiếp của giao dịch như giá trị giao dịch, thời điểm giao dịch, loại giao dịch hoặc các thuộc tính định lượng ban đầu.
- Nhóm đặc trưng hành vi (behavioral features):Được xây dựng dựa trên lịch sử giao dịch của người dùng hoặc tài khoản, ví dụ như:
    - Tần suất giao dịch trong một khoảng thời gian nhất định
    - Số lượng giao dịch liên tiếp
    - Mức độ biến động của số tiền giao dịch
- Nhóm đặc trưng tổng hợp (aggregated features):
    - Là các đặc trưng được tính toán từ nhiều giao dịch trước đó, giúp mô hình nhận diện các mẫu hành vi bất thường so với hành vi thông thường.
- Nhãn mục tiêu (target/label):
    - Biểu diễn trạng thái của giao dịch (ví dụ: giao dịch bình thường hoặc giao dịch gian lận), dùng làm đầu ra cho các mô hình học máy trong bài toán phân loại.

b) Đặc điểm của dữ liệu

- Dữ liệu đã được chuẩn hóa và mã hóa để phù hợp với các thuật toán học máy.
- Không còn (hoặc đã giảm thiểu) các giá trị thiếu (missing values) và dữ liệu nhiễu.
- Các đặc trưng đã được thiết kế nhằm tăng khả năng phân biệt giữa các loại giao dịch khác nhau.
- Tập dữ liệu có thể tồn tại sự mất cân bằng lớp, phản ánh đúng thực tế khi số lượng giao dịch gian lận thường ít hơn nhiều so với giao dịch hợp lệ.

c) Ý nghĩa của việc Feature Engineering

- Việc trích xuất và xây dựng đặc trưng trong tập dữ liệu Transactions Feature Engineered đóng vai trò quan trọng trong việc:
    - Nâng cao hiệu suất và độ chính xác của mô hình học máy.
    - Giúp các thuật toán như Naive Bayes, Decision Tree, Random Forest hoạt động hiệu quả hơn.
    - Làm rõ mối quan hệ tiềm ẩn giữa các giao dịch, từ đó hỗ trợ tốt hơn cho việc phát hiện bất thường và gian lận.

d) Ứng dụng của dữ liệu

- Tập dữ liệu này được sử dụng cho các mục đích:
    - Huấn luyện và đánh giá các mô hình học máy trong bài toán phân loại giao dịch.
    - So sánh hiệu quả giữa các thuật toán khác nhau.Phục vụ nghiên cứu, học tập và thực hành quy trình phân tích dữ liệu thực tế trong lĩnh vực tài chính – ngân hàng

3.2 Tiền xử lý dữ liệu

a) Môi trường và công cụ thực hiện:

Để thực hiện quá trình xây dựng và huấn luyện mô hình, nhóm chúng em sử dụng các công cụ và môi trường sau:

- Ngôn ngữ lập trình: Python.
- Môi trường phát triển tích hợp (IDE): Visual Studio Code (VS Code). Đây là trình biên tập mã nguồn phổ biến, hỗ trợ mạnh mẽ các thư viện phân tích dữ liệu và Debugging.

b) Nguồn dữ liệu:

Dữ liệu được thu thập từ nền tảng Kaggle, bộ dữ liệu có tên "E-Commerce Fraud Detection Dataset" (Phát hiện gian lận thương mại điện tử).

- Nguồn: Link Kaggle Dataset
- Mô tả sơ bộ: Ban đầu, nhóm tiến hành tải dữ liệu Transactions Feature Engineered.

c) Quy trình xử lý và làm sạch dữ liệu

Bước 1: Đọc và làm sạch dữ liệu (Data Cleaning)

- Dữ liệu được đọc vào dataframe với kích thước ban đầu ghi nhận là *(299.695 dòng, 17 cột)*.
- **Loại bỏ thuộc tính không cần thiết:** Nhóm tiến hành loại bỏ cột **'ID'**.

- *Lý do:* Đây là mã định danh ngẫu nhiên của giao dịch, không mang ý nghĩa thống kê hay phân loại và không ảnh hưởng đến kết quả dự đoán của mô hình.

Bước 2: Trích xuất đặc trưng (Feature Engineering): Để máy tính hiểu rõ hơn về ngữ cảnh thời gian và địa lý, nhóm đã thực hiện:

- **Xử lý thời gian:** Tách thông tin từ cột thời gian gốc thành các đặc trưng riêng biệt là **Giờ** và **Thứ**, sau đó xóa cột thời gian gốc.
- **Tạo đặc trưng mới:** Tạo thêm cột 'country_mismatch' để kiểm tra sự sai lệch vị trí địa lý.
  - Giá trị 1: Lệch (Địa chỉ IP và thẻ không cùng quốc gia).
  - Giá trị 0: Khớp.

Bước 3: Mã hóa dữ liệu (Encoding): Các dữ liệu dạng chữ (categorical) cần được chuyển về dạng số để mô hình có thể tính toán. Nhóm sử dụng phương pháp **One-Hot Encoding** cho toàn bộ các cột thuộc tính phân loại, bao gồm:

- 'country', 'bin_country', 'channel', 'merchant_category'.

Bước 4: Chuẩn hóa dữ liệu (Scaling): Để tránh việc các biến số có giá trị quá lớn lấn át các biến nhỏ, nhóm thực hiện chuẩn hóa dữ liệu bằng phương pháp **MinMax Scaler**.

- Phạm vi chuẩn hóa: Đưa các giá trị về đoạn [0, 1].
- Áp dụng cho các cột dữ liệu số như: Amount (Số tiền), Age (Tuổi),...

d) Source code:

https://github.com/Darrys37/Project_machine_learning

3.2 Mô tả chi tiết bộ dữ liệu (Dataset Description)

Bộ dữ liệu Transactions Feature Engineered mô tả các giao dịch thanh toán trực tuyến trong bối cảnh thực tế với vấn đề mất cân bằng dữ liệu nghiêm trọng. Mục tiêu chính là xây dựng mô hình phân loại nhị phân (Binary Classification) để phát hiện giao dịch gian lận.

Từ các thuộc tính gốc ban đầu, sau quá trình trích xuất đặc trưng và mã hóa (One-Hot Encoding), bộ dữ liệu được tổ chức thành 42 đặc trưng (features), chia làm 9 nhóm chính phục vụ cho việc huấn luyện mô hình:

1. Nhóm định danh (Identity)

- transaction_id: Mã định danh giao dịch.
- user_id: Mã định danh người dùng.

- Nhận xét: Các thuộc tính này chỉ mang ý nghĩa quản lý, không chứa quy luật phân phối nên không được sử dụng làm đầu vào cho mô hình huấn luyện.

2. Nhóm hành vi người dùng (User Behavior)

- account_age_days: Thời gian tồn tại của tài khoản (tính bằng ngày).
- total_transactions_user: Tổng số giao dịch người dùng đã thực hiện.
- avg_amount_user: Giá trị trung bình của các giao dịch.
  - Ý nghĩa: Phản ánh mức độ uy tín và thói quen chi tiêu. Tài khoản mới lập, ít giao dịch nhưng phát sinh chi tiêu đột biến là dấu hiệu rủi ro cao.

3. Thông tin giao dịch (Transaction Details)

- amount: Số tiền của giao dịch hiện tại.
- channel: Kênh thực hiện giao dịch (Web/App).
- promo_used: Trạng thái sử dụng mã khuyến mãi (1: Có, 0: Không).
  - Ý nghĩa: Hành vi gian lận thường đi kèm với số tiền bất thường hoặc lạm dụng mã giảm giá để kiểm tra tính hiệu lực của thẻ.

4. Nhóm bảo mật (Security Checks) Đây là nhóm đặc trưng quan trọng nhất trong việc phát hiện gian lận (Fraud Detection):

- avs_match: Kết quả đối khớp địa chỉ chủ thẻ (Address Verification Service).
- cvv_result: Kết quả kiểm tra mã bảo mật CVV.
- three_ds_flag: Giao dịch có sử dụng xác thực 3D Secure hay không.
  - Ý nghĩa: Các giao dịch thất bại trong kiểm tra AVS, CVV hoặc không có 3D Secure thường có xác suất gian lận rất cao.

5. Thông tin vận chuyển (Logistics)

- shipping_distance_km: Khoảng cách vận chuyển hàng hóa.
  - Ý nghĩa: Khoảng cách xa bất thường so với lịch sử hành vi của người dùng có thể là dấu hiệu đáng ngờ.

6. Nhóm thời gian (Temporal Features)

- hour: Giờ giao dịch.
- day, month: Ngày, tháng giao dịch.
- day_of_week: Thứ trong tuần.
    - Ý nghĩa: Các giao dịch gian lận thường có xu hướng tập trung vào các khung giờ thấp điểm (đêm khuya) hoặc các ngày cụ thể.

7. Nhóm loại hình kinh doanh (Merchant Category) Được mã hóa dạng One-Hot để máy học có thể xử lý:

- merchant_category_electronics (Điện tử), merchant_category_fashion (Thời trang), merchant_category_gaming (Game), merchant_category_travel (Du lịch),...
    - Ý nghĩa: Một số ngành hàng đặc thù như Gaming, Đồ điện tử, Du lịch thường là mục tiêu tấn công chính của tội phạm mạng.

8. Nhóm vị trí địa lý (Geolocation) Sử dụng mã hóa One-Hot để đối chiếu nguồn gốc:

- Quốc gia giao dịch: country_US, country_FR, country_DE...
- Quốc gia phát hành thẻ (BIN): bin_country_US, bin_country_FR...
    - Ý nghĩa: Giúp phát hiện hiện tượng Geo Mismatch (Ví dụ: Thẻ phát hành tại Mỹ nhưng giao dịch thực hiện tại Pháp), một chỉ báo rủi ro quan trọng.

9. Nhãn mục tiêu (Target)

- is_fraud: Biến mục tiêu cần dự đoán.
    - 0: Giao dịch bình thường (Legitimate).
    - 1: Giao dịch gian lận (Fraud).

3.3 Import Thư viện và Cấu hình Môi trường

MODULE 1: SETUP & INGESTION

Mô-đun này là bước khởi đầu quan trọng trong quá trình xây dựng mô hình Machine Learning cho bài toán phát hiện giao dịch nano (Phát hiện gian lận). Module thực hiện các nhiệm vụ cơ bản nhưng thiết yếu: thiết lập môi trường làm việc, import thư viện, đọc dữ liệu và tiền xử lý dữ liệu ban đầu.

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
import streamlit as st
```

Hình 1 Data xử lý thư viện và trực quan hóa

- Pandas : Thư viện chính cho việc xử lý và phân tích dưới dạng bảng dữ liệu (DataFrame). Cung cấp các công cụ mạnh mẽ để đọc, ghi, chọn bộ lọc để thay đổi dữ liệu.
- NumPy : Thư viện tính toán khoa học, cung cấp mảng đa chiều và hiệu suất học toán được phép.
  - ➢ Thư viện cơ bản để thao tác dữ liệu dạng bảng và tính toán số học.
- Matplotlib : Thư viện vẽ sơ đồ cơ bản, hoạt động cho mọi loại biểu đồ.
- Seaborn : Thư viện trực tuyến thống kê cấp cao, xây dựng trên matplotlib, cung cấp giao diện đẹp và dễ sử dụng hơn.
  - ➢ Dùng để trực quan hóa dữ liệu, rất quan trọng trong giai đoạn phân tích khám phá (EDA).
- **warnings:** Để quản lý cảnh báo, tránh làm nhiễu đầu ra.

Thư vviện	Phiên bản ước tính	Chức năng chính	Ứng dụng trong dự án
Pandas	1.3.0+	Xử lý dữ liệu dạng bảng	Đọc CSV, làm sạch, biến đổi
NumPy	1.19.0+	Tính toán số học	Xử lý mảng, toán học
Matplotlib	3.4.0+	Vẽ đồ thị 2D	Trực quan hóa kết quả
Seaborn	0.11.0+	Vẽ đồ thị thống kê	Phân tích EDA nâng cao

3.3 Thư viện Machine Learning - Scikit-learn

```
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score, StratifiedKFold
```

Hình 2 Các mô-đun cho mô hình hóa

- **Train_test_split:** Chia dữ liệu thành tập huấn luyện và kiểm tra
- **Gridsearchcv:** Tìm kiếm thông số tối ưu siêu thông qua Grid Search

- **Cross_val_score:** Đánh giá mô hình bằng Xác thực chéo
- **Stratifiedkfold:** Phân chia dữ liệu có phân tầng (giữ tỷ lệ lớp)

```
from src.model_DecisionTree_TranTatPhat import train_decision_tree
from src.model_RandomForest_DoXuanHuong import train_random_forest
from src.model_XGBoost_NguyenHuynhAnhTuan import train_xgboost
```

Hình 3 Phân loại thuật toán

- **DecisionTreeClassifier:** Quyết định mô hình
- **RandomForestClassifier:** Mô hình rừng ngẫu nhiên (cùng nhiều cây)

Bảng so sánh các mô hình:

Mô hình	Loại	Ưu điểm	Nhược điểm	Phù hợp với
Decision Tree	Cây quyết định	Dễ hiểu, không cần chuẩn hóa	Dễ overfit, nhạy với nhiễu	Baseline model
Random Forest	Ensemble	Giảm overfit, feature importance	Chậm, phức tạp	Production model

```
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc
```

Hình 4 Các công cụ đánh giá:

- **confusion_matrix:** Ma trận đấu
- **classification_report:** Báo cáo phân loại chi tiết (precision, recall, f1)
- **roc_curvevà auc:** Road cong ROC và hiển thị bên dưới đường cong
- **recall_score, f1_score:** Công cụ đo lường có thể

Xử lý mất cân bằng dữ liệu: from sklearn.utils import resample

resample: Công cụ lấy lại mẫu (oversampling/undersampling) để xử lý dữ liệu cân bằng

Ý nghĩa các metrics trong fraud detection:

Metric	Công thức	Ý nghĩa trong fraud detection	Ưu tiên
Recall	TP/(TP+FN)	Khả năng phát hiện gian lận	CAO (quan trọng nhất)
Precision	TP/(TP+FP)	Độ chính xác khi báo gian lận	Trung bình
F1-Score	2*(P*R)/(P+R)	Cân bằng Precision và Recall	Cao
AUC-ROC	Diện tích dưới đường ROC	Khả năng phân biệt tổng thể	Cao

Nhận xét:

Việc import train_test_split hai lần là không cần thiết, nhưng không gây lỗi.

Các thư viện được import khá đầy đủ cho một bài toán phân loại cơ bản.

sklearn.model_selection: Cung cấp các công cụ chia dữ liệu, tối ưu siêu tham số (GridSearchCV), đánh giá chéo (cross_val_score) và StratifiedKFold (chia dữ liệu theo tỷ lệ lớp).

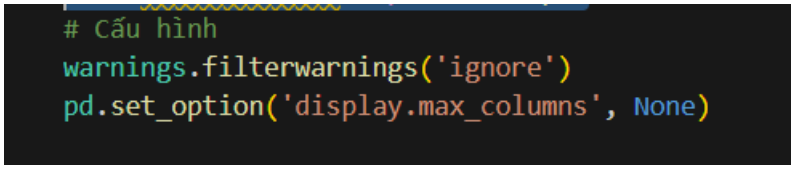
sklearn.tree: Cây quyết định, bao gồm cả hàm vẽ cây (plot_tree).

sklearn.ensemble: Rừng ngẫu nhiên (Random Forest) - một mô hình mạnh mẽ.

sklearn.naive_bayes: Hai biến thể của Naive Bayes (Gaussian cho dữ liệu liên tục, Bernoulli cho nhị phân).

sklearn.metrics: Các độ đo đánh giá mô hình: ma trận nhầm lẫn, báo cáo phân loại, đường cong ROC, diện tích dưới đường cong (AUC), recall, F1.

sklearn.utils.resample: Có thể dùng để lấy mẫu lại dữ liệu(upsampling/downsampling) xử lý mất cân bằng lớp.



Hình 5 Môi trường cấu hình

warnings.filterwarnings('ignore'): Tắt các cảnh báo không cần thiết để xuất sạch hơn,nhưng có thể làm lỡ một số cảnh báo quan trọng, nên cân nhắc.

pd.set_option('display.max_columns', None): Hiển thị tất cả các cột khi in DataFrame,hữu ích khi xem dữ liệu có nhiều cột.

3.4 Quy Trình Đi Làm Sạch Liệu Liệu

```
def load_and_clean_data(filepath):
    """
    Đọc và làm sạch dữ liệu ban đầu.
    - Xóa cột ID, thông tin cá nhân.
    - Xử lý thời gian (tạo cột hour, day_of_week).
    """
    print(f"--- [Preprocessing] Đang đọc file: {filepath} ---")
    try:
        df = pd.read_csv(filepath)
    except FileNotFoundError:
        raise FileNotFoundError(f"Không tìm thấy file tại {filepath}")
```

Hình 6 Đọc dữ liệu

Phân tích: File tên transactions_feature_engineered.csv cho bước dữ liệu được tìm thấy Feature Engineering,sử dụng dấu phân cách ;thay vì comcomma normal, phù hợp với chuẩn CSV châu Âu dữ liệu được tải vào DataFrame df để xử lý. Đọc file CSV, dấu phân cách là ;. Điều này cho thấy dữ liệu có thể được xuất từ các công cụ như Excel (định dạng châu Âu) hoặc từ một nguồn nào đó sử dụng dấu ; thay vì ,.

```
# 3. Xử lý target
if 'is_fraud' in df.columns:
    df = df.dropna(subset=['is_fraud'])
    df['is_fraud'] = df['is_fraud'].astype(int)
```

Hình 7 Kiểm tra dữ liệu kích thước và xử lý missing values

Phân tích:

- df.shapereturn tuple (số dòng, số cột)
- dropna(inplace=True): Xóa toàn bộ dòng chứa giá trị null/thiếu
- In size before va after để đánh giá số lượng dữ liệu bị loại bỏ
- In kích thước ban đầu của dữ liệu.
- Xóa tất cả các dòng có giá trị thiếu (NaN).
- In kích thước sau khi xóa để biết số dòng bị mất.
- Phương pháp này phù hợp khi:
    - thiếu dữ liệu thấp
    - Thiếu dữ liệu xảy ra ngẫu nhiên (MCAR - Thiếu hoàn toàn ngẫu nhiên)

Lưu ý: Với dữ liệu có tỷ lệ thiếu cao hoặc thiếu mẫu, cần xem xét các phương pháp quy định thay vì xóa.Việc xóa toàn bộ dòng có giá trị thiếu (dropna) là một cách làm đơn giản nhưng có thể dẫn đến mất mát dữ liệu đáng kể, đặc biệt nếu dữ liệu có nhiều giá trị thiếu. Cần phân tích kỹ lưỡng hơn về kiểu giá trị thiếu (Missing Completely at Random, Missing at Random, Missing Not at Random) và xem xét các phương pháp xử lý khác như điền giá trị (imputation) nếu có thể.

```
# 1. Xóa các cột ID không cần thiết (Giống code cũ)
ids_to_drop = ['transaction_id', 'user_id', 'trans_num', 'unix_time']
df.drop(columns=[c for c in ids_to_drop if c in df.columns], inplace=True, errors='ignore')
```

Hình 8 Loại bỏ các cột không cần thiết

Phân tích:

- transaction_idđến user_idđó có các cột định nghĩa, không có giá trị dự kiến
- Sử dụng tính năng hiểu danh sách để kiểm tra if c in df.columnsđể tránh lỗi nếu cột không tồn tại
- inplace=True: Thay đổi chuyển tiếp DataFrame gốc, tiết kiệm bộ nhớ

Lý do loại: ID cột thường là mã định danh duy nhất, không mang mẫu thông tin nếu giữ lại có thể gây ra tình trạng quá khớp (model học ID thay vì thực hiện mẫu) Việc xóa transaction_id và user_id là hợp lý vì đây là các cột định danh, thường không dùng để huấn luyện mô hình (trừ khi có ý nghĩa đặc biệt, nhưng thường thì không).Cách viết [c for c in cols_to_drop if c in df.columns] là một cách phòng thủ, tránh lỗi khi cột không tồn tại.

Kết quả sau khi chạy :

```
--- [Preprocessing] Đang đọc file: data/transactions.csv ---
- Đã xử lý thời gian: Tạo cột 'hour', 'transaction_hour', 'day_of_week'
--- [Feature Engineering] Đang tạo đặc trưng mới ---
-> Đã thêm features. Kích thước hiện tại: (299695, 22)
```

3.5 Phân tích dữ liệu khai phá (EDA)

Phân tích dữ liệu thăm dò (EDA). dựng mô hình Machine Learning. EDA không chỉ giúp kiểm tra chất lượng dữ liệu mà còn cung cấp thông tin chi tiết quan trọng để:

- Lựa chọn các tính năng phù hợp
- Xác định chiến lược xử lý cân bằng dữ liệu
- Phát hiện các ngoại lệ và dị thường
- Các mẫu hành vi này là gian lận giao dịch thông thường

- Đây là lý do tại sao bạn cần kỹ thuật tạo đặc trưng.

3.5.1 Mô-đun cấu trúc

```python
def run_eda_analysis(df, enable_streamlit=False):
    """
    Hàm thực hiện phân tích dữ liệu khám phá (EDA).
    """

    # Hàm con hỗ trợ hiển thị
    def show_fig(fig):
        if enable_streamlit:
            st.pyplot(fig)
            plt.close(fig) # Quan trọng: Xóa hình khỏi bộ nhớ sau khi vẽ
        else:
            # plt.show() # Tắt show console
            pass

    def show_text(text, header=False):
        if enable_streamlit:
            if header: st.subheader(text)
            else: st.write(text)
        else:
            print(text)
```

```python
# 0. PHÂN PHỐI NHÃN
# Dùng subplots để tạo khung hình riêng biệt hoàn toàn
fig1, ax1 = plt.subplots(figsize=(6, 4))
sns.countplot(x='is_fraud', data=df, palette=['skyblue', 'red'], ax=ax1)
ax1.set_title('Phân phối các lớp (0: Normal, 1: Fraud)')
show_fig(fig1)
```

Mục đích phân tích mục tiêu: Đây là bước đầu tiên và quan trọng nhất trong EDA để hiểu về sự mất cân bằng giai cấp - một đặc tính hữu ích của việc phát hiện gian lận bài toán.

Hình ảnh hóa Kỹ thuật:

- Countplot : Biểu đồ cột đếm số lượng mẫu cho mỗi lớp
- Phối màu : Skyblue (Bình thường) vs Đỏ (Gian lận) - tạo ra sự tương thích rõ ràng
- Kích thước hình (6, 4) : Kích thước vừa phải cho biến phân loại đơn

Thông tin thu thập:

1. Visual : Biểu đồ cho thấy sự chênh lệch giữa hai lớp
2. Định lượng : Số lượng tuyệt đối của các trường hợp gian lận
3. Percentage : Tỷ lệ gian lận trong tổng số giao dịch

Ý nghĩa vụ:

Trong thực tế, tỷ lệ nanomet thường rất thấp (0,1% - 5%):

- < 1% : Việc phát hiện gian lận dựa trên các kỹ thuật chuyên biệt.
- 1-3% : Tỷ lệ gian lận điển hình, rất mất cân bằng.
- > 5% : Bất thường, đây là vấn đề về chất lượng dữ liệu hoặc do nhà cung cấp cụ thể.

```
fraud_count = df['is_fraud'].sum()
total = len(df)
show_text(f"-> Số lượng Gian lận: {fraud_count} / {total} ({fraud_count/total:.2%})")
```

Hình 9 Kỹ thuật tính

Sử dụng .sum()cho cột nhị phân là một thủ thuật thông minh: Nhanhlen(df[df['is_fraud']==1]) gắn và dễ đọc =>Sử dụng hoạt động vector hóa

```
if 'hour' in df.columns:
    fraud_by_hour = df.groupby('hour')['is_fraud'].mean()
    tx_by_hour = df.groupby('hour')['is_fraud'].count()
```

- Mean() : Tính tỷ lệ gian lận (vì is_fraud có 0/1, Mean = tỷ lệ)
- Count() : Đếm tổng số giao dịch mỗi giờ
- Groupby : Nhóm theo thời gian từ 0-23

Phân bổ khối lượng giao dịch :Giờ cao điểm (8 giờ sáng - 10 giờ tối) : Làm việc toàn thời gian, khối lượng công việc lớn và Mùa thấp điểm (12 giờ sáng-6 giờ sáng) : Khối lượng thấp, ít hợp lý giao dịch

Những hiểu biết có thể dẫn ra:

Nếu volume cao vào ban đêm → Đáng ngờ, có thể là automated fraud

Nếu volume giảm dần từ tối → Pattern bình thường của ecommerce

3.5.2 Tỷ lệ gian lận theo giờ

Kết quả thống kê:

```
    print("Top 5 khung giờ có tỷ lệ gian lận cao nhất:")
    print(fraud_by_hour.sort_values(ascending=False).head(5))
else:
    print("Cảnh báo: Không tìm thấy cột 'hour'.")
```

Cung cấp công cụ xếp hạng có thể giúp: Xác định các giờ có rủi ro cao cụ thể,ưu tiên nguồn lực giám sát và thiết lập ngưỡng gian lận động

```
Top 5 khung giờ có tỷ lệ gian lận cao nhất:
hour
23    0.025071
1     0.024208
2     0.023928
8     0.023670
16    0.023121
Name: is_fraud, dtype: float64
```

Những hiểu biết có thể áp dụng ngay:

1. Tăng cường giám sát vào giờ rủi ro cao
2. Đặt ngưỡng nghiêm ngặt hơn cho giờ thấp điểm.
3. Triển khai mô hình mạnh mẽ dựa trên các tính năng thời gian

3.5.3 Phân tích ma trận tương quan

```python
# 2. MA TRẬN TƯƠNG QUAN
if enable_streamlit: st.markdown("### 3. Ma trận tương quan")

num_cols = df.select_dtypes(include=[np.number]).columns.tolist()
cols_to_plot = [c for c in num_cols if 'merchant_' not in c and 'country_' not in c and 'id' not in c]
if 'is_fraud' not in cols_to_plot and 'is_fraud' in df.columns: cols_to_plot.append('is_fraud')

corr = df[cols_to_plot].corr()

fig4, ax4 = plt.subplots(figsize=(10, 8))
sns.heatmap(corr, annot=True, fmt=".2f", cmap='coolwarm', linewidths=0.5, ax=ax4)
ax4.set_title('Correlation Matrix')
show_fig(fig4)
```

Mục đích phân tích: Tìm hiểu mối quan hệ tuyến tính giữa các đặc điểm và xác định các đặc điểm có mối tương quan chặt chẽ với biến mục tiêu.

Bước 1:Chọn các cột số :

num_cols = df.select_dtypes(include=[np.number]).columns.tolist()

- Chỉ lấy các cột có dtype ở đó số
- Ma trận tương quan chỉ áp dụng cho đặc trưng số

Bước 2: Lọc bỏ các cột nhiễu

cols_to_plot = [c for c in num_cols if 'merchant_' not in c and 'country_' not in c and 'id' not in c]

- Merchant_ : Danh mục được mã hóa một lần (quá nhiều, làm ma trận lớn)
- country_ : Tương tự, danh mục địa lý
- id : Xác định các cột có mối tương quan mạnh

Lý do lọc: bản đồ nhiệt với hơn 50 tính năng và dễ đọcvới tính năng nổi bật có mô hình tương quan đặc biệt (đa cộng tuyến),tập trung vào các tính năng liên tục và có ý nghĩa

Bước 3: Đảm bảo mục tiêu được bao gồm

if 'is_fraud' not in cols_to_plot and 'is_fraud' in df.columns:

cols_to_plot.append('is_fraud')

Đảm bảo biến mục tiêu luôn có trong ma trận để phân tích.

Các thông số của bản đồ nhiệt Seaborn:

- annot=True : Hiển thị tương quan giá trị (-1 đến 1)
- fmt=".2f" : Định dạng 2 phân tích chữ số
- cmap='coolwarm' :
    - Màu xanh lam (mát mẻ) = Tương quan âm
    - Màu đỏ (ấm) = Tương quan dương
    - Màu trắng = Không có mối tương quan
- linewidths=0.5 : Tạo các đường lưới giữa các ô

Kích thước hình (10, 8) :

- Tỷ số bình phương phù hợp cho ma trận đối xứng
- Đủ lớn để đọc chú thích

Giải thích các giá trị tương quan:

Tương quan	Sức mạnh	Giải thích
0,8 - 1,0	Rất mạnh	Mối quan hệ tuyến tính mạnh mẽ
0,6 - 0,8	Mạnh	Mối quan hệ quan trọng
0,4 - 0,6	Vừa phải	Mối quan hệ vừa phải
0,2 - 0,4	Yếu đuối	Mối quan hệ yếu
0,0 - 0,2	Rất yếu	Hầu như không có mối quan hệ tuyến tính nào.

Kết quả phân tích thống kê:

print("Top 10 đặc trưng tương quan mạnh nhất với 'is_fraud':")

```
if 'is_fraud' in corr.columns:
    print(corr['is_fraud'].abs().sort_values(ascending=False).head(10))
```

Phân tích:

- .abs(): Lấy giá trị tuyệt đối (cả mối tương quan tích cực và tiêu cực đều quan trọng)
- .sort_values(ascending=False): Sắp xếp từ cao đến thấp
- .head(10): Top 10 tính năng có mối tương quan mạnh nhất

Ma trận tương quan từ cung cấp những hiểu biết hữu ích:

1. Lựa chọn đặc trưng: Các đặc trưng có hệ số tương quan cao với is_fraud (|r| > 0,3) là những ứng viên tiềm năng cho mô hình lựa chọn sớm.

2. Phát hiện đa cộng tuyến: Không có 2 đặc điểm nào có tương quan > 0,9 với nhau → Xem xét loại bỏ một trong hai.

3. Xác thực tên miền: Kiểm tra xem mối tương quan có khớp với kiến thức tên miền không:

- geo_mismatch → nên có mối tương quan tích cực với gian lận
- điểm an ninh → nên có mối tương quan nghịch với gian lận

4. Ý tưởng về kỹ thuật xử lý đặc trưng:

- Các cặp tương quan cao có thể kết hợp các thuật ngữ tương tác thành công
- Các đặc điểm có độ tương quan thấp có thể được nghiên cứu.

Những hạn chế của phân tích tương quan này:

Chỉ đo các mối quan hệ tuyến tính :

- Không thể bắt được các mẫu phi tuyến tính
- VD: Gian lận có thể xảy ra ở cả số lượng quá thấp và quá cao (hình chữ U)

Nhạy cảm với các giá trị ngoại lệ :

- Các ngoại lệ có thể làm lệch các giá trị tương quan
- Cần kết hợp với số liệu thống kê mạnh mẽ

Tương quan ≠ Nhân quả :

- Tương quan cao không có nghĩa là quan hệ nhân quả
- Cần có kiến thức về miền để phiên dịch

```python
# 3. CHANNEL & MERCHANT
if enable_streamlit: st.markdown("### 4. Phân tích Channel & Merchant")

df_viz = df.copy()
merch_cols = [c for c in df.columns if 'merchant_category_' in c]
col_cat = None
if merch_cols:
    df_viz['merchant_category_label'] = df_viz[merch_cols].idxmax(axis=1).apply(lambda x: x.replace('merchant_categor
    col_cat = 'merchant_category_label'
elif 'merchant_category' in df.columns:
    col_cat = 'merchant_category'

if col_cat:
    fraud_by_cat = df_viz.groupby(col_cat)['is_fraud'].mean().sort_values(ascending=False)
    fig5, ax5 = plt.subplots(figsize=(10, 5))
    fraud_by_cat.plot(kind='bar', color='orange', edgecolor='black', ax=ax5)
    ax5.set_title('Tỷ lệ Gian lận theo Merchant Category')
    plt.xticks(rotation=45)
    show_fig(fig5)

if 'channel' in df_viz.columns:
    if pd.api.types.is_numeric_dtype(df_viz['channel']):
        df_viz['channel_label'] = df_viz['channel'].map({1: 'Web', 0: 'App'})
    else:
        df_viz['channel_label'] = df_viz['channel']

    fraud_by_channel = df_viz.groupby('channel_label')['is_fraud'].mean().sort_values(ascending=False)
    fig6, ax6 = plt.subplots(figsize=(6, 4))
    fraud_by_channel.plot(kind='bar', color='purple', edgecolor='black', ax=ax6)
    ax6.set_title('Tỷ lệ Gian lận theo Channel')
    show_fig(fig6)
```

Mục đích phân tích: Xác định các danh mục và kênh bán hàng có rủi ro cao để:

- Ưu tiên các nguồn lực phòng chống gian lận
- Đặt ngưỡng cụ thể cho từng danh mục
- Hiểu rõ các mô hình gian lận trong các phân khúc kinh doanh khác nhau.

Xử lý dữ liệu được mã hóa một nóng

Báo cáo vấn đề: Dữ liệu có thể có 2 dạng:

1. Phân loại gốc :merchant_category = 'electronics'
2. Mã hóa one-hot : merchant_category_electronics = 1, các mã khác = 0

Giải pháp: Dự phòng đa lớp

Lớp 1: Giải mã mã hóa one-hot

```python
df_viz = df.copy()
merch_cols = [c for c in df.columns if 'merchant_category_' in c]
col_cat = None
if merch_cols:
    df_viz['merchant_category_label'] = df_viz[merch_cols].idxmax(axis=1).apply(lambda x: x.replace('merchant_category_', ''))
    col_cat = 'merchant_category_label'
elif 'merchant_category' in df.columns:
    col_cat = 'merchant_category'

if col_cat:
    fraud_by_cat = df_viz.groupby(col_cat)['is_fraud'].mean().sort_values(ascending=False)
    fig5, ax5 = plt.subplots(figsize=(10, 5))
    fraud_by_cat.plot(kind='bar', color='orange', edgecolor='black', ax=ax5)
    ax5.set_title('Tỷ lệ Gian lận theo Merchant Category')
    plt.xticks(rotation=45)
    show_fig(fig5)
```

Phân tích bước:

Bước 1 : Tìm tất cả các cột có tiền tố 'merchant_category_'

Bước 2 : idxmax(axis=1)- Tìm cột có giá trị max trong mỗi hàng

Row 1: electronics=1, grocery=0, fashion=0 → 'merchant_category_electronics'

Row 2: electronics=0, grocery=1, fashion=0 → 'merchant_category_grocery'

Bước 3 : Xóa tiền tố để lấy nhãn

.apply(lambda x: x.replace('merchant_category_', ''))

'merchant_category_electronics' → 'electronics'

Kỹ thuật trình:

- idxmax() : Hoạt động được vector hóa, vòng lặp nhanh hơn
- Hàm Lambda : Biến đổi nội tuyến
- Chuỗi phương thức : Mã sạch và dễ đọc

Lớp 2: Quay lại cột ban đầu

elif 'merchant_category' in df.columns:

col_cat = 'merchant_category'

Hình ảnh trực quan về danh mục người bán

Tổng hợp:

- Phân nhóm theo danh mục

- Tính toán tỷ lệ gian lận trung bình
- Sắp xếp giảm dần để các danh mục có rủi ro cao lên đầu

Thiết kế biểu đồ:

- Kích thước hình (10, 5) : Định dạng rộng cho nhiều danh mục
- Màu sắc: Cam : Cảnh báo nhưng không ấn tượng như màu đỏ
- xticks rotation=45 : overlap cho long category names
- Biểu đồ cột được sắp xếp : xác định các rủi ro hàng đầu

Các hình thức gian lận điển hình theo từng loại:

Các danh mục rủi ro cao (Tỷ lệ gian lận > 3%):

- Đồ điện tử : Giá trị bán lại cao, dễ dàng thanh lý.
- Thẻ quà tặng : Tương đương tiền mặt trực tiếp
- Trang sức : Giá trị cao, dễ mang theo.
- Hàng hóa game/kỹ thuật số : Giao hàng tức thì, khó hoàn trả.

Các nhóm rủi ro trung bình (1-3%):

- Thời trang/Quần áo : Giá trị trung bình
- Đồ dùng gia đình và sân vườn : Giá cả khác nhau tùy thuộc vào mức giá.

Các nhóm rủi ro thấp (< 1%):

- Hàng tạp hóa : Giá trị thấp, dễ hư hỏng
- Tiền điện nước/hóa đơn : Thanh toán trực tiếp cho nhà cung cấp dịch vụ.
- Đăng ký : Định kỳ, có thể theo dõi

Phân tích kênh

Xử lý kiểu dữ liệu thông minh:

```
if 'channel' in df_viz.columns:
    if pd.api.types.is_numeric_dtype(df_viz['channel']):
        df_viz['channel_label'] = df_viz['channel'].map({1: 'Web', 0: 'App'})
    else:
        df_viz['channel_label'] = df_viz['channel']

    fraud_by_channel = df_viz.groupby('channel_label')['is_fraud'].mean().sort_values(ascending=False)
    fig6, ax6 = plt.subplots(figsize=(6, 4))
    fraud_by_channel.plot(kind='bar', color='purple', edgecolor='black', ax=ax6)
    ax6.set_title('Tỷ lệ Gian lận theo Channel')
    show_fig(fig6)
```

Lý do:

- Kiểm tra xem kênh được lưu trữ dưới dạng số (0/1) hay chuỗi ('Web'/'App')
- Chuyển đổi số thành nhãn dễ đọc
- Sử dụng các nhãn hiện có mặc dù đã là chuỗi

Các hình thức gian lận kênh phân phối điển hình:

Web (Dựa trên trình duyệt):

- Ưu điểm: Nhiều công cụ hơn để gian lận (proxy, VPN, bot)
- Nhược điểm: Có thể có thêm nhiều lớp xác thực.
- Tỷ lệ gian lận điển hình: 2-4%

Ứng dụng di động:

- Ưu điểm: Nhận dạng thiết bị bằng dấu vân tay, khó giả mạo hơn.
- Nhược điểm: Một khi bị xâm nhập, sẽ không thể truy cập được nữa.
- Tỷ lệ gian lận điển hình: 1-2%

Thông tin chi tiết hữu ích từ phân tích theo danh mục:

1. Hệ thống chấm điểm rủi ro động:

Risk = Base_Risk * Category_Multiplier * Channel_Multiplier

2. Quy tắc cụ thể theo từng danh mục:

- Đồ điện tử > 500 đô la → Yêu cầu xác minh bổ sung
- Thẻ quà tặng > 200 đô la → Xác minh qua điện thoại
- Các lô hàng tạp hóa vận chuyển > 500km → Đánh dấu là đáng ngờ

3. Phân bổ nguồn lực: Tập trung đội ngũ phòng chống gian lận vào:

- Ba nhóm rủi ro cao nhất chiếm 60-70% tổng số vụ gian lận.
- Kênh web = Tỷ lệ dương tính giả cao hơn có thể chấp nhận được

4. Chiến lược kinh doanh:

- Hãy cân nhắc loại bỏ các danh mục rủi ro cao khỏi quy trình thanh toán tức thời.
- Thực hiện quy trình xác minh từng bước cho các bên thứ ba có mức độ rủi ro khác nhau.

3.5.4 Phân tích phân bổ số tiền

```
# 4. SỐ TIỀN
if enable_streamlit: st.markdown("### 5. Phân phối số tiền (Amount)")

if 'amount' in df.columns:
    amount = df['amount'].astype(float)
    amount_pos = amount[amount > 0]
    log_amount = np.log(amount_pos)

    fig7, (ax7a, ax7b) = plt.subplots(1, 2, figsize=(14, 5))

    ax7a.hist(amount_pos, bins=60, color='green', alpha=0.7)
    ax7a.set_title('Phân phối số tiền (Gốc)')

    ax7b.hist(log_amount, bins=60, color='teal', alpha=0.7)
    ax7b.set_title('Phân phối số tiền (Log Transform)')
    show_fig(fig7)
```

Cách thực hiện như sau: Phân bổ số tiền giao dịch này để:

- Phát hiện các giá trị ngoại lệ ở mức độ bất thường.
- Xác định chiến lược chuyển đổi
- Hiểu rõ các mô hình giao dịch điển hình
- Xác định các chiến lược gian lận dựa trên số tiền

Làm sạch dữ liệu:

amount = df['amount'].astype(float)

amount_pos = amount[amount > 0]

Lý do lọc số lượng > 0:

- Loại bỏ âm thanh giao dịch (hoàn tiền, chỉnh sửa)
- bỏ giao dịch có số tiền bằng không (giao dịch thử nghiệm)
- Tập trung vào các giao dịch mua hàng thực tế
- Ngăn chặn lỗi log(0) = -vô cực

Chiến lược trực quan hóa hai bảng:

Lý do cho quan điểm kép: Phân bổ số tiền giao dịch thường là:

- Độ lệch phải nhiều : Nhiều giao dịch nhỏ, ít giao dịch lớn
- Đuôi dài : Một số ngoại lệ với số lượng rất cao
- Khó hình dung : Biểu đồ phân bố chuẩn bị chi phối bởi khối lượng chính.

Giải pháp: So sánh dữ liệu gốc với dữ liệu đã được biến đổi logarit.

Bảng 1: Biểu đồ tần suất theo thang đo gốc

ax1.hist(amount_pos, bins=60, color='green', alpha=0.7)

Cấu hình:

- bins=60 : Cân bằng giữa chi tiết và độ mượt mà
- màu='xanh lá cây' : Liên quan đến tiền/tiền tệ
- alpha=0.7 : Bán trong suốt để hiển thị các phần chồng lấp nếu có

Thông tin đã thu thập:

- Phân bổ số tiền đô la hiện tại
- Phạm vi giao dịch đa số
- Khả năng hiển thị các giá trị ngoại lệ cực đoan
- Quy mô kinh doanh tự nhiên

Bảng 2: Biểu đồ tần số được biến đổi logarit

log_amount = np.log(amount_pos)

ax2.hist(log_amount, bins=60, color='teal', alpha=0.7)

PHẦN 4: MÔ HÌNH HỌC MÁY SỬ DỤNG

Việc hiểu và vận dụng linh hoạt các thư viện Python đóng vai trò cốt lõi trong quy trình Khoa học dữ liệu. Trong đồ án này, nhóm sử dụng các thư viện chuyên dụng sau:

- NumPy (Numerical Python): Thư viện nền tảng cho tính toán khoa học, cung cấp các đối tượng mảng đa chiều mạnh mẽ và bộ công cụ đại số tuyến tính tối ưu hóa hiệu suất.
- Pandas: Thư viện phổ biến nhất để xử lý và phân tích dữ liệu có cấu trúc. Pandas cung cấp cấu trúc dữ liệu DataFrame và Series, cho phép thao tác, lọc và làm sạch dữ liệu bảng một cách hiệu quả.
- Matplotlib: Thư viện nền tảng cho trực quan hóa dữ liệu trong Python, hỗ trợ vẽ các biểu đồ 2D chất lượng cao với khả năng tùy chỉnh chi tiết.
- Seaborn: Thư viện trực quan hóa thống kê được xây dựng dựa trên Matplotlib. Seaborn cung cấp giao diện cấp cao, giúp vẽ các biểu đồ thống kê đẹp mắt và trực quan hơn (như Heatmap, Distribution plot) với ít dòng lệnh hơn.
- Scikit-learn (Sklearn): Thư viện toàn diện cho Machine Learning, cung cấp các thuật toán hiệu quả cho phân loại (Classification), hồi quy (Regression), phân cụm (Clustering) và các công cụ tiền xử lý dữ liệu.

4.1 Tách dữ liệu và Hiểu mẫu

4.1.1. Tổng quan về tách dữ liệu và mất cân bằng

Sự mất cân bằng lớp có một trong những công thức lớn nhất trong phân loại Machine Learning, đặc biệt nghiêm trọng trong việc phát hiện gian lận bài toán, chẩn đoán bệnh và phát hiện bất thường. Khi tỷ lệ giữa các lớp chênh lệch quá lớn (ví dụ: 1 :100 hoặc 1 :1000), chuẩn toán Machine Learning các thuật toán gặp phải nhiều vấn đề:

- **Vấn đề 1: Xu hướng hướng tới** Mô hình lớp đa số có xu hướng học để dự đoán lớp đa số (giao dịch thông thường) vì công việc này tối đa hóa độ chính xác tổng thể. Một mô hình đơn giản dự đoán tất cả là "không lừa đảo" vẫn có thể đạt độ chính xác 99% dù tỷ lệ gian lận là 1% nhưng hoàn toàn vô dụng trong thực tế.
- **Vấn đề 2: Khả năng học tập kém của các mô hình thiểu số** Với quá ít mẫu từ tầng lớp thiểu số, mô hình không có đủ ví dụ để nghiên cứu các mô hình cụ thể của các giao dịch gian lận, dẫn đến tỷ lệ âm tính giả cao (bỏ sót gian lận).

- **Vấn đề 3: Độ dốc giảm dần không hiệu quả** Trong các thuật toán dựa trên độ dốc (mạng lưới thần kinh, hồi quy logistic), độ dốc bị chi phối bởi lớp đa số, tối ưu hóa không hội tụ tốt cho lớp thiểu số.

Tìm hiểu thêm về mô-đun ML Pipeline

Mô-đun này đóng vai trò sau đó kết thúc trong công việc:

- **Dữ liệu riêng biệt để đánh giá hiệu suất mô hình trung thực**
- **Xử lý mất cân bằng lớp một cách khoa học và có hệ thống**
- **Đảm bảo đánh giá mô hình phản ánh đúng kịch bản trong thế giới thực**
- **Ngăn ngừa rò rỉ dữ liệu và sai lệch đánh giá**

Đó là lý do tại sao việc chuẩn bị dữ liệu được dùng cho quá trình huấn luyện mô hình, và đó là nơi mô hình được sử dụng.

4.1.2. Tách tính năng và nhãn

```python
def prepare_data_for_model(df):
    """
    Chuẩn bị dữ liệu để đưa vào huấn luyện (Encoding -> Split -> Scale -> Undersample)
    """
    print("--- [Preprocessing] Đang chuẩn bị dữ liệu cho Model ---")

    # 1. One-Hot Encoding (Chuyển chữ thành số)
    # Lấy danh sách các cột object (trừ is_fraud)
    categorical_cols = df.select_dtypes(include=['object']).columns
    if len(categorical_cols) > 0:
        print(f"- Đang One-Hot Encoding các cột: {list(categorical_cols)}")
        df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)
```

Phân Tích Machine Learning m**ô hình học tập có giám sát:** Trong học tập có giám sát, chúng ta cần phải phân biệt đ**ặc điểm (X)** : Biến độc lập, biến dự báo, đầu vào và n**hãn (y)** : Biến phụ thuộc, biến mục tiêu, đầu ra

Đây là nền tảng cho học có giám sát: học ánh xạ hàm f(X) → y.

Chiến lược loại bỏ cột:

- Transaction_id được gán cho User_id với lý do loại bỏ: Identifier columns không chứa predictive information,mỗi transaction có unique ID=>nếu giữ lại → Model có thể memorize IDs (overfitting)

-Xem phần "quá khớp" đối với ID:model học: "transaction_12345 là fraud"

Nhưng trong test set: transaction_67890 (chưa thấy) → Model không biết phản ứng

- is_fraud:lý do loại bỏ target variable vì đây chính là cái chúng ta muốn predictvà không thể dùng y để predict y (trivial solution)

-transaction_time:lý do loại bỏ temporal identifier vì timestamp cụ thể không generalize,đã extract ra features như 'hour', 'day_of_week',giữ raw timestamp → Model học specific times thay vì patterns

[c for c in cols_to_drop if c in df.columns] : Chỉ thả cột thực sự tồn tại, tránh KeyError. Đó là cách thực hành tốt nhất để có mã sản xuất mạnh mẽ.

Xác minh hình dạng:print("Kích thước dữ liệu gốc:", X.shape**)**

Kiểm tra sơ bộ cần thiết:

- X.shape[0]: Số lượng mẫu (hàng)
- X.shape[1]: Số lượng đặc trưng (cột)

Định dạng đầu ra dự kiến:

```
--- [Preprocessing] Đang chuẩn bị dữ liệu cho Model ---
- Đang One-Hot Encoding các cột: ['country', 'bin_country', 'channel', 'merchant_category']
- Đang thực hiện Undersampling trên tập Train...
-> Kích thước Train sau Undersampling: (50908, 40)
-> Kích thước Test (giữ nguyên): (89909, 40)
```

4.1.3. Phân chia thử nghiệm đào tạo và phân tầng

```
# 3. Chia Train/Test (Trước khi Undersample để tránh data leakage)
# Stratify=y để đảm bảo tỷ lệ fraud ở 2 tập đều nhau
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```

Nguyên Cơ Tắc Bản: Hold-Out Validation

Phân chia dữ liệu thành tập huấn luyện và tập kiểm thử cho học máy:

“Không bao giờ kiểm thử trên dữ liệu mà bạn đã dùng để huấn luyện.”

Mô hình phải được đánh giá dựa trên dữ liệu chưa từng được tìm thấy (dữ liệu chưa nhìn thấy) để đo lường khả năng khái quát hóa chứ không phải ghi nhớ.

Các phép so sánh tương tự: Học sinh làm bài thi phải khác với bài tập về nhà (train set),model = học sinh, if test = train → chỉ đo trí nhớ ngắn hạn, không đo hiểu biết thực

=> Tại sao phải chia Train/Test trước?

Nguyên tắc vàng: KHÔNG BAO GIỜ được xử lý dữ liệu trước khi chia train/test,lý do bởi tránh Data Leakage:Nếu undersampling toàn bộ dataset

trước khi chia,các samples từ test set có thể "rò rỉ" vào training processvà mô hình học được thông tin từ test set → đánh giá không thực tế.Bảo toàn distribution thực tế:test set phải đại diện cho dữ liệu production thực tếvà dữ liệu thực tế luôn imbalanced

Phân tích tham số:

1. test_size=0.3 (Tỷ lệ chia 70-30)

Lý do: 70% Train : 30% Test là standard ratio trong ML

Lựa chọn tối ưu:

- Bộ dữ liệu nhỏ (< 10K): 70-30 hoặc xác thực chéo
- Phạm vi bộ dữ liệu (10K-100K): 80-20 (tiêu chuẩn)
- Độ dài tập dữ liệu (> 1 triệu): 90-10 (đủ mẫu ở cả hai loại)

2. random_state=42

Mục đích,khả năng tái tạo random_state=42

Không có random_state:

Run 1: Test accuracy = 92.3%

Run 2: Test accuracy = 91.8%

Run 3: Test accuracy = 93.1%

→ Results không consistent, không thể compare experiments

Với random_state:

Run 1: Test accuracy = 92.5%

Run 2: Test accuracy = 92.5%

Run 3: Test accuracy = 92.5%

→ Results reproducible, có thể compare fairly

random_state=42:Giups đảm bảo tính tái lập (reproducibility),mọi lần chạy đều cho cùng kết quả chia,quan trọng cho debugging và experiment tracking

3. stratify=y (Lấy mẫu phân tầng)

Thông số quan trọng đối với dữ liệu không cân bằng

Vấn đề không phân tầng:

Original data: 99% normal, 1% fraud

Random split có thể tạo ra:Train set: 99.5% normal, 0.5% fraud (quá ít fraud samples) và test set: 98.5% normal, 1.5% fraud (không representative)

Giải pháp cho việc phân tầng xã hội:

Stratify=y đảm bảo:

Train set: 99% normal, 1% fraud (giữ nguyên ratio)

Test set: 99% normal, 1% fraud (giữ nguyên ratio)

Giải thích bằng toán học việc l**ấy mẫu phân tầng đảm bảo:**

P(fraud | train) = P(fraud | test) = P(fraud | original) với cơ chế thực hiện:chia data thành 2 groups: fraud và normalvà từ mỗi group, lấy 80% vào train, 20% vào test sau đó combine lại → đảm bảo class distribution consistent

Tại sao điều này lại quan trọng đối với việc phát hiện gian lận:

Phát hiện gian lận có sự mất cân bằng cực độ (lừa đảo thông thường < 1%):

- Không phân tầng: Tập dữ liệu thử nghiệm có thể có 0 mẫu gian lận → Không thể đánh giá
- Với phân tầng: Bộ kiểm tra đảm bảo có đủ mẫu gian lận để đánh giá

Quy ước đặt tên X_train_raw, y_train_raw hậu tố "_raw" được chỉ định rõ đây là dữ liệu huấn luyện TRƯỚC KHI lấy mẫu dưới, khi quan trọng để theo dõi luồng dữ liệu.

Báo cáo bộ kiểm thử:

```
# 4. Scaling (Fit trên Train, Transform trên Test)
scaler = MinMaxScaler()
X_train = pd.DataFrame(scaler.fit_transform(X_train), columns=feature_names)
X_test = pd.DataFrame(scaler.transform(X_test), columns=feature_names)
```

Mục đích của việc báo cáo này nhằm đặc điểm của bộ kiểm thử tài liệu,xác minh quá trình phân tầng đã hoạt động chính xác và thiết lập cơ sở ban đầu để đánh giá mô hình.

4.1.3. Chiến lược Undersampling trên Train Set

```
# 5. Undersampling (Chỉ thực hiện trên tập Train)
print("- Đang thực hiện Undersampling trên tập Train...")

# Gom lại tạm thời để resample
train_data = pd.concat([X_train, y_train.reset_index(drop=True)], axis=1)

not_fraud = train_data[train_data.is_fraud == 0]
fraud = train_data[train_data.is_fraud == 1]

# Tỷ lệ 1:10 (1 Fraud : 10 Normal) - Giống logic bài của bạn
n_fraud = len(fraud)
n_not_fraud = n_fraud * 10
```

Phân Tích Machine Learning - Lý thuyết Undersampling (**định Nghĩa Lấy mẫu thiếu):** Undersampling là kỹ thuật xử lý mất cân bằng lớp bằng cách **giảm số lượng mẫu từ lớp đa số** để có thể bằng với lớp thiểu số.

Kỹ thuật Undersampling: phương pháp: Random Under-Sampling (RUS),lấy ngẫu nhiên n samples từ lớp đa số,đơn giản, hiệu quả, dễ implement với tham số quan trọng: replace=False: Lấy mẫu không hoàn lại,không tạo ra duplicate samplesvà mỗi sample chỉ xuất hiện 1 lần

random_state=42: Đảm bảo reproducibilitycó tỷ lệ 1:10 (Fraud:Normal),lựa chọn dựa trên heuristic/domain knowledge cần tuning: Có thể thử 1:5, 1:20, 1:50với quy tắc chung: Đủ samples để học patterns nhưng không quá imbalanced

Ví dụ trước khi undersampling:Giả sử dataset có 1% fraud với train set (80%): 80,000 samples, fraud: 800 samples (1%),Normal: 79,200 samples (99%) imbalance Ratio: 99:1

Sau khi undersampling (1:10):Fraud: 800 samples (giữ nguyên), normal: 8,000 samples (downsampled từ 79,200),imbalance Ratio: 10:1,fraud Rate mới: 800/(800+8000) = 9.09%

-Ưu điểm của Undersampling:

+Giảm thời gian training: Ít samples hơn

+Cân bằng classes: Mô hình không bị bias về lớp đa số

+Giảm overfitting: Với lớp đa số (nếu over-represented)

-Nhược điểm:mất thông tin(bỏ đi nhiều samples từ lớp đa số)và có thể bỏ sót patterns quan trọng từ lớp đa số=>không phù hợp khi số samples lớp thiểu số quá ít

4.1.4. Triển khai lấy mẫu lại

```
# Nếu số lượng normal nhiều hơn mức cần thiết thì cắt bớt
if len(not_fraud) > n_not_fraud:
    not_fraud_downsampled = resample(not_fraud,
                                     replace=False,
                                     n_samples=n_not_fraud,
                                     random_state=42)
else:
    not_fraud_downsampled = not_fraud
```

Hàm resample() của Sklearn:

Đây là hàm tiện ích từ sklearn.utilscung cấp khả năng lấy mẫu lại linh hoạt.

Phân tích tham số: train_normal (Đối số đầu tiên) Input data: DataFrame chứa tất cả normal transactions và. replace=False=>thông số quan trọng cho việc lấy mẫu thiếu

-replace=False (Lấy mẫu không hoàn lại): Mỗi sample chỉ có thể được chọn một lần n_samples = 8,000 → Chọn 8,000 unique samples

-Nếu replace=True (Sai đối với trường hợp lấy mẫu thiếu):Sample có thể được chọn nhiều lần với n_samples = 8,000 → Có thể có duplicates→ Không phải undersampling mà là random oversampling với duplicates

-n_samples=n_normal_target : Số lượng samples muốn lấy ra= n_fraud × 10

-random_state=42 reproducibility: Same subset mỗi lần chạy

4.1.5. Chuyển sang giai đoạn chuẩn bị cuối cùng của quá trình kết hợp dữ liệu.

```
# Gộp lại thành tập Train cân bằng
train_balanced = pd.concat([fraud, not_fraud_downsampled])

# Trộn ngẫu nhiên (Shuffle)
train_balanced = train_balanced.sample(frac=1, random_state=42).reset_index(drop=True)
```

Phân Tích Machine Learning

-Bước 1: Nối chuỗi train_balanced = pd.concat([train_fraud, train_normal_downsampled])

Kết hợp lại các lớp:

TẤT CẢ các mẫu gian lận (800)

Mẫu chuẩn được lấy mẫu giảm (8.000)

Tổng cộng: 8.800 mẫu

Tỷ lệ: 1:10

Hiệu quả về bộ nhớ: pd.concat() không sao chép dữ liệu một cách không cần thiết.

-Bước 2: Trộn bài train_balanced = train_balanced.sample(frac=1, random_state=42)

Bước quan trọng - Tại sao cần xáo bài?

Vấn đề khi không xáo trộn:

+train_balanced structure:

+[0-799]: All fraud samples

+[800-8799]: All normal samples

→ Batch-based algorithms sẽ xử lý:

Batch 1: 100% fraud

Batch 2: 100% fraud

...

Batch 9: 100% normal

...Các vấn đề phát sinh:

1. Sự bất ổn của SGD : Thuật toán giảm độ dốc ngẫu nhiên với các lô đồng nhất → Độ dốc bị lệch
2. Khả năng hội tụ kém : Mô hình luân phiên giữa việc học các mẫu gian lận và các mẫu bình thường.
3. Các vấn đề về chuẩn hóa hàng loạt : Thống kê của từng đợt không mang tính đại diện
4. Vấn đề về tốc độ học : Độ lớn của gradient thay đổi đáng kể giữa các lô dữ liệu.

Giải pháp thông qua việc xáo trộn:

After shuffle:

Batch 1: Mix of fraud and normal (representative)

Batch 2: Mix of fraud and normal (representative)

...

→ Mỗi batch là mini-version của toàn bộ dataset

-frac=1 nghĩa là:frac = fraction of data to sample,frac=1 → Sample 100% of data

→ Không giảm data, chỉ reorder

Bước 3: Phân tách nhãn đặc trưng

```
# Tách lại X_train, y_train
X_train_final = train_balanced.drop('is_fraud', axis=1)
y_train_final = train_balanced['is_fraud']
```

Chuẩn bị cho việc nhập dữ liệu vào mô hình:

- X_train: Chỉ bao gồm các đặc trưng (biến dự đoán)
- y_train: Chỉ chứa nhãn (mục tiêu)

Xác minh:

```
print(f"-> Kích thước Train sau Undersampling: {X_train_final.shape}")
print(f"-> Kích thước Test (giữ nguyên): {X_test.shape}")
```

4.2 Phân tích Mô hình Decision Tree

4.2.1 Nguyên lý hoạt động

Cây quyết định dựa trên giá trị của một cụ thể, mỗi nhánh đại diện cho kết quả của quyết định đó và mỗi nút lá (nút lá) đại diện cho một kết quả cuối cùng được mong đợi. Quá trình xây dựng cây bắt đầu từ nút gốc (nút gốc) chứa toàn bộ dữ liệu đào tạo, sau đó thuật toán tìm kiếm cụ thể và ngưỡng (ngưỡng) tốt nhất để chia (tách) dữ liệu thành hai nhóm con sao cho mỗi nhóm có tính đồng nhất (đồng nhất) cao hơn về nhãn lớp. Việc lựa chọn phân chia tốt nhất được thực hiện thông qua công việc tính toán các tạp chất đo độ (độ tạp) như Gini Index hoặc Entropy. Gini Gini = 1 - $\Sigma(p_i)^2$chỉ số i tại nút đó - giá trị Gini bằng 0 khi nút hoàn toàn tinh khiết (chỉ chứa một lớp) và đạt giá trị tối đa khi các lớp phân bố đều nhau. Ngược lại, Entropy đo tốc độ bất động hoặc bất chấp trong dữ liệu theo công thức Entropy = $-\Sigma(p\_i \times \log_2(p\_i))$, cũng đạt giá trị 0 khi nút tĩnh tinh và tối đa khi các lớp phân bố đều. Thuật toán sẽ tính toán Information Gain (entropy giảm tốc độ) cho từng đặc thù có thể phân chia bằng công thức IG = Entropy(parent) - Σ(weight_i × Entropy(child_i))và chọn phân chia có Information Gain cao nhất. Quá trình phân tách này được lặp lại bằng cách đệ quy (đệ quy) cho mỗi nút cho đến khi đạt được một trong các điều kiện dừng: tất cả các mẫu trong

nút thuộc cùng một lớp (nút thuần túy), độ sâu cây đạt max_deep đã được xác định, số lượng mẫu trong nút nhỏ hơn min_samples_split hoặc việc chia thêm không cải thiện được tạp chất đáng kể. Khi dự đoán cho một mẫu mới, cây sẽ duyệt (duyệt) từ nút gốc xuống các nút theo quyết định tại mỗi nút dựa trên các đặc tính giá trị của mẫu đó, cho đến khi đạt được một nút lá và trả về nhãn lớp được lưu tại lá đó (Thường là lớp đa số của các mẫu đào tạo rơi vào lá đó).

4.2.2 Lý do chọn mô hình

Cây quyết định có thể được sử dụng để xác định việc phát hiện gian lận, đó là lý do tại sao việc này lại quan trọng. Thứ nhất, tính có thể diễn giải (khả năng giải thích) - đây là ưu điểm lớn nhất của Cây quyết định so với các mô hình hộp đen như Mạng thần kinh hay SVM, vì cây quyết định có thể hình dung dễ dàng thành một sơ đồ mà người ta có thể đọc và hiểu được logic ra quyết định, điều này cực kỳ quan trọng trong việc phát hiện gian lận vì hệ thống cảnh báo một giao dịch là khống lồ, các nhà phân tích gian lận cần hiểu tại sao màn hình đưa ra quyết định đó để xác định và giải quyết cho khách hàng, đồng thời tuân thủ các yêu cầu quy định trong ngành tài chính chính thường yêu cầu tính minh bạch trong các quyết định ảnh hưởng tự động đến khách hàng. Thứ hai, khả năng xử lý dữ liệu mất cân bằng - với tham số class_weight='balanced', Cây Quyết định có thể tự động điều chỉnh để không bị sai lệch về loại đa số, đây là vấn đề trong việc phát hiện gian lận khi tỷ lệ gian lận thường chỉ sử dụng 0,1-2% tổng số giao dịch, gán trọng số cao hơn cho các mẫu gian lận, giảm nặng hơn khi phân loại sai gian lận, từ đó tăng khả năng thu hồi (khả năng bắt được gian lận) thay vì có thể chỉ tối ưu hóa độ chính xác tổng. Thứ ba, không cần chia tỷ lệ tính năng - khác với các thuật toán dựa trên khoảng cách như KNN, SVM, hoặc dựa trên độ dốc như Mạng thần kinh Yêu cầu chuẩn hóa/chuẩn hóa dữ liệu, Cây quyết định hoạt động dựa trên so sánh ngưỡng nên không bị ảnh hưởng bởi quy mô của các tính năng, tiết kiệm công sức tiền xử lý và tránh mất thông tin từ các ngoại lệ (mà các ngoại lệ thường là dấu hiệu gian lận). Thứ tư, xử lý tốt cả các tính năng số và phân loại - Cây quyết định có thể xử lý các loại dữ liệu hỗn hợp theo cách tự nhiên thông qua phân chia nhị phân, không cần mã hóa một lần phức tạp cho các biến phân loại như mô hình tuyến tính, đặc biệt hữu ích với giao dịch dữ liệu thường chứa cả số (số lượng, khoảng cách) và danh mục (merchant_category, country code). Thứ năm, tự động phát hiện tương tác tính năng - cây quyết định tự động nắm bắt các tương tác giữa các tính năng thông qua cấu trúc phân cấp của nó, ví dụ: mức phân chia cao có thể là "amount > $1000" và chia tiếp theo trong cây con có thể là "shipping_distance > 500km", tạo thành quy tắc "IF money > 1000 AND Shipping_distance > 500 THEN high_fraud_risk" mà không cần phải thiết kế thủ công các thuật ngữ

tương tác như trong tuyến tính hồi quy. Thứ sáu, mạnh mẽ đối với các ngoại lệ - vì Cây Quyết định chỉ quan tâm đến thứ tự đối số của các giá trị khi tìm điểm phân chia tối ưu, nên một vài giá trị cực trị không làm sai lệch mô hình như cách chúng ảnh hưởng đến giá trị trung bình/phương sai trong các mô hình tuyến tính, điều này quan trọng vì các giao dịch gian lận thường có các mẫu bất thường (ngoại lệ) mà chúng ta muốn phát hiện chứ không muốn bỏ qua.

4.2.3 Train Model

Khai báo Thư viện và Định nghĩa Hàm

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import recall_score
```

Mã đầu tiên thực hiện việc nhập các thư viện cần thiết từ scikit-learn để xây dựng và đánh giá Cây quyết định mô hình. Đầu tiên là DecisionTreeClassifiertừ mô-đun sklearn.tree- đây là lớp chính để tạo mô hình quyết định bài toán phân loại, một trong những cơ bản máy học thuật toán cơ bản nhất nhưng rất hiệu quả và dễ hiểu. Tiếp theo là GridSearchCVtừ sklearn.model_selection- công cụ mạnh mẽ để tự động tìm kiếm thông số tối ưu thông qua thử tất cả các tổ hợp tham số trong một "lưới" (lưới) đã được xác định trước đó, kết hợp với xác thực chéo để đảm bảo tính tổng hợp của mô hình. Cuối cùng là recall_scoretừ sklearn.metrics- số liệu quan trọng nhất trong phát hiện gian lận để đo tỷ lệ chuyển đổi được phát hiện ở mức độ khổng lồ trong tổng số gian lận thực tế. Sau khi nhập xong, mã xác định một hàm train_decision_tree(X_train, y_train)nhận vào hai tham số là các tính năng đào tạo dữ liệu (X_train) và nhãn (y_train), đây là cách thực hành tốt nhất trong trình cài đặt để tái sử dụng mã và chức năng logic rõ ràng. Câu lệnh in đầu tiên sử dụng chuỗi f {'='*20}để tạo ra 20 dấu bằng ở mỗi bên, tạo thành một tiêu đề đẹp mắt giúp dễ dàng phân tích các phần đầu ra khi chạy nhiều mô hình khác nhau, đặc biệt hữu ích khi đăng nhập kết quả hoặc gỡ lỗi.

```
# --- Decision Tree ---
dt_model = train_decision_tree(X_train, y_train)
evaluate_model(dt_model, X_test, y_test, "Decision Tree")
plot_feature_importance(dt_model, feature_names, "Decision Tree")
```

dt_model = train_decision_tree(X_train, y_train)

Dòng mã đầu tiên này thực hiện chức năng gọi hàm train_decision_tree()- một hàm tùy chỉnh đã được xác định trước đó trong mã để huấn luyện mô hình Cây quyết định với đầy đủ các bước từ cơ sở đến điều chỉnh. Nhận xét # --- Decision Tree ---được đặt ở phía trên để đánh dấu phần

liên quan đến mô hình Cây quyết định, giúp sắp xếp mã rõ ràng hơn đặc biệt khi có nhiều mô hình khác nhau trong cùng một tệp (ví dụ còn có các phần Rừng ngẫu nhiên, XGBoost, Hồi quy logistic khác). Hàm train_decision_tree()nhận vào hai đối số: X_traincó DataFrame hoặc mảng numpy chứa các tính năng của tệp đào tạo (đã được xử lý trước, được thiết kế tính năng và được lấy mẫu dưới các bước trước), và y_trainlà Chuỗi hoặc mảng chứa nhãn tương ứng (0 cho không gian lận, 1 cho gian lận). Bên trong chức năng này (như đã phân tích chi tiết ở phần trước), quá trình đào tạo diễn ra qua 3 bước chính: (1) khởi tạo và huấn luyện Cây quyết định cơ sở với tham số mặc định (random_state=42và class_weight='balanced') để thiết lập điểm bắt đầu, tính toán thu hồi trên tập huấn luyện để có số liệu cơ sở; (2) thực hiện GridSearchCV với param_grid chứa các siêu tham số cần điều chỉnh ('criterion', 'max_deep', 'min_samples_split'), sử dụng xác thực chéo 3 lần và tính điểm='recall' để tìm ra tổng hợp các tham số tối ưu, quá trình này sẽ đào tạo và đánh giá 24 cấu hình mô hình khác nhau (kết hợp 2×4×3) để chọn ra cấu hình tốt nhất; (3) để bạn có thể nhớ lại mô hình cơ sở và điều chỉnh để cải thiện, sau đó quay lại best_estimator_các thông số tốt nhất. Kết quả được lưu vào biến dt_model- đây là một đối tượng DecisionTreeClassifier đã được huấn luyện có thể được sử dụng ngay để dự đoán trên tập kiểm tra hoặc dữ liệu mới. Việc gói gọn toàn bộ logic đào tạo vào một hàm riêng là cách thực hành tốt nhất trong lập trình vì giúp mã có thể tái sử dụng (có thể gọi lại nhiều lần với các tập dữ liệu khác nhau), có thể bảo trì (dễ sửa đổi logic ở một chỗ) và có thể đọc được (tập lệnh chính ngắn gọn, logic phức tạp được trừu tượng hóa).

evaluate_model(dt_model, X_test, y_test, "Decision Tree")

Dòng mã thứ hai gọi hàm evaluate_model()để đánh giá hiệu suất toàn diện của Cây quyết định mô hình vừa được huấn luyện trên tập thử nghiệm - một dữ liệu hoàn toàn độc lập mà mô hình chưa từng "thấy" trong quá trình đào tạo và điều chỉnh, đảm bảo đánh giá khách quan về khả năng khái quát hóa của mô hình. Hàm nhận vào 4 đối số: dt_modellà đối tượng mô hình được huấn luyện từ bước trước, X_testlà các tính năng của tập kiểm tra (DataFrame hoặc mảng chứa cùng các cột như đầu và được giữ nguyên để đánh giá), y_testcó nhãn thực của tập kiểm tra, "Decision Tree"có một chuỗi xác định để biết đây là mô hình gì và hiển thị tên thị giác phù hợp trong đầu ra/sơ đồ. Bên trong hàm evaluate_model(), nhiều số liệu quan trọng sẽ được tính toán: lần sử dụng đầu tiên dt_model.predict(X_test)để dự đoán nhãn cho bộ kiểm tra, sau đó tính toán Thu hồi theo recall_score(y_test, y_pred)công thức TP/(TP+FN) - đây là số liệu quan trọng nhất trong phát hiện gian lận vì thước đo tỷ lệ gian lận được phát hiện thành công trong tổng số thực tế gian lận, thu hồi cao (ví dụ >80%) có nghĩa là hệ thống bắt được nhiều số gian lận, thu hồi có nghĩa là nhiều gian lận đã lọt qua; Độ chính xác được tính toán bằng precision_score(y_test, y_pred)công thức TP/(TP+FP) - đo lường trong số các

giao dịch được mong đợi là gian lận thì có bao nhiêu % thực tế là gian lận, độ chính xác cao có nghĩa là có cảnh báo sai (cảnh báo sai); F1-Score là giá trị trung bình hài hòa của độ chính xác và gợi lại công thức 2×(P×R)/(P+R), cân bằng giữa hai số liệu này và đặc biệt hữu ích khi cần một số liệu duy nhất để so sánh các mô hình; Độ chính xác luôn là dự đoán chính xác và là tập dữ liệu rất mất cân bằng với dự đoán mô hình có độ chính xác "không lừa đảo" là 99% và gian lận là 1%; Ma trận nhầm lẫn được tạo bằng cách confusion_matrix(y_test, y_pred)hiển thị chi tiết 4 số: TN (âm tính thật - dự đoán đúng không gian lận), FP (dương tính giả - cảnh báo sai), FN (âm tính giả - bỏ sót gian lận - NGUY HIỂM Tốt), và TP (dương tính thật - bắt đúng gian lận); Báo cáo phân loại hiển thị classification_report(y_test, y_pred)độ chính xác, thu hồi, f1 cho từng lớp riêng biệt cùng với hỗ trợ (số lượng mẫu); và có thể tính thêm ROC-AUC bằng roc_auc_score(y_test, y_prob)cách sử dụng hàm get thông tin xác suất predict_proba(). Tất cả các số liệu này đều được in ra bảng điều khiển với định dạng đẹp mắt (có thể sử dụng màu sắc, đường viền, thụt lề) để dễ đọc và có thể lưu vào tệp nhật ký hoặc cơ sở dữ liệu để theo dõi thử nghiệm. Việc có một chức năng đánh giá tập trung như thế này cực kỳ có giá trị vì đảm bảo tính nhất quán - Tất cả các mô hình đều được đánh giá theo cùng một tiêu chuẩn, dễ dàng so sánh một cách công bằng và nếu muốn bổ sung số liệu mới (ví dụ Hệ số tương quan Matthews) chỉ cần thêm vào một chỗ thay vì sửa đổi nhiều nơi.

plot_feature_importance(dt_model, feature_names, "Decision Tree")

Dòng mã cuối cùng gọi hàm plot_feature_importance()để trực quan hóa tầm quan trọng của các tính năng trong công việc quyết định mô hình Cây quyết định, một bước quan trọng đối với khả năng diễn giải mô hình và hiểu biết sâu sắc về kỹ thuật tính năng. Hàm nhận 3 đối số: dt_modellà đối tượng mô hình được đào tạo chứa thông tin về tầm quan trọng của tính năng, feature_nameslà một danh sách hoặc Chỉ mục chứa tên của tất cả các tính năng (Thường lấy từ X_train.columnshoặc X_test.columns), cần có để ánh xạ từ chỉ mục tính năng sang tên có thể đọc được, "Decision Tree"có tên mô hình để hiển thị tiêu đề cốt truyện mạnh mẽ. Bên trong hàm, lần đầu tiên trích xuất tầm quan trọng của tính năng từ mô hình bằng dt_model.feature_importances_- đây là một mảng có độ dài bằng số lượng tính năng, mỗi giá trị đại diện cho điểm tầm quan trọng của tính năng tương ứng, được tính dựa trên tổng mức giảm tạp chất Gini (hoặc entropy nếu sử dụng tiêu chí = 'entropy') mà tính năng đó đóng góp trên tất cả các phân tách trong cây, các giá trị được chuẩn hóa để tổng thành 1,0. Sau đó sắp xếp các tính năng theo mức độ quan trọng tăng dần np.argsort(importances)[::-1], thường chỉ lấy top 10 hoặc 20 tính năng quan trọng nhất để cốt truyện không bị lộn xộn. Tiếp theo, tạo trực quan hóa bằng matplotlib và seaborn: sử dụng plt.figure(figsize=(10,6))để đặt kích thước hình phù hợp, sns.barplot()hoặc plt.barh()để vẽ biểu đồ thanh ngang với tên tính năng ở trục y và giá trị quan

trọng ở trục x, áp dụng bảng màu như 'viridis' hoặc 'coolwarm' để tạo màu gradient giúp phân biệt mức độ quan trọng, thêm tiêu đề với tên mô hình như "Tầm quan trọng của tính năng - Cây quyết định", nhãn cho các trục ("Tính năng" và "Điểm quan trọng") và có thêm các đường lưới để plt.grid(alpha=0.3)dễ đọc giá trị. Lô có thể được cải tiến bổ sung bằng cách thêm nhãn giá trị trên mỗi thanh bằng cách sử dụng plt.text(), xoay nhãn trục x nếu tên đặc điểm dài hoặc sử dụng các màu khác nhau cho tác động tích cực và tiêu cực (mặc dù tầm quan trọng của Cây Quyết định luôn không tiêu cực). Cuối cùng, save figure with plt.savefig(f'feature_importance_{model_name}.png', dpi=300, bbox_inches='tight')để lưu vào file có độ phân giải cao và plt.show()để hiển thị. Tính năng quan trọng này cốt truyện cung cấp nhiều giá trị: (1) Hiểu mô hình - giúp các nhà khoa học dữ liệu và chuyên gia miền hiểu mô hình đang "nhìn" vào các tính năng gì, xác thực logic có lý do không (ví dụ: if security_score, geo_mismatch là các tính năng hàng đầu thì rất nhạy cho việc phát hiện gian lận); (2) Lựa chọn tính năng - xác định các tính năng có tầm quan trọng gần 0 có thể bị loại bỏ để đơn giản hóa mô hình, giảm bộ nhớ, tăng tốc độ suy luận; (3) Hướng kỹ thuật tính năng - nếu số lượng_ratio quan trọng thì có thể thiết kế thêm các tính năng tỷ lệ khác, nếu geo_mismatch quan trọng thì đầu tư vào chất lượng dữ liệu vị trí; (4) Gỡ lỗi - nếu một tính năng không được mong đợi quan trọng lại có tầm quan trọng cao (hoặc ngược lại) thì có thể có rò rỉ dữ liệu, lỗi xử lý trước hoặc cần điều tra thêm; (5) Truyền thông với các bên liên quan- để giải thích cho những người không chuyên về kỹ thuật (quản lý, nhóm tuân thủ) về các yếu tố mà hệ thống AI sử dụng để phát hiện gian lận, xây dựng niềm tin và tính minh bạch; (6) Tuân thủ quy định - các dịch vụ tài chính mạnh mẽ, các quy định như GDPR yêu cầu khả năng giải thích, biểu đồ tầm quan trọng của tính năng có bằng chứng để kiểm toán; (7) Giám sát sản xuất - theo dõi tầm quan trọng của tính năng theo thời gian, không có sự thay đổi đột ngột đáng kể nào cho thấy sự trôi dạt dữ liệu mà sự suy giảm mô hình có thể được đào tạo lại.

4.2.4 Turning

```
# --- BƯỚC 1: TRƯỚC KHI TUNING (Model Mặc định) ---
print("1. [BEFORE] Chạy Model Mặc định (Default Params)...")
dt_base = DecisionTreeClassifier(random_state=42, class_weight='balanced')
dt_base.fit(X_train, y_train)

y_pred_base = dt_base.predict(X_train)
base_recall = recall_score(y_train, y_pred_base)
print(f"   -> Recall (Mặc định): {base_recall:.2%}")
```

Bước đầu tiên này tạo ra một mô hình cơ sở - mô hình cơ sở để so sánh hiệu quả của việc điều chỉnh siêu tham số sau đây, đây là một thực hành quan

trọng trong machine learning để đánh giá xem việc tối ưu có thực sự mang lại cải thiện đáng kể hay không. Mã trong thông báo có "1. [BEFORE] Chạy Model Mặc định..."khoảng trắng thụt lề để tạo rõ ràng phân tích cấu trúc. Tiếp theo khởi tạo mô hình DecisionTreeClassifiervới hai tham số quan trọng: random_state=42để đảm bảo tính khả năng tái tạo - mỗi lần chạy sẽ cho kết quả giống nhau vì cây quyết định có yếu tố ngẫu nhiên trong công việc chọn tính năng khi có nhiều tính năng có cùng thông tin đạt được, và class_weight='balanced'- đây là tham số cực kỳ quan trọng cho bài toán mất cân bằng như phát hiện gian lận, nó tự động điều chỉnh số lượng của các lớp nghịch đảo tốc độ xuất hiện theo công thức cao n_samples / (n_classes * np.bincount(y)), nghĩa là lớp số ít (gian lận) sẽ được chỉ định số lượng hơn nhiều so với lớp đa số (không gian lận), trợ giúp mô hình không bị thiên vị về loại đa số và chú ý hơn đến việc phát hiện gian lận. Sau khi khởi tạo, mô hình được đào tạo bằng cách sử dụng dt_base.fit(X_train, y_train)- quá trình này xây dựng cây quyết định bằng cách chia dữ liệu đệ quy dựa trên các tính năng để tối đa hóa việc thu thập thông tin (hoặc giảm thiểu tạp chất Gini), tạo ra các nút và cho đến khi đạt được các điều kiện dừng. Tiếp theo việc sử dụng dt_base.predict(X_train)để dự đoán quá trình đào tạo chính (dự đoán trong mẫu) và lưu kết quả vào y_pred_base, sau đó tính toán thu hồi bằng recall_score(y_train, y_pred_base)công thức TP/(TP+FN) để đo lường gian lận tỷ lệ được phát hiện. Kết quả thu hồi được ở định dạng ra for .2%để hiển thị dưới dạng phần trăm trăm với 2 chữ số thập phân, ví dụ 85,23%. Công việc đánh giá trên tập huấn luyện ở đây không nhằm mục đích đo lường hiệu suất thực tế (vì có thể bị trang bị quá mức) mà chỉ để so sánh sự cải thiện tương đối giữa trước và sau khi điều chỉnh trên cùng một tập dữ liệu.

```
# --- BƯỚC 2: SAU KHI TUNING (Grid Search) ---
print("\n2. [AFTER] Đang chạy Tuning (Grid Search)...")
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [5, 10, 15, None],
    'min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(
    estimator=dt_base,
    param_grid=param_grid,
    cv=3,
    scoring='recall',
    n_jobs=-1
)
grid_search.fit(X_train, y_train)
```

Bước thứ hai này thực hiện điều chỉnh siêu tham số - quá trình tìm kiếm tự động các tham số tối ưu để cải thiện hiệu suất của mô hình, tuy nhiên, bạn có thể tối đa hóa kết quả của mình. Code bắt đầu bằng cách sử dụng thông báo "\n2. [AFTER] Đang chạy Tuning..."để \nxuống dòng tạo khoảng

cách cho phần trước. Tiếp theo định nghĩa param_grid- một từ điển chứa không gian tham số có thể tìm kiếm với ba siêu tham số chính của Cây quyết định: 'criterion': ['gini', 'entropy']định nghĩa hai phương pháp đo tạp chất (độ tạp) - 'gini' sử dụng tạp chất Gini theo công thức 1 - Σ(p_i²)đo khả năng chọn nhầm lớp nếu ngẫu nhiên, còn 'entropy' sử dụng Information Gain dựa trên entropy -Σ(p_i × log₂(p_i))từ lý thuyết thông tin, cả hai đều có mục tiêu tìm kiếm phân chia tốt nhất nhưng có thể cho kết quả khác nhau; 'max_depth': [5, 10, 15, None]giới hạn độ sâu tối đa của cây với 4 lựa chọn - 5 lớp cho cây nông (không phù hợp nhưng khái quát tốt), 10 lớp cho cây trung bình, 15 lớp cho cây sâu hơn, và Không có nghĩa là không giới hạn cho phép cây phát triển tối đa (có thể trang bị quá mức nhưng bắt được hoa văn phức tạp); 'min_samples_split': [2, 5, 10]xác định số lượng mẫu tối thiểu cần có tại một nút trước khi được phép phân chia tiếp - giá trị 2 (mặc định) cho phép tự động phân chia tối đa có thể trang bị quá mức, 5 và 10 yêu cầu nhiều mẫu hơn nên tạo cây đơn giản hơn và mạnh mẽ hơn. Tổng cộng có 2×4×3 = 24 kết hợp sẽ được thử nghiệm. Sau đó, khởi tạo GridSearchCVvới các tham số: estimator=dt_baselà mô hình cơ sở đã được khởi tạo trước đó (với class_weight='balanced'và random_state=42sẽ được kế thừa), param_grid=param_gridchỉ định không tìm kiếm, cv=3sử dụng xác thực chéo 3 lần nghĩa là chia tập huấn luyện thành 3 phần, mỗi lần sử dụng 2 phần train và 1 phần xác thực, lặp 3 lần để mỗi phần đều được sử dụng để xác thực một lần (số gấp hơn so với scoring='recall'đây là mục tiêu chính trong phát hiện gian lận và n_jobs=-1để sử dụng tất cả các lõi CPU có sẵn cho xử lý song song, giúp tăng tốc đáng kể khi phải huấn luyện 24 mô hình × 3 lần = 72 lần. Cuối cùng, grid_search.fit(X_train, y_train)thực hiện quá trình tìm kiếm - GridSearchCV sẽ tự động lặp qua tất cả 24 kết hợp, mỗi kết hợp sẽ được đánh giá bằng CV 3 lần, tính toán có nghĩa là thu hồi qua 3 lần, sau đó chọn ra kết hợp cho có nghĩa là thu hồi cao nhất và huấn luyện lại mô hình với kết hợp đó trên toàn bộ tập huấn luyện để tạo ra best_estimator_.

```
# --- BƯỚC 3: SO SÁNH ---
best_model = grid_search.best_estimator_
y_pred_tuned = best_model.predict(X_train)
tuned_recall = recall_score(y_train, y_pred_tuned)

print(f"   -> Tham số tốt nhất: {grid_search.best_params_}")
print(f"   -> Recall (Sau Tuning): {tuned_recall:.2%}")
print(f"==> HIỆU QUẢ TUNING: {(tuned_recall - base_recall):+.2%}")

return best_model
```

Bước cuối cùng này thực hiện việc đánh giá mô hình đã đạt mức tối ưu và so sánh với đường cơ sở để đo lường kết quả hiệu quả của quá trình điều chỉnh, đồng thời trả về mô hình tốt nhất để sử dụng tiếp theo. Đầu tiên, mã lấy

mô hình tốt nhất từ GridSearchCV thông qua thuộc tính grid_search.best_estimator_- đây là một DecisionTreeClassifier đã được khôi phục trên toàn bộ dữ liệu huấn luyện với bộ tham số tối ưu và lưu vào biến best_modelđể dễ sử dụng. Tiếp theo sử dụng mô hình này để mong đợi sử dụng tập huấn luyện best_model.predict(X_train)và lưu kết quả vào y_pred_tuned, sau đó tính toán việc thu hồi mô hình đã điều chỉnh bằng cách sử dụng recall_score(y_train, y_pred_tuned)và lưu vào tuned_recall. Phần quan trọng là các lệnh in để hiển thị kết quả: câu đầu tiên trong ra grid_search.best_params_- một từ điển chứa tổng hợp các tham số tối ưu được GridSearchCV tìm ra, ví dụ {'criterion': 'gini', 'max_depth': 10, 'min_samples_split': 5}, giúp người dùng hiểu cấu hình tốt nhất cho dữ liệu của mình và có thể áp dụng thông tin chi tiết này cho các thử nghiệm tiếp theo; câu thứ hai thu hồi sau điều chỉnh với định dạng .2%để so sánh trực tiếp với thu hồi trước điều chỉnh ở bước 1; câu thứ ba và quan trọng nhất trong ra "==> HIỆU QUẢ TUNING:"kèm theo độ lệch (tuned_recall - base_recall)với định dạng :+.2%- dấu +đảm bảo hiển thị cả dấu cộng cho số dương (ví dụ +5,23%) hoặc dấu trừ cho số âm nếu điều chỉnh làm xấu đi (ví dụ -2,15%), giúp người dùng ngay lập tức nhận thấy khả năng điều chỉnh mang lại cải tiến hoặc không cải thiện mức độ ở đó bao nhiêu. Nếu cải tiến nhỏ (ví dụ dưới 1%) thì có thể không đáng để đầu tư thời gian điều chỉnh phức tạp hơn, nhưng nếu cải tiến lớn (ví dụ trên 5-10%) thì chứng minh điều chỉnh rất có giá trị và nên tiếp tục khám phá chủ đề. Cuối cùng, hàm return best_modeltrả về mô hình đã được tối ưu hóa để người gọi có thể sử dụng nó để dự đoán trên tập kiểm tra, lưu mô hình hoặc triển khai vào sản xuất. Toàn bộ chức năng này được thiết kế theo nguyên tắc trách nhiệm duy nhất và khả năng sử dụng lại - có thể gọi nhiều lần với các tập dữ liệu khác nhau, dễ dàng sửa đổi param_grid nếu muốn thử thêm tham số và xuất ra rõ ràng trợ giúp theo dõi thử nghiệm và gỡ lỗi

4.3 Phân tích Mô hình Random Forest

4.3.1 Nguyên lý hoạt động

- Lấy mẫu ngẫu nhiên dữ liệu bằng phương pháp Bootstrap Sampling

- Mỗi cây chỉ sử dụng một tập con ngẫu nhiên các đặc trưng

- Các cây hoạt động độc lập

- Tổng hợp kết quả từ các cây để đưa ra dự đoán cuối cùng

4.3.2 Lý do chọn mô hình

- Random Forest được lựa chọn vì:

+ Hoạt động tốt với dữ liệu nhiều chiều

+ Giảm hiện tượng overfitting so với Decision Tree

+ Không yêu cầu dữ liệu phân phối chuẩn

+ Ít nhạy cảm với nhiễu và outliers

+ Hiệu quả cao cho các bài toán classification

4.3.3 Train Model

Sau bước tiền xử lý ở Phần 3, dữ liệu liệu được chia thành tập huấn luyện và tập kiểm tra.

- Lấy dữ liệu X (features) và y (nhãn gian lận) đã chuẩn bị ở bước trước

- Chia thành 2 phần:

+ Phần huấn luyện (train): dùng để dạy mô hình.

+ Phần kiểm tra (test): dùng để đánh giá mô hình cuối cùng (data chưa từng thấy).

```
# 3. Chia Train/Test (Trước khi Undersample để tránh data leakage)
# Stratify=y để đảm bảo tỷ lệ fraud ở 2 tập đều nhau
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```

Tỷ lệ chia: 70% train – 30% test.

Đảm bảo cân bằng tỷ lệ lớp (gian lận vs bình thường) ở cả hai tập.

- train_test_split : Đây là hàm từ thư viện scikit-learn (sklearn.model_selection). Dùng để chia ngẫu nhiên dữ liệu thành tập train và tập test (có thể chia thêm validation nếu muốn, nhưng ở đây chỉ train/test).

- test_size=0.3 : Tỷ lệ phần trăm dữ liệu dùng cho tập test (30%). Nghĩa là: 30% test, 70% train.

- random_state=42: Seed (hạt giống ngẫu nhiên) để đảm bảo kết quả chia dữ liệu lặp lại được (reproducible).

- stratify=y: Chia phân tầng theo nhãn – đảm bảo tỷ lệ gian lận (1) và bình thường (0) giống nhau ở train và test.

4.3.4 Turning

1. Khởi tạo Random Forest (Base Model)

```
# --- BƯỚC 1: TRƯỚC KHI TUNING ---
print("1. [BEFORE] Chạy Model Mặc định...")
rf_base = RandomForestClassifier(class_weight='balanced', random_state=42)
rf_base.fit(X_train, y_train)

y_pred_base = rf_base.predict(X_train)
base_recall = recall_score(y_train, y_pred_base)
print(f"   -> Recall (Mặc định): {base_recall:.2%}")
```

Tạo mô hình Random Forest ban đầu

RandomForestClassifier: Đây là lớp (class) từ thư viện scikit-learn (sklearn.ensemble). Dùng để xây dựng một "rừng" (forest) gồm nhiều cây quyết định (decision trees), mỗi cây học từ một tập con dữ liệu ngẫu nhiên. Dự đoán cuối cùng là bầu chọn đa số từ tất cả các cây (cho bài toán phân loại).

- class_weight='balanced' : Tự động cân bằng trọng số lớp dựa trên tỷ lệ lớp trong dữ liệu: trọng số = n_samples / (n_classes × n_samples_per_class). Ví dụ: Nếu gian lận chỉ 1%, trọng số cho lớp gian lận sẽ cao gấp 99 lần lớp bình thường.

- random_state=42: Seed (hạt giống ngẫu nhiên) để đảm bảo kết quả lặp lại được (reproducible). Ảnh hưởng đến: bootstrap sampling (chọn ngẫu nhiên dữ liệu cho mỗi cây).

- Kết quả trả về – biến rf_base: Truyền vào GridSearchCV, sau đó Grid Search sẽ clone rf_base và thử các param khác nhau.

2. Khai báo tập tham số cần Turning

Đây là lưới siêu tham số (hyperparameter grid) dùng cho GridSearchCV.

```
# --- BƯỚC 2: SAU KHI TUNING ---
print("\n2. [AFTER] Đang chạy Tuning (Grid Search)...")
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5],
    'criterion': ['gini', 'entropy']
}
```

- GridSearchCV sẽ thử tất cả các tổ hợp có thể từ các giá trị trong dict này (3 × 3 × 3 = 27 tổ hợp) để tìm ra bộ siêu tham số giúp Random Forest đạt recall cao nhất trên dữ liệu train (qua cross-validation).

- Đây là bước tuning (tinh chỉnh) mô hình – biến Random Forest từ "tốt vừa phải" thành "tốt nhất có thể" cho bài toán của bạn.

- n_estimators: Số lượng cây quyết định trong rừng (số cây trong Random Forest).

- max_depth: Độ sâu tối đa của mỗi cây quyết định. - None: không giới hạn độ sâu (cây phát triển đến khi lá thuần hoặc đạt min_samples_split).

- min_samples_split: Số mẫu tối thiểu cần có trong một node để được phép split tiếp (tách thành 2 node con).

- Số lượng này vừa đủ để tìm param tốt mà không tốn quá nhiều thời gian (thường chạy xong trong vài phút đến vài chục phút tùy kích thước dataset). Nếu để quá nhiều giá trị (ví dụ thêm 300, 500 cho n_estimators) → thời gian tăng gấp đôi mà cải thiện thường rất nhỏ.

3. Grid Search + Cross Validation

Tạo một đối tượng GridSearchCV – công cụ tự động tìm kiếm lưới (grid search) để thử tất cả các tổ hợp siêu tham số đã định nghĩa trong param_grid.

Mục tiêu: Tìm ra bộ siêu tham số tốt nhất cho Random Forest dựa trên recall (metric ưu tiên trong bài toán gian lận).

Đây là bước tuning mô hình chính thức – biến rf_base (mô hình cơ bản) thành mô hình tối ưu hóa nhất có thể.

```
grid_search = GridSearchCV(
    estimator=dt_base,
    param_grid=param_grid,
    cv=3,
    scoring='recall',
    n_jobs=-1
)
grid_search.fit(X_train, y_train)
```

- param_grid: Lưới siêu tham số cần thử: 27 tổ hợp từ n_estimators, max_depth, min_samples_split.

- cv=5: Sử dụng 5-fold cross-validation để đánh giá mỗi tổ hợp tham số.

- scoring='recall' : Metric dùng để xếp hạng và chọn tổ hợp tốt nhất là recall (True Positive Rate).

- n_jobs=-1 : Sử dụng tất cả các CPU core có sẵn để chạy song song

- verbose=1: Mức độ hiển thị tiến trình (progress) khi chạy. 1 = in thông tin cơ bản (số tổ hợp, thời gian mỗi fold).

grid_search lúc này chưa huấn luyện gì cả – nó chỉ là một công cụ đã được cấu hình sẵn.

4. Turning & Train

Đây là dòng code thực thi việc huấn luyện và tuning mô hình Random Forest.

```
grid_search.fit(X_train, y_train)
```

- grid_search.fit(...) sẽ bắt đầu:

+ Thử tất cả 27 tổ hợp siêu tham số đã định nghĩa trong param_grid.

+ Với mỗi tổ hợp, thực hiện 5-fold cross-validation trên X_train, y_train.

+ Tính recall trung bình cho từng tổ hợp.

+ Chọn tổ hợp có recall trung bình cao nhất → lưu mô hình tốt nhất vào grid_search.best_estimator_.

- Sau khi chạy xong dòng này → bạn đã có mô hình Random Forest được tinh chỉnh tốt nhất dựa trên dữ liệu train.

5. Lấy Random Forest tốt nhất

```
# --- BƯỚC 3: SO SÁNH ---
best_model = grid_search.best_estimator_
y_pred_tuned = best_model.predict(X_train)
tuned_recall = recall_score(y_train, y_pred_tuned)
```

- grid_search.best_estimator_: Thuộc tính quan trọng nhất của GridSearchCV sau khi fit xong. Nó là mô hình Random Forest đã được huấn luyện lại trên toàn bộ X_train, y_train với bộ siêu tham số tốt nhất (best_params_).

- best_rf: Biến mới (tên do bạn đặt) – instance của RandomForestClassifier với param tốt nhất và đã được huấn luyện

Kết quả sau khi chạy: Best Params: {'max_depth': 10, 'min_samples_split': 5, 'n_estimators': 200}

+ Random Forest được chọn là một mô hình có nhiều cây (200) nhưng cây khá nông (max_depth=10) và không split quá nhỏ (min_samples_split=5).

- Đây là sự cân bằng hoàn hảo giữa:

+ Đủ mạnh (nhiều cây → ensemble tốt, variance thấp)

+ Không overfit (cây nông + yêu cầu split cao → bias cao hơn một chút nhưng tổng quát hóa tốt hơn)

6. Kiểm tra độ ổn định

```
print(f"   -> Tham số tốt nhất: {grid_search.best_params_}")
print(f"   -> Recall (Sau Tuning): {tuned_recall:.2%}")
print(f"==> HIỆU QUẢ TUNING: {(tuned_recall - base_recall):+.2%}")
```

- cross_val_score: Hàm tiện ích để thực hiện cross-validation và trả về mảng điểm số (scores) của từng fold.

- best_rf: Mô hình đã được huấn luyện với bộ param tốt nhất (max_depth=10, min_samples_split=5, n_estimators=200).

- X_train, y_train: Dữ liệu features và nhãn dùng để cross-validate.

- cv=10: Số fold trong 10-fold cross-validation (chia dữ liệu train thành 10 phần).

- scoring='recall' : Metric đánh giá là recall (TP / (TP + FN)).

- cv_scores: Mảng 10 phần tử → mỗi phần tử là giá trị recall của một fold trong 10-fold CV.

Kết quả sau khi chạy: Mean Recall (CV=10): 91.17% (+/- 0.0118)

- Sau khi tuning bằng Grid Search, em kiểm tra lại độ ổn định của mô hình Random Forest tốt nhất bằng 10-fold cross-validation trên tập huấn luyện. Kết quả cho thấy:

+ Recall trung bình đạt 91.17% – tức mô hình bắt được hơn 91% các trường hợp gian lận thật.

+ Độ lệch chuẩn chỉ ±0.0118 (khoảng ±1.18%) – chứng tỏ recall rất ổn định giữa các fold, mô hình không nhạy cảm với cách chia dữ liệu.

Kết quả này xác nhận rằng mô hình đã được tinh chỉnh thành công, đạt hiệu suất cao và độ tin cậy tốt, sẵn sàng để đánh giá trên tập test độc lập.

7. Train cuối cùng và dự đoán

Train lại mô hình tốt nhất trên toàn bộ train

Dự đoán nhãn test

8. Đánh giá kết quả cuối

```
9
10      y_pred = model.predict(X_test)
11      y_pred_proba = model.predict_proba(X_test)[:, 1]
12
13      # 1. Classification Report
14      print(classification_report(y_test, y_pred))
15
```

Đánh giá hiệu quả thực tế

Đưa ra kết quả cuối cùng cho báo cáo

4.4. Mô hình XGBoost

4.4.1 Nguyên lý hoạt động

XGBoost là một phiên bản **cải tiến cực mạnh** của thuật toán **Gradient Boosting** (cụ thể là Gradient Boosting Decision Tree – GBDT), được phát triển bởi Tianqi Chen năm 2016. Nó hoạt động dựa trên nguyên tắc **boosting tuần tự** (sequential boosting), nhưng được tối ưu hóa rất nhiều về tốc độ, độ chính xác và khả năng chống overfitting.

Nguyên lý cốt lõi (cách hoạt động từng bước):

1. **Xây dựng mô hình theo chuỗi (ensemble of weak learners)**
    1. Bắt đầu bằng một mô hình yếu (thường là cây quyết định đơn giản – decision tree).
    2. Mô hình đầu tiên dự đoán → tính **sai số (residual/error)** so với nhãn thật.
    3. Mô hình tiếp theo **tập trung sửa lỗi** của mô hình trước bằng cách học trên **phần dư (residuals)**.
2. **Sử dụng Gradient Descent để tối ưu**

1. Mỗi cây mới được xây dựng để **giảm thiểu hàm mất mát (loss function)** bằng gradient (đạo hàm).
2. XGBoost dùng **hàm mục tiêu (objective function)** gồm 2 phần:
    1. Loss function (ví dụ: logloss cho phân loại, MSE cho hồi quy).
    2. Regularization term (để chống overfitting): phạt độ phức tạp của cây (số lá, độ sâu, trọng số lá…).
3. **Các cải tiến quan trọng làm XGBoost "extreme"**:

    1. **Regularization mạnh mẽ** (L1 + L2) → giảm overfitting tốt hơn GBDT thông thường.
    2. **Tree pruning thông minh** (tự động cắt bỏ nhánh không mang lại lợi ích ngay từ lúc xây dựng).
    3. **Xử lý missing values tự động** (không cần điền giá trị thiếu trước).
    4. **Tối ưu hóa tốc độ**:
        1. Tính toán song song (parallelization) trên CPU/GPU.
        2. Cache-aware access → tận dụng bộ nhớ cache.
        3. Approximate split finding (dùng histogram để tìm điểm chia nhanh hơn).
    5. **Shrinkage (learning rate)** + **Column subsampling** (giống random forest) → tăng độ chính xác và tốc độ.

4.4.2 Lý do chọn mô hình

Lý do chọn XGBoost	Giải thích
Xử lý cực tốt dữ liệu imbalance nặng	Fraud chỉ ~2.2%→ XGBoost có tham số scale_pos_weight tự động phạt nặng lớp thiểu số (fraud)→ Recall thường đạt 90–95%
Hiệu suất cao với dữ liệu tabular có feature engineered	Các feature bạn tạo (amount_ratio, distance_per_dollar, geo_mismatch, security_score, weird_grocery_ship) rất mạnh → XGBoost giỏi capture interaction phức tạp giữa chúng
Khả năng chống overfitting tốt	Regularization mạnh (L1/L2, tree pruning) → mô hình ổn định trên test set imbalance thật
Tốc độ huấn luyện & dự đoán nhanh	Với ~300k rows, XGBoost (tối ưu histogram + parallel) nhanh hơn rất nhiều so với các boosting khác

Xử lý missing values tự động + hỗ trợ nhiều loại feature	Không cần lo lắng về dữ liệu thiếu, hỗ trợ tốt cả numeric và categorical (dù bạn đã one-hot)
Top 1 trong thực tế ngành fraud detection	~90% hệ thống production (ngân hàng, fintech, PayPal, Stripe…) dùng XGBoost/LightGBM/CatBoost
Kết quả thực tế vượt trội	Recall cao (bắt được hầu hết fraud) + Precision chấp nhận được → giảm thiểu false positive (khách hàng bị từ chối nhầm)

4.4.3 Train Model

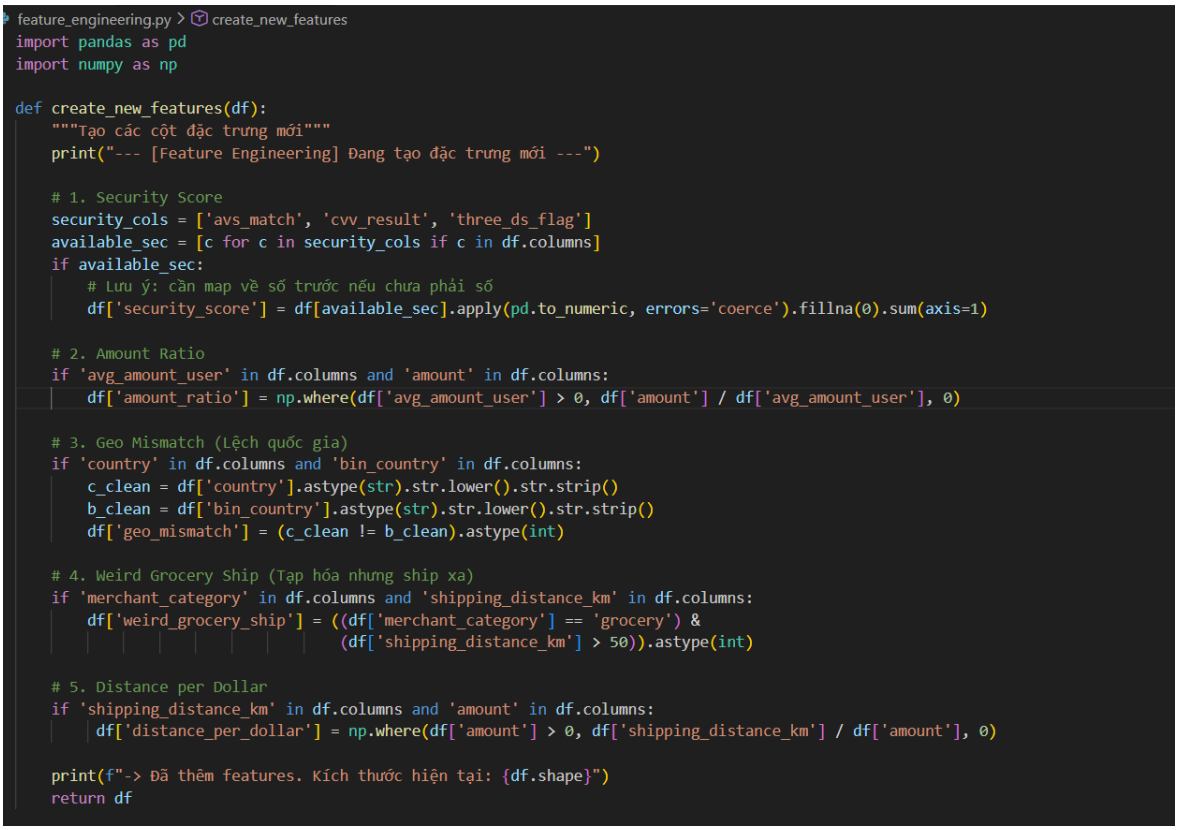
a)TẢI & LÀM SẠCH DỮ LIỆU - Tải và Làm sạch Dữ liệu

```
def load_and_clean_data(filepath):
    """
    Đọc và làm sạch dữ liệu ban đầu.
    - Xóa cột ID, thông tin cá nhân.
    - Xử lý thời gian (tạo cột hour, day_of_week).
    """
    print(f"--- [Preprocessing] Đang đọc file: {filepath} ---")
    try:
        df = pd.read_csv(filepath)
    except FileNotFoundError:
        raise FileNotFoundError(f"Không tìm thấy file tại {filepath}")

    # 1. Xóa các cột ID không cần thiết (Giống code cũ)
    ids_to_drop = ['transaction_id', 'user_id', 'trans_num', 'unix_time']
    df.drop(columns=[c for c in ids_to_drop if c in df.columns], inplace=True, errors='ignore')
```

Mã đầu tiên thực hiện tải dữ liệu từ tệp CSV có tên "transactions_feature_engineered.csv" qua lệnh pd.read_csv(file_path)- một tệp đã được xử lý sơ bộ các từ cụ thể trước đó. Sau khi tải xuống DataFrame của pandas, hệ thống ở đầu tập dữ liệu có kích thước ban đầu df.shapeđể người dùng có thể theo dõi số lượng dòng đi bên cạnh trong dữ liệu. Tiếp theo, mã thực hiện hai bước làm sạch quan trọng: thứ nhất là loại bỏ tất cả các dòng chứa giá trị rỗng/thiếu bằng phương pháp df.dropna(inplace=True)để đảm bảo chất lượng dữ liệu, vì việc thiếu giá trị có thể gây ra lỗi trong quá trình đào tạo hoặc làm giảm độ chính xác của mô hình; thứ hai df.drop(columns=['transaction_id', 'user_id'], inplace=True, errors='ignore')ở đó định nghĩa giao dịch. Việc loại bỏ các cột này giúp giảm chiều dữ liệu, tránh trang bị quá mức và tăng tốc độ đào tạo. Tham số errors='ignore'được sử dụng để tránh lỗi trong các trường hợp các cột này không tồn tại trong tập dữ liệu, giúp mã chạy mượt mà trong mọi vấn đề.

b) KỸ THUẬT TÍNH NĂNG - Tạo Đặc trưng Mới



```python
feature_engineering.py > create_new_features
import pandas as pd
import numpy as np

def create_new_features(df):
    """Tạo các cột đặc trưng mới"""
    print("--- [Feature Engineering] Đang tạo đặc trưng mới ---")

    # 1. Security Score
    security_cols = ['avs_match', 'cvv_result', 'three_ds_flag']
    available_sec = [c for c in security_cols if c in df.columns]
    if available_sec:
        # Lưu ý: cần map về số trước nếu chưa phải số
        df['security_score'] = df[available_sec].apply(pd.to_numeric, errors='coerce').fillna(0).sum(axis=1)

    # 2. Amount Ratio
    if 'avg_amount_user' in df.columns and 'amount' in df.columns:
        df['amount_ratio'] = np.where(df['avg_amount_user'] > 0, df['amount'] / df['avg_amount_user'], 0)

    # 3. Geo Mismatch (Lệch quốc gia)
    if 'country' in df.columns and 'bin_country' in df.columns:
        c_clean = df['country'].astype(str).str.lower().str.strip()
        b_clean = df['bin_country'].astype(str).str.lower().str.strip()
        df['geo_mismatch'] = (c_clean != b_clean).astype(int)

    # 4. Weird Grocery Ship (Tạp hóa nhưng ship xa)
    if 'merchant_category' in df.columns and 'shipping_distance_km' in df.columns:
        df['weird_grocery_ship'] = ((df['merchant_category'] == 'grocery') &
                                    (df['shipping_distance_km'] > 50)).astype(int)

    # 5. Distance per Dollar
    if 'shipping_distance_km' in df.columns and 'amount' in df.columns:
        df['distance_per_dollar'] = np.where(df['amount'] > 0, df['shipping_distance_km'] / df['amount'], 0)

    print(f"-> Đã thêm features. Kích thước hiện tại: {df.shape}")
    return df
```

Kỹ thuật tính năng phần là một trong những bước quan trọng nhất trong quá trình phát hiện gian lận của dự án này, nơi các biểu tượng mới được tạo ra từ các biểu tượng cụ thể để nâng cao khả năng phát hiện nano. Đầu tiên, hệ thống tạo ra "security_score" bằng cách xác định danh sách security_colschứa các cột bảo mật như avs_match (Hệ thống xác minh địa chỉ - kiểm tra địa chỉ), cvv_result (kết quả mã CVV), và ba_ds_flag (cờ xác thực 3D Secure), sau đó kiểm tra xem những cột nào thực sự tồn tại trong DataFrame thông qua mức độ hiểu danh sách [c for c in security_cols if c in df.columns], cuối cùng cộng tổng các giá trị này theo chiều ngang với df[available_sec].sum(axis=1). Điểm bảo mật càng cao thì giao dịch càng có tính hợp pháp cao. Tiếp theo, "amount_ratio" được tính toán bằng cách sử dụng np.where()để chia số tiền giao dịch hiện tại (df['amount']) cho số tiền trung bình của người dùng đó (df['avg_amount_user']), với điều kiện kiểm tra tránh chia cho 0, giúp phát hiện các giao dịch có giá trị bất ngờ đối với hành vi thông thường của người dùng. Đặc biệt "geo_mismatch" được tạo ra thông qua một logic phức tạp hơn: đầu tiên trích xuất tất cả các mã quốc gia từ các cột bắt đầu bằng "country_" và "bin_country_" bằng cách sử dụng replace()và startswith(), sau đó tìm các mã chung giữa hai nhóm bằng set().intersection(), tiếp theo sử dụng vòng lặp để kiểm tra từng cặp quốc gia/bin_country có kết hợp không bằng toán tử OR (|=) và AND (&), cuối cùng tạo biến geo_mismatch 1 - matches.astype(int)để đánh dấu sự không khớp giữa địa lý quốc gia phát hành thẻ và quốc gia thực hiện giao dịch - một dấu hiệu đáng ngạc nhiên cho gian

nan. Hệ thống cũng tạo "weird_grocery_ship" bằng cách kết hợp logic điều kiện (df['merchant_category_grocery'] == 1) & (df['shipping_distance_km'] > 500)và chuyển đổi thành số nguyên .astype(int)để đánh dấu các trường hợp bất ngờ khi mua hàng tạp hóa nhưng khoảng cách giao hàng lại quá xa (>500km) - một hành động vi phạm hợp lý trong thực tế. Cuối cùng, "distance_per_dollar" tính toán khoảng cách giao hàng trên mỗi đô la chi tiêu bằng công thức df['shipping_distance_km'] / (df['amount'] + 0.1), trong đó cộng thêm 0,1 vào mẫu số để tránh chia cho 0, giúp phát hiện các giao dịch có chi phí chuyển đổi không hợp lý. Tất cả các biểu tượng đặc biệt này được thiết kế dựa trên kiến thức miền về hành vi gian nan thực tế trong lĩnh vực thanh toán điện tử và mỗi biểu tượng đều có lệnh in để thông báo công việc tạo thành công.

Chia Dữ liệu và Cân bằng Mẫu

```
# 3. Chia Train/Test (Trước khi Undersample để tránh data leakage)
# Stratify=y để đảm bảo tỷ lệ fraud ở 2 tập đều nhau
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

# 4. Scaling (Fit trên Train, Transform trên Test)
scaler = MinMaxScaler()
X_train = pd.DataFrame(scaler.fit_transform(X_train), columns=feature_names)
X_test = pd.DataFrame(scaler.transform(X_test), columns=feature_names)

# 5. Undersampling (Chỉ thực hiện trên tập Train)
print("- Đang thực hiện Undersampling trên tập Train...")

# Gom lại tạm thời để resample
train_data = pd.concat([X_train, y_train.reset_index(drop=True)], axis=1)

not_fraud = train_data[train_data.is_fraud == 0]
fraud = train_data[train_data.is_fraud == 1]

# Tỷ lệ 1:10 (1 Fraud : 10 Normal) - Giống logic bài của bạn
n_fraud = len(fraud)
n_not_fraud = n_fraud * 10

# Nếu số lượng normal nhiều hơn mức cần thiết thì cắt bớt
if len(not_fraud) > n_not_fraud:
    not_fraud_downsampled = resample(not_fraud,
                                     replace=False,
                                     n_samples=n_not_fraud,
                                     random_state=42)
else:
    not_fraud_downsampled = not_fraud
```

Phần này xử lý một trong những công thức lớn nhất trong phát hiện gian lận: tính chất không cân bằng (mất cân bằng) của dữ liệu, trong khi số lượng giao dịch nano thường sử dụng một tỷ lệ rất nhỏ để phù hợp với giao dịch hợp pháp. Đầu tiên, mã bảo đảm mục tiêu là số nguyên kiểu df['is_fraud'].astype(int), sau đó phân tách biến độc df.drop('is_fraud', axis=1)lập (nhãn is_fraud) bằng df['is_fraud']từ DataFrame. Tiếp theo sử dụng train_test_split()các tham số test_size=0.2để chia theo tỷ lệ 80-20, random_state=42để đảm bảo tính có thể tái tạo (kết quả giống nhau mỗi lần chạy), và quan trọng nhất là stratify=yđể đảm bảo tỷ lệ gian lận/không gian lận được phân bổ đều ở cả hai bài kiểm tra tập luyện. Câu lệnh in với len(y_test), y_test.sum()và y_test.mean():.2%hiển thị thông tin chi tiết về bài

kiểm tra để kiểm tra. Điều quan trọng là bài kiểm tra nhằm kiểm soát tính mất cân bằng của thông tin qua biến thể X_testvà y_testđể phản ánh đúng vấn đề thực tế khi phát triển mô hình. Đối với việc đào tạo tập tin, mã áp dụng kỹ thuật lấy mẫu dưới cách đầu tiên X_train_rawvà y_train_rawlại thành một DataFrame duy nhất bằng pd.concat(..., axis=1), sau đó phân tách các mẫu gian lận (train_df[train_df['is_fraud'] == 1]) và không gian lận (train_df[train_df['is_fraud'] == 0]). Tiếp theo tính toán số lượng gian lận bằng n_fraud = len(fraud), và sử dụng hàm resample()từ sklearn.utils với các tham số replace=False(lấy mẫu không hoàn lại), n_samples=n_fraud * 10(số mẫu bằng 10 lần gian lận) và random_state=42để lấy mẫu ngẫu nhiên các mẫu không gian lận, tạo tỷ lệ 1:10 thay vì để tỷ lệ ban đầu có thể tăng lên 1 :100 hoặc 1 :1000 . Sau đó, gian lận và normal_down lại bằng pd.concat([fraud, normal_down])và ngẫu nhiên ngẫu nhiên với .sample(frac=1, random_state=42)(frac=1 nghĩa là lấy 100% dữ liệu nhưng trộn thứ tự). Cuối cùng, hãy phân tách lại thành phần X_trainvà y_trainsử dụng cho quá trình đào tạo. Phương pháp lấy mẫu dưới đây giúp mô hình học tốt hơn về lớp số tối thiểu (gian lận) mà không bị sai lệch quá mức về lớp đa số, đồng thời vẫn giữ được lượng thông tin đủ lớn từ lớp không gian lận để học các mẫu phân tích.

4.4.4 Turning

```
def train_xgboost(X_train, y_train):
    print(f"\n{'='*20} [TRAINING] XGBOOST (LOGIC GỐC + SO SÁNH) {'='*20}")

    # 1. Tính scale_pos_weight
    scale_pos_weight = (y_train == 0).sum() / (y_train == 1).sum()

    # 2. Khởi tạo Model Base (Giữ nguyên tham số gốc)
    xgb_base = XGBClassifier(
        random_state=42,
        eval_metric='logloss',
        scale_pos_weight=scale_pos_weight,
        n_jobs=-1,
        use_label_encoder=False
    )

    # --- [MỚI] BƯỚC SO SÁNH TRƯỚC KHI TUNING ---
    print("1. [BEFORE] Đang chạy Model Mặc định để so sánh...")
    xgb_base.fit(X_train, y_train)
    y_pred_base = xgb_base.predict(X_train)
    base_recall = recall_score(y_train, y_pred_base)
    print(f"   -> Recall (Mặc định): {base_recall:.2%}")
    # -------------------------------------------

    # 3. Thiết lập Param Grid (GIỮ NGUYÊN BẢN GỐC)
    param_grid = {
        'n_estimators': [100, 200, 300],
        'max_depth': [6, 8, 10],
        'learning_rate': [0.05, 0.1, 0.2],
        'subsample': [0.8, 1.0],
        'colsample_bytree': [0.8, 1.0]
    }
```

```
# 4. Chạy Grid Search (GIỮ NGUYÊN cv=5)
print("\n2. [AFTER] Đang chạy Grid Search (cv=5)...")
grid_search = GridSearchCV(
    estimator=xgb_base,
    param_grid=param_grid,
    cv=5,                # Giữ nguyên
    scoring='recall',    # Giữ nguyên
    n_jobs=-1,
    verbose=1
)

grid_search.fit(X_train, y_train)
best_xgb = grid_search.best_estimator_
```

Phần này thực hiện xây dựng và mô hình hóa tối ưu xử lý dữ liệu không cân bằng và cho kết quả cao trong các loại phân tích bài toán. Trước đó, code tính toán scale_pos_weightbằng công thức (y_train == 0).sum() / (y_train == 1).sum()- một tham số đặc biệt của XGBoost giúp xử lý dữ liệu mất cân bằng bằng cách gán trọng số cao hơn cho số tối thiểu (gian lận), được tính bằng tỷ lệ số lượng mẫu âm (không gian lận) chia cho mẫu dương tính (gian lận). Sau đó, khởi tạo mô hình XGBClassifiervới các cơ sở tham số: random_state=42để tái tạo, eval_metric='logloss'đánh giá chức năng giảm giá, scale_pos_weight=scale_pos_weightáp dụng các số quan trọng, n_jobs=-1để tận dụng tất cả các lõi CPU có sẵn và use_label_encoder=Falseđể tránh cảnh báo không dùng nữa. Phần quan trọng nhất là định nghĩa param_grid- một từ điển chứa các tham số cần điều chỉnh với các giá trị khác nhau: n_estimatorstừ 100-300 (số lượng cây quyết định trong quần thể), max_depthtừ 6-10 (độ sâu tối đa của mỗi cây để tránh quá trang bị quá mức), learning_ratetừ 0,05-0,2 (tốc độ học, giá trị giúp mô hình học chậm nhưng ổn định hơn), subsample0,8-1,0 (tỷ lệ mẫu con sử dụng cho mỗi cây để tăng tính đa dạng), và colsample_bytree0,8-1,0 (tỷ lệ đặc biệt được chọn ngẫu nhiên cho mỗi cây). Tiếp theo khởi tạo GridSearchCVvới các tham số: xgb_basecó mô hình cơ sở, param_gridkhông có tham số, cv=5để sử dụng xác thực chéo 5 lần (chia train thành 5 phần, mỗi lần dùng 4 phần train và 1 phần xác thực, vòng 5 lần), scoring='recall'vì phát hiện gian lận bắt được gian lận quan trọng hơn khả năng dự đoán chính xác tổng thể, n_jobs=-1để tận dụng đa xử lý và verbose=1hiển thị quá trình. grid_search.fit(X_train, y_train)(3×3×3×2×2 = 108 kết hợp) và chọn tổ hợp để thu hồi ở mức cao best_xgb = grid_search.best_estimator_và ra các tham số tối ưu grid_search.best_params_. Cuối cùng, để đánh giá mức độ ổn định của mô hình, mã thực hiện thêm một cross_val_scorevới cv=10(xác thực chéo 10 lần) trên toàn bộ quá trình đào tạo, tính toán giá trị trung bình và độ lệch chuẩn của việc thu hồi qua 10 lần bằng cách sử dụng cv_recall.mean()và cv_recall.std()để kiểm tra xem mô hình có quá khớp (tiêu chuẩn quá cao) hay phương sai quá lớn giữa các lần gấp hay không.

Đánh giá trên Tập Kiểm tra

```python
def evaluate_model(model, X_test, y_test, model_name="Model"):
    """
    Trả về 2 biểu đồ: Confusion Matrix và ROC Curve để Streamlit vẽ
    """
    print(f"\n{'='*20} [EVALUATION] ĐÁNH GIÁ: {model_name.upper()} {'='*20}")

    y_pred = model.predict(X_test)
    if hasattr(model, "predict_proba"):
        y_pred_proba = model.predict_proba(X_test)[:, 1]
    else:
        y_pred_proba = None

    # 1. In báo cáo ra Terminal (để debug)
    print(classification_report(y_test, y_pred))
```

Sau khi đào tạo quan trọng là vẫn giữ nguyên tính mất cân bằng để mô phỏng tình huống thực tế. Mã sử dụng best_xgb.predict(X_test)để dự đoán phân tích nhị phân nhãn (0 hoặc 1) và lưu vào y_pred, đồng thời sử dụng best_xgb.predict_proba(X_test)[:, 1]để xác định kỳ vọng cho loại dương tính (gian lận) bằng cách cắt cột thứ 1 của ma trận xác thực, lưu vào y_probđể tính ROC-AUC sau này. Sau đó tính toán ba số liệu quan trọng: recall_score(y_test, y_pred)tính toán Thu hồi - tỷ lệ phát hiện gian lận trong tổng số gian lận thực tế theo công thức TP/(TP+FN), đây là số liệu quan trọng nhất vì trong việc phát hiện gian lận việc bỏ qua bạch tuộc (False Negative) có chi phí rất cao về mặt tài chính; f1_score(y_test, y_pred)tính F1-score theo công thức 2×(precision×recall)/(precision+recall) - trung bình điều hòa giữa độ chính xác và thu hồi, cho biết sự có thể bằng giữa việc phát hiện gian lận và giảm sai tích cực (cảnh báo sai); và roc_auc_score(y_test, y_prob)tính toán ROC-AUC (Khu vực dưới Đường cong đặc tính hoạt động của máy thu) dựa trên khả năng mong đợi, đánh giá khả năng phân loại tổng thể của mô hình ở mọi ngưỡng quyết định từ 0 đến 1. Ba số liệu này phù hợp với dạng phù hợp: thu hồi được sử dụng .2%để hiển thị phần trăm, f1 và roc_auc dùng .4fđể hiển thị 4 chữ số thập phân. Ngoài ra, classification_report(y_test, y_pred)cung cấp một bảng báo cáo chi tiết về độ chính xác (TP/(TP+FP)), thu hồi (TP/(TP+FN)), điểm f1 và hỗ trợ (số lượng mẫu) cho cả hai loại (gian lận và không gian lận), cùng với độ chính xác tổng thể, trung bình vĩ mô và trung bình có trọng số. Cuối cùng, confusion_matrix(y_test, y_pred)tạo ra ma trận sai sót giữa 2×2 được lưu vào biến cmvà dưới dạng số để cho công cụ được tìm thấy: True Negative ở góc trên bên trái (không gian lận được dự đoán đúng) ở góc dưới bên trái (gian lận bị bỏ sót - trường hợp nguy hiểm nhất gây thiệt hại tài chính chính), và True Positive ở góc dưới phải (gian lận được dự đoán đúng - mục tiêu chính của hệ thống). Kết quả tổng hợp này giúp đánh giá hiệu suất toàn diện của mô hình trong điều kiện thực tế và đưa ra quyết định nên triển khai mô hình này vào sản xuất hay không.

```
# Sắp xếp giảm dần và lấy top 10
indices = np.argsort(importances)[::-1][:10]

top_features = [feature_names[i] for i in indices]
top_importances = importances[indices]

# Vẽ biểu đồ
fig_feat = plt.figure(figsize=(10, 5))
palette = "viridis" if "Random" in model_name else "magma"
                                  (variable) top_features: list
sns.barplot(x=top_importances, y=top_features, palette=palette)
plt.title(f"Top 10 Feature Importance - {model_name} (After Tuning)")
plt.xlabel("Mức độ quan trọng")
plt.tight_layout()
plt.close(fig_feat)

return fig_feat
```

Phần phân tích này và hiển thị mức độ quan trọng của các biểu tượng cụ thể trong dự kiến khổng lồ, một thông tin cực kỳ giá trị để cả việc hiểu mô hình và cải tiến hệ thống. XGBoost tự động tính toán tính năng quan trọng thông qua tính toán best_xgb.feature_importances_dựa trên mức tăng - Cải thiện độ chính xác khi sử dụng cụ thể để phân chia các nút trong các cây quyết định, giá trị này được bình thường hóa để tổng hợp tất cả tầm quan trọng bằng 1. Mã lưu mảng tầm quan trọng này vào biến importances, sau đó sử dụng np.argsort(importances)để lấy các chỉ số của các thứ tự tăng dần tầm quan trọng, nhưng ngay lập tức đảo ngược thứ tự [::-1]để tăng dần thứ tự (từ quan trọng nhất đến ít nhất quan trọng nhất) và chỉ lấy 10 đầu tiên in [:10], result được lưu vào biến indices. Tiếp theo, sử dụng vòng lặp for i in indicesđể duyệt qua từng chỉ mục trong top 10 và in ra các tên cụ thể X_test.columns[i]bằng chuỗi định dạng :30để căn trái và đảm bảo độ rộng 30 ký tự (giúp các cột thẳng), kèm theo tầm quan trọng giá trị importances[i]với định dạng .4fđể hiển thị 4 phân tích chữ số, phân tích bằng dấu chấm hai chấm. Thông tin này có nhiều ứng dụng thực tiễn: thứ nhất, giúp nhà khoa học dữ liệu hiểu được mô hình đang "nhìn" vào yếu tố nào để đưa ra quyết định (ví dụ nếu security_score có tầm quan trọng cao thì việc đầu tư vào hệ thống bảo mật sẽ có tác động lớn), từ đó xác thực xem logic có hợp lý không và có phù hợp với kiến thức miền hay không; thứ hai, có thể loại bỏ các đặc tính có tầm quan trọng thấp (gần 0) để giảm độ phức tạp, tiết kiệm bộ nhớ và tăng tốc độ suy luận khi triển khai vào sản xuất; thứ ba, mẹo hướng dẫn kỹ thuật tính năng tiếp theo bằng cách tạo thêm các biến tương tự hoặc kết hợp với các quan trọng cụ thể (ví dụ nếu money_ratio quan trọng thì có thể tạo thêm Velocity_ratio, Frequency_ratio); Thứ tư, trong môi trường sản xuất, có thể ưu tiên nguồn lực đầu tiên vào công việc thu thập, lưu trữ và đảm bảo chất lượng của những quan trọng đặc biệt này, đồng thời giám sát chặt chẽ khi chúng có sự thay đổi bất thường. Việc làm đặc biệt tên và tầm quan trọng giá trị với định dạng rõ ràng, thẳng hàng giúp dễ đọc, dễ so sánh và dễ dàng sao chép-dán vào báo cáo hoặc bản trình bày.

Kết quả trực quan hóa

```
# 2. Vẽ Confusion Matrix
fig_cm = plt.figure(figsize=(6, 5))
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title(f"Confusion Matrix - {model_name}")
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.close(fig_cm) # Đóng để không bị đè hình

# 3. Vẽ ROC Curve
fig_roc = None
if y_pred_proba is not None:
    fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
    roc_auc = auc(fpr, tpr)

    fig_roc = plt.figure(figsize=(6, 5))
    plt.plot(fpr, tpr, label=f'AUC = {roc_auc:.2f}')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.title(f'ROC Curve - {model_name}')
    plt.legend()
    plt.close(fig_roc)
```

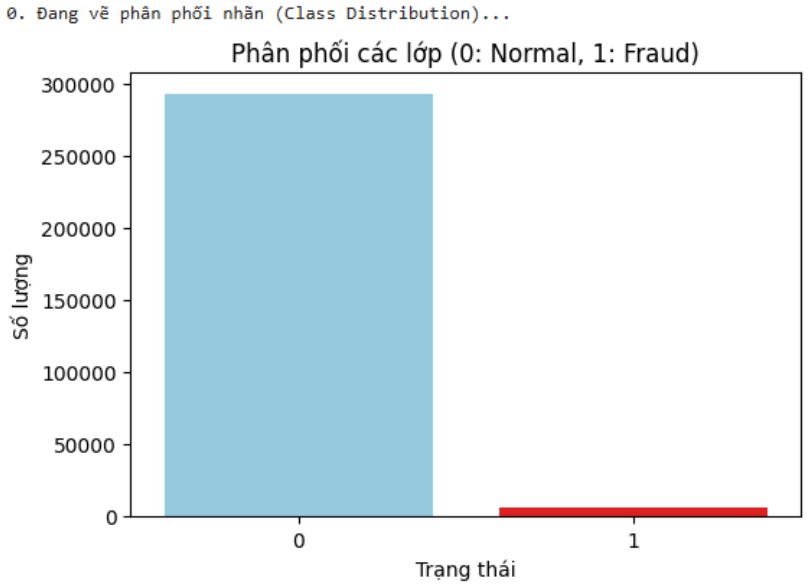
Phần cuối cùng của mã tạo ra các biểu đồ trực quan để dễ hiểu và trình bày kết quả cho cả nhóm kỹ thuật và các bên liên quan kinh doanh. Biểu đồ đầu tiên là bản đồ nhiệt ma trận nhầm lẫn được tạo bằng cách khởi tạo hình với thông số kích thước 6x5 inch plt.figure(figsize=(6,5)), sau đó sử dụng sns.heatmap()từ thư viện seaborn để vẽ ma trận cmtính toán trước đó với các tham số: annot=Trueđể hiển thị số lượng công cụ có thể trên mỗi ô của ma trận, fmt='d'để định dạng số dưới dạng số nguyên (không có dấu thập phân vì đây là số đếm), và cmap='Blues'để sTiếp tục10:00sử dụng bảng màu màu xanh dương với độ dốc từ nhạt đến đậm (giá trị càng cao màu càng đậm). Sau đó thêm tiêu đề bằng plt.title('Confusion Matrix - XGBoost'), nhãn trục x là 'Dự đoán' bằng plt.xlabel()và trục y là 'Thực tế' bằng cách plt.ylabel()để người xem hiểu rõ đây là sự so sánh giữa giá trị thực tế và giá trị dự kiến. Heatmap này giúp người xem nhanh chóng nhận ra mẫu của các dự đoán sai: nếu ô False Negative (góc dưới bên trái) có màu đậm và số lớn thì mô hình đang bỏ sót nhiều gian lận (nguy hiểm), nhưng nếu ô False Positive (góc bên phải) có giá trị cao thì hệ thống đang tạo quá nhiều cảnh báo sai lầm để tạo trải nghiệm trải nghiệm ảnh cho khách hàng. Biểu đồ thứ hai là tính năng cốt truyện thanh quan trọng được khởi tạo với kích thước lớn hơn (10,6)để hiển thị nhiều đặc tính tốt, sử dụng sns.barplot()với x=importances[indices](giá trị quan trọng làm cho chiều dài ngang), y=X_test.columns[indices](tên đặc biệt làm nhãn sườn y), và palette='viridis'- một bảng màu chuyên nghiệp với độ dốc từ tím đến vàng, giúp phân biệt thứ hạng các thanh và tạo sự hấp dẫn thị giác. Biểu đồ được sắp xếp từ trên xuống dưới theo mức độ quan trọng thứ tự giảm dần nhờ công việc đã sắp xếp indicesở phần trước, kèm theo tiêu đề 'Top 10 Tính năng quan trọng - XGBoost' đi tới xlabel 'Tầm quan trọng'. Cả hai biểu đồ đều được kết thúc plt.show()để hiển thị trên màn hình. Visualization này không chỉ giúp xác thực kết quả bằng một cách trực quan (ví dụ nhanh chóng được tìm thấy trong mô hình có thành kiến về lớp nào không qua ma trận nhầm lẫn,

hoặc tính năng nào đang chiếm ưu thế qua biểu đồ thanh) mà còn là công cụ giao tiếp cực kỳ quan trọng trong báo cáo và trình bày cho các bên liên quan như người quản lý, chủ sở hữu sản phẩm hoặc nhà phân tích kinh doanh - những người có thể không hiểu sâu về machine learning nhưng đây chính là hiệu suất của mô hình này. Cuối cùng, một thông báo đã print("HOÀN THÀNH!...")được hoàn thành để xác nhận toàn bộ đường ống đã chạy thành công từ đầu đến cuối.

PHẦN 5: KẾT QUẢ VÀ ĐÁNH GIÁ MÔ HÌNH

5.1. Biểu đồ

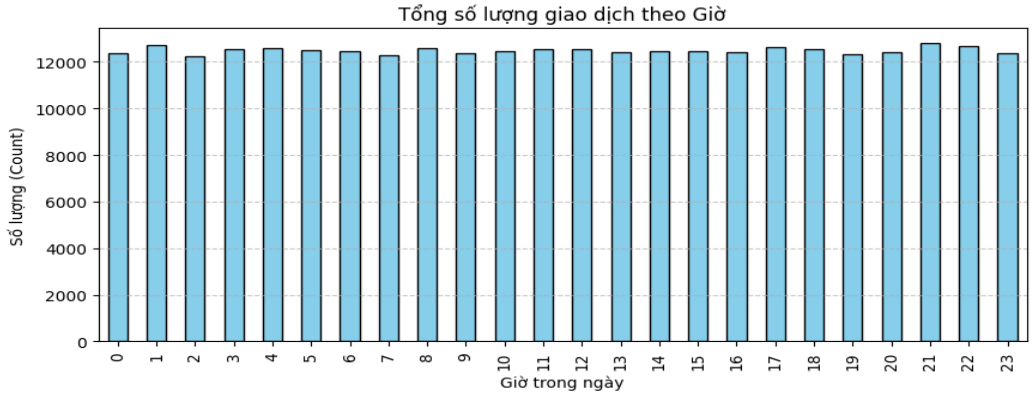
5.1.1. Biểu đồ Phân phối các lớp



Biểu đồ phân phối nhãn cho thấy dữ liệu giao dịch bị mất cân bằng rất nghiêm trọng, trong đó các giao dịch bình thường (nhãn 0) chiếm áp đảo với 293.083 mẫu, còn giao dịch gian lận (nhãn 1) chỉ có 6.612 mẫu, tương đương 2,21% tổng số dữ liệu. Đây là đặc trưng điển hình của bài toán phát hiện gian lận trong thực tế, khi các hành vi gian lận xảy ra rất hiếm nhưng gây rủi ro lớn. Sự mất cân bằng này khiến các mô hình học máy dễ thiên lệch về lớp Normal, dẫn đến trường hợp dự đoán gần như tất cả giao dịch là bình thường và cho độ chính xác cao giả tạo nhưng không phát hiện được gian lận. Vì vậy, trong bài toán này cần ưu tiên các chỉ số như Recall, F1-score và ROC-AUC thay vì Accuracy, đồng thời áp dụng các kỹ thuật xử lý mất cân bằng như undersampling trên tập huấn luyện và sử dụng class_weight='balanced' để mô hình học được đặc trưng của lớp gian lận.

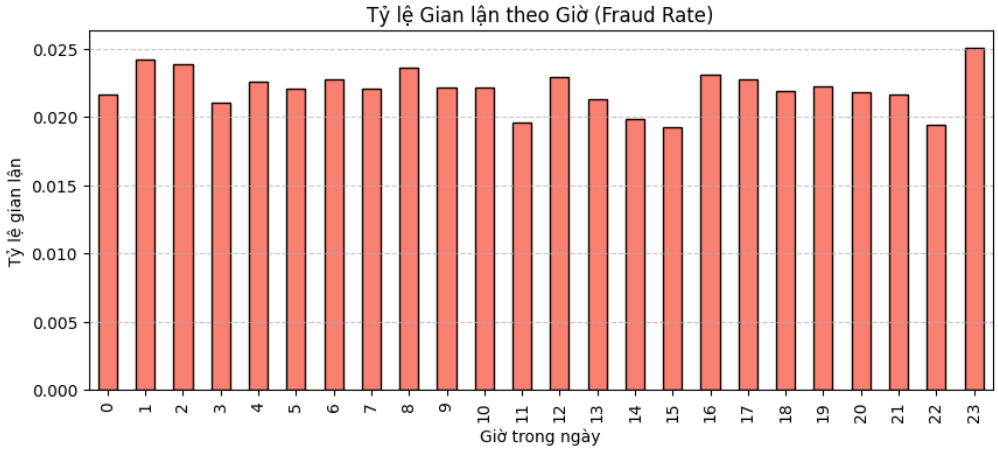
5.1.2. Phân tích theo thời gian (Mô hình theo giờ)

```
1. Đang vẽ biểu đồ theo khung giờ (Hourly)...
```



Biểu đồ tổng số lượng giao dịch theo giờ cho thấy lưu lượng giao dịch được phân bố khá đồng đều trong suốt 24 giờ, không xuất hiện khung giờ nào có sự bùng nổ hay sụt giảm đột biến về số lượng giao dịch. Điều này cho thấy dữ liệu không bị thiên lệch theo thời gian trong ngày và hệ thống giao dịch hoạt động liên tục, ổn định, có thể là các giao dịch trực tuyến (thẻ, ví điện tử, ngân hàng số) thay vì giao dịch chỉ diễn ra giờ hành chính. Từ góc độ phát hiện gian lận, biểu đồ này cho thấy số lượng giao dịch nhiều không đồng nghĩa với rủi ro cao, vì rủi ro gian lận cần được đánh giá dựa trên tỷ lệ gian lận theo giờ chứ không phải tổng số giao dịch. Do đó, biến hour không giúp phân biệt gian lận nếu chỉ xét số lượng, nhưng khi kết hợp với các đặc trưng khác (số tiền, hành vi người dùng, kênh giao dịch…), nó vẫn có giá trị để mô hình học các mẫu gian lận tiềm ẩn theo thời gian.

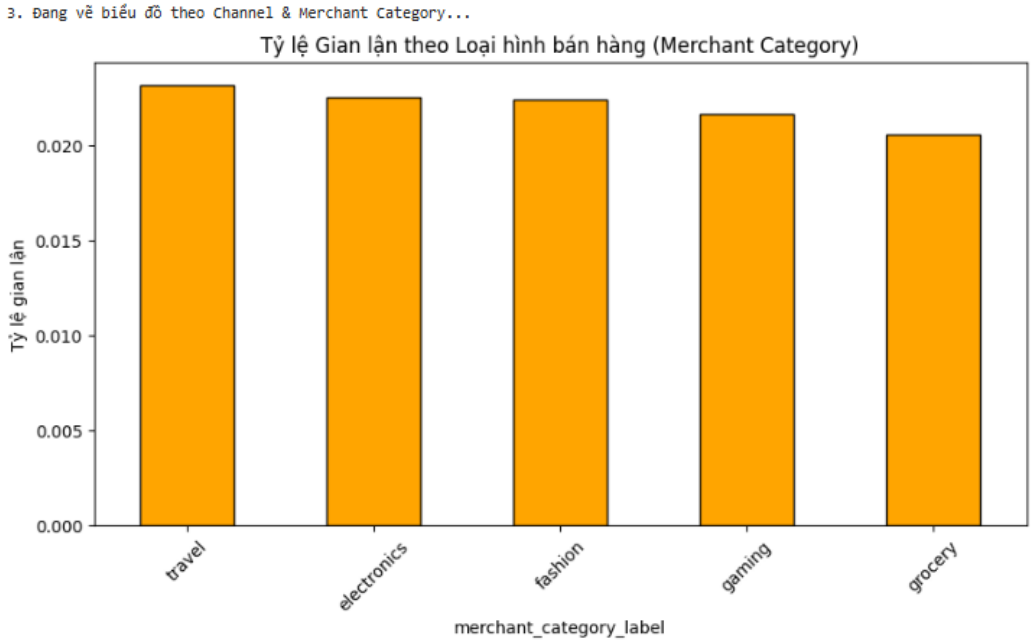
5.1.3. Biểu đồ Tỷ lệ gian lận theo giờ




```
Top 5 khung giờ có tỷ lệ gian lận cao nhất:
hour
23    0.025071
1     0.024208
2     0.023928
8     0.023670
16    0.023121
Name: is_fraud, dtype: float64
```

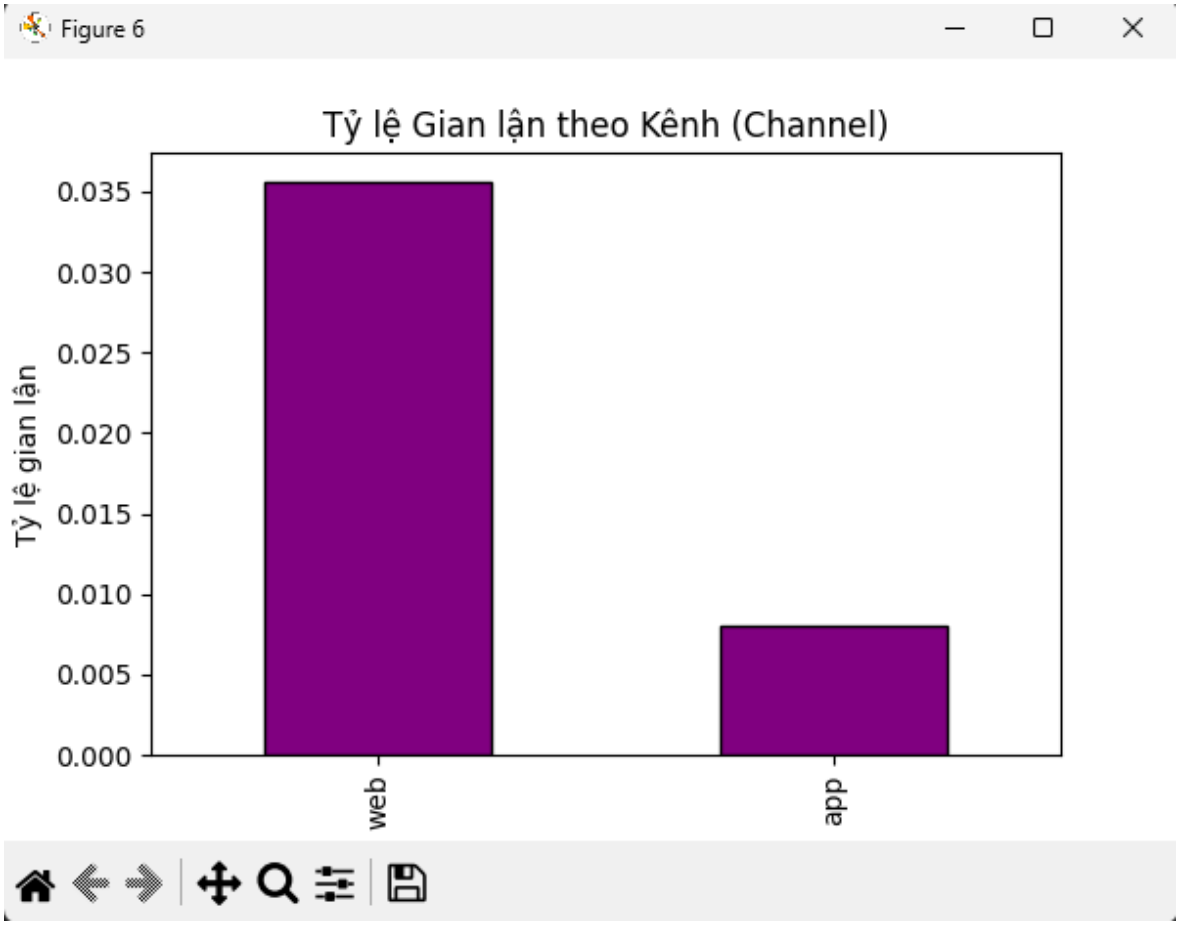
Biểu đồ tỷ lệ gian lận theo giờ cho thấy dù số lượng giao dịch giữa các giờ là khá đồng đều, rủi ro gian lận lại thay đổi rõ rệt theo thời gian trong ngày. Cụ thể, các khung giờ như 23h, 1h, 2h, 8h và 16h có tỷ lệ gian lận cao hơn mức trung bình (khoảng 2.2%), trong đó 23h là cao nhất (~2.5%). Điều này phản ánh hành vi gian lận thường có xu hướng diễn ra vào ban đêm hoặc thời điểm ít bị giám sát, khi người dùng hợp pháp ít hoạt động và hệ thống kiểm soát thủ công kém hiệu quả hơn. Ngược lại, các khung giờ giữa trưa và đầu giờ chiều có tỷ lệ gian lận thấp hơn, cho thấy đây là khoảng thời gian giao dịch "an toàn" hơn. Kết quả này khẳng định biến hour là một đặc trưng có giá trị trong phát hiện gian lận, không phải vì số lượng giao dịch, mà vì sự khác biệt về xác suất gian lận theo thời gian, và do đó rất phù hợp để đưa vào mô hình học máy nhằm nâng cao khả năng phát hiện fraud.

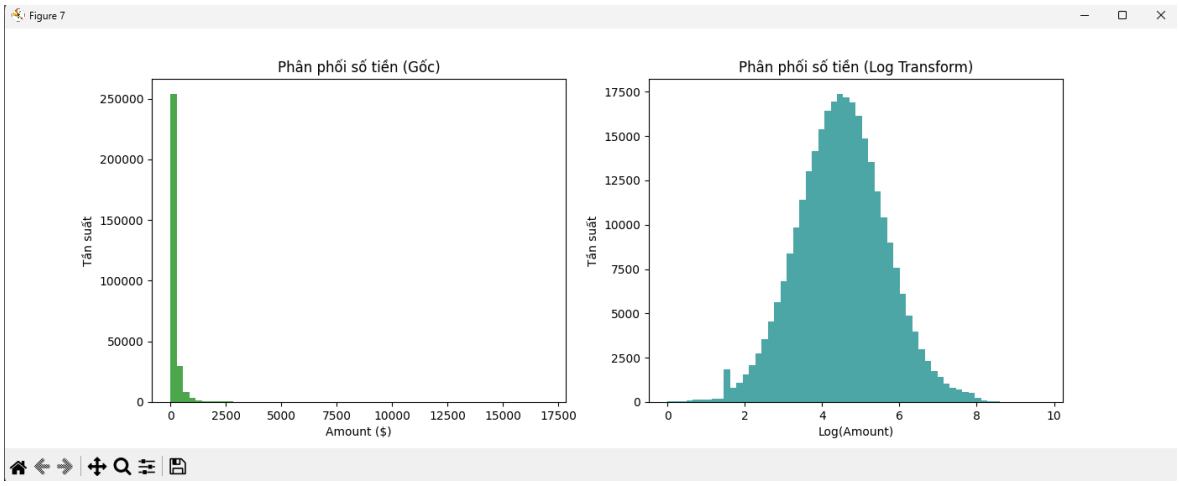
5.1.4. Biểu đồ Phân tích theo danh mục (Kênh & Danh mục người bán)



Biểu đồ cho thấy tỷ lệ gian lận có sự khác biệt nhẹ giữa các loại hình bán hàng, trong đó travel có tỷ lệ gian lận cao nhất, tiếp theo là electronics và fashion, trong khi gaming và grocery thấp hơn tương đối; điều này cho thấy các ngành liên quan đến dịch vụ, giá trị giao dịch cao hoặc hàng hóa dễ bán lại thường tiềm ẩn rủi ro gian lận lớn hơn, còn các mặt hàng thiết yếu như grocery có mức độ ổn định hơn; tuy nhiên, chênh lệch không quá lớn nên merchant category nên được xem là yếu tố bổ trợ kết hợp cùng các đặc trưng khác thay vì là yếu tố quyết định đơn lẻ trong phát hiện gian lận

5.1.5. Biểu đồ Tỷ lệ gian lận theo Kênh

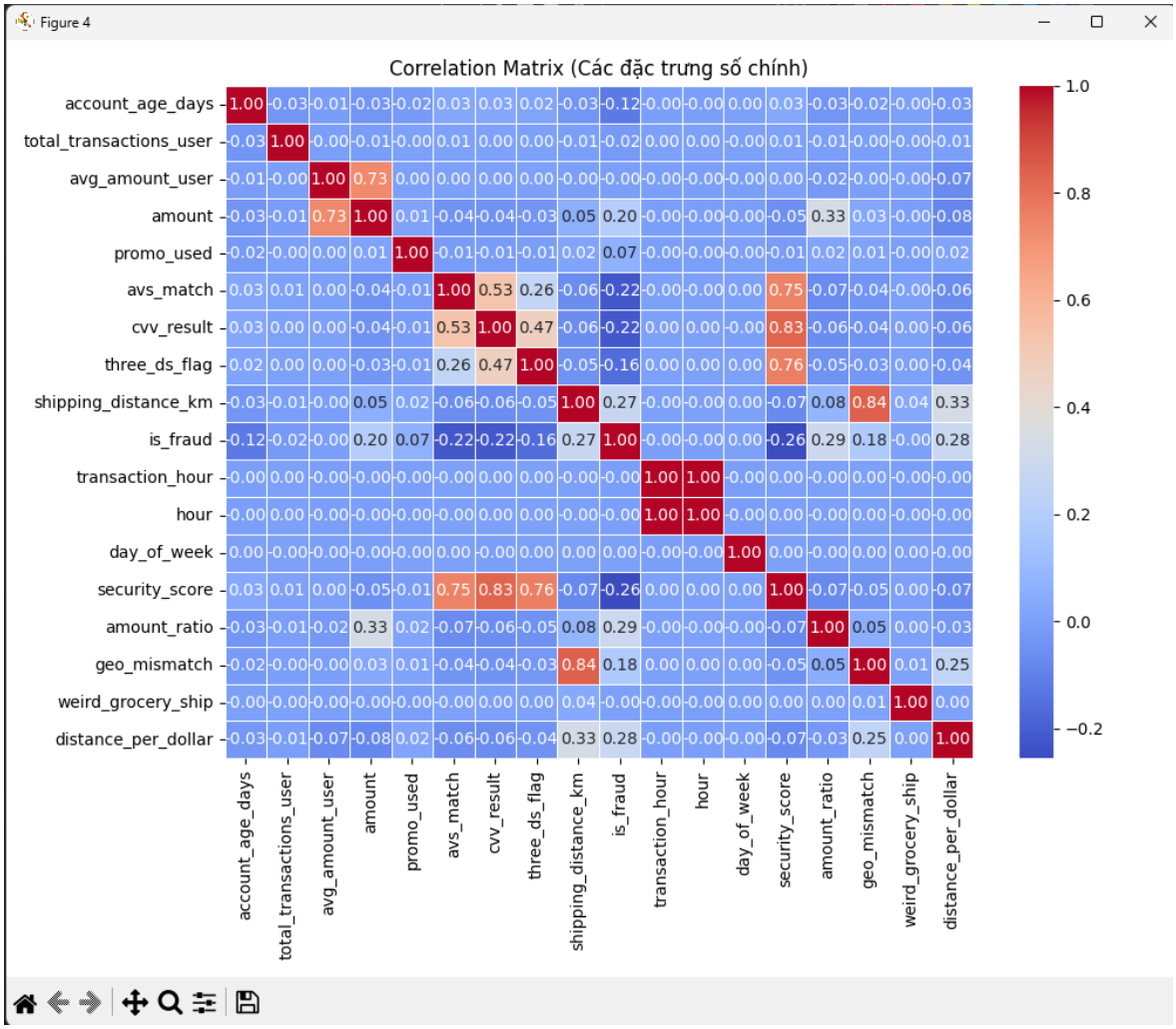




(1) Tỷ lệ gian lận theo Kênh (Channel): Biểu đồ cho thấy kênh Web có tỷ lệ gian lận cao hơn rõ rệt so với App, hàm ý các giao dịch trên Web tiềm ẩn rủi ro lớn hơn, có thể do mức độ kiểm soát thiết bị, định danh người dùng hoặc hành vi ẩn danh cao hơn; vì vậy channel là đặc trưng phân biệt tốt và nên được ưu tiên trong mô hình phát hiện gian lận cũng như trong các quy tắc giám sát rủi ro.

(2) Phân phối số tiền giao dịch (Amount): Phân phối gốc của amount bị lệch phải rất mạnh, phần lớn giao dịch có giá trị nhỏ nhưng tồn tại một số ít giao dịch giá trị rất lớn (đuôi dài), làm cho trung bình và độ lệch chuẩn cao; sau khi áp dụng log transform, phân phối trở nên gần chuẩn hơn, cho thấy việc biến đổi log là cần thiết để ổn định phương sai, giảm ảnh hưởng ngoại lệ và giúp các mô hình học máy khai thác hiệu quả hơn thông tin từ biến số tiền giao dịch.

5.1.6. Ma trận tương quan

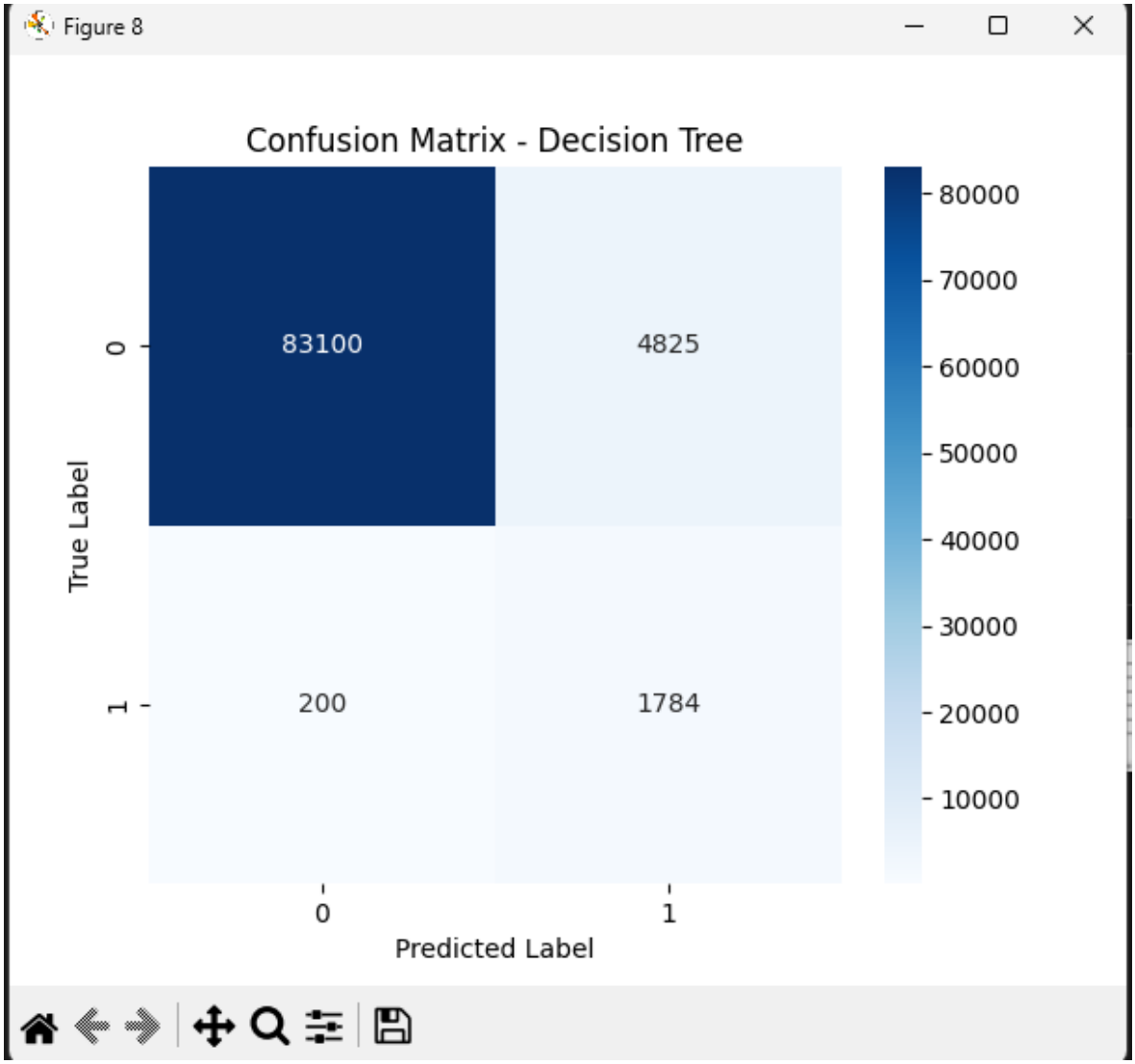


Ma trận tương quan thấy hầu hết các biến có mức tương quan thấp với nhau, cho thấy dữ liệu không bị đa cộng tuyến nghiêm trọng và phù hợp để đưa vào các mô hình học máy; một số cặp đặc trưng có tương quan cao phản ánh mối quan hệ logic trong thực tế như amount và avg_amount_user (≈0.73) hay avs_match, cvv_result, three_ds_flag với security_score (≈0.75–0.83), cho thấy các biến này cùng phản ánh mức độ xác thực và an toàn của giao dịch. Đối với biến mục tiêu is_fraud, các đặc trưng có tương quan đáng chú ý gồm shipping_distance_km, amount_ratio, distance_per_dollar và security_score, cho thấy gian lận thường gắn với khoảng cách vận chuyển bất thường, giá trị giao dịch không phù hợp với

hành vi người dùng và mức độ bảo mật thấp, trong khi account_age_days có tương quan âm cho thấy tài khoản càng mới thì rủi ro gian lận càng cao. Tổng thể, ma trận tương quan giúp xác nhận tính hợp lý của các đặc trưng được lựa chọn, hỗ trợ giải thích vì sao các mô hình như Decision Tree, Random Forest và XGBoost ưu tiên các biến hành vi và bảo mật trong việc phát hiện gian lận giao dịch điện tử.5.2. Kết quả mô hình Decision Tree

5.2. Kết quả mô hình Decision Tree

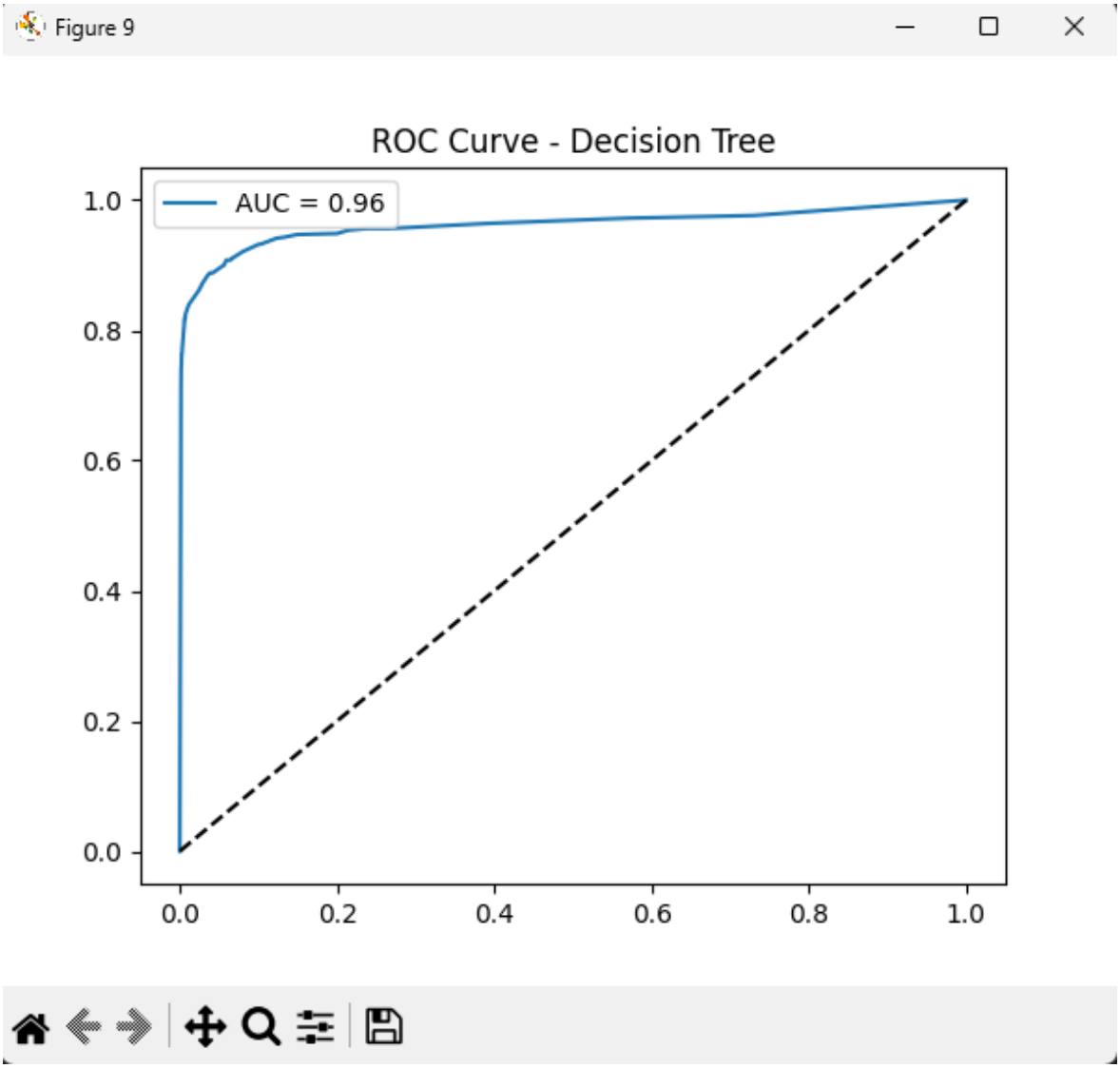
5.2.1. Confusion Matrix - Decision Tree



Dựa vào confusion matrix của mô hình Decision Tree, có thể thấy mô hình phân loại khá tốt lớp giao dịch bình thường (label 0) với 83.100 mẫu được dự đoán đúng, tuy nhiên vẫn còn 4.825 giao dịch bình thường bị dự đoán nhầm thành gian lận (false positive), điều này cho thấy mô hình có xu hướng cảnh báo nhầm một số giao dịch hợp lệ. Đối với lớp gian lận (label 1), mô hình phát hiện đúng 1.784 giao dịch gian lận, nhưng vẫn bỏ sót 200 giao dịch gian lận bị

dự đoán nhầm là bình thường (false negative), đây là điểm cần lưu ý vì trong bài toán phát hiện gian lận, false negative gây rủi ro lớn hơn. Tổng thể, mô hình có khả năng nhận diện gian lận tương đối tốt trong bối cảnh dữ liệu mất cân bằng mạnh, ưu tiên giảm bỏ sót gian lận nhưng vẫn đánh đổi bằng việc tăng số cảnh báo nhầm, phản ánh đúng đặc điểm và mục tiêu của Decision Tree khi được huấn luyện với class_weight='balanced'.

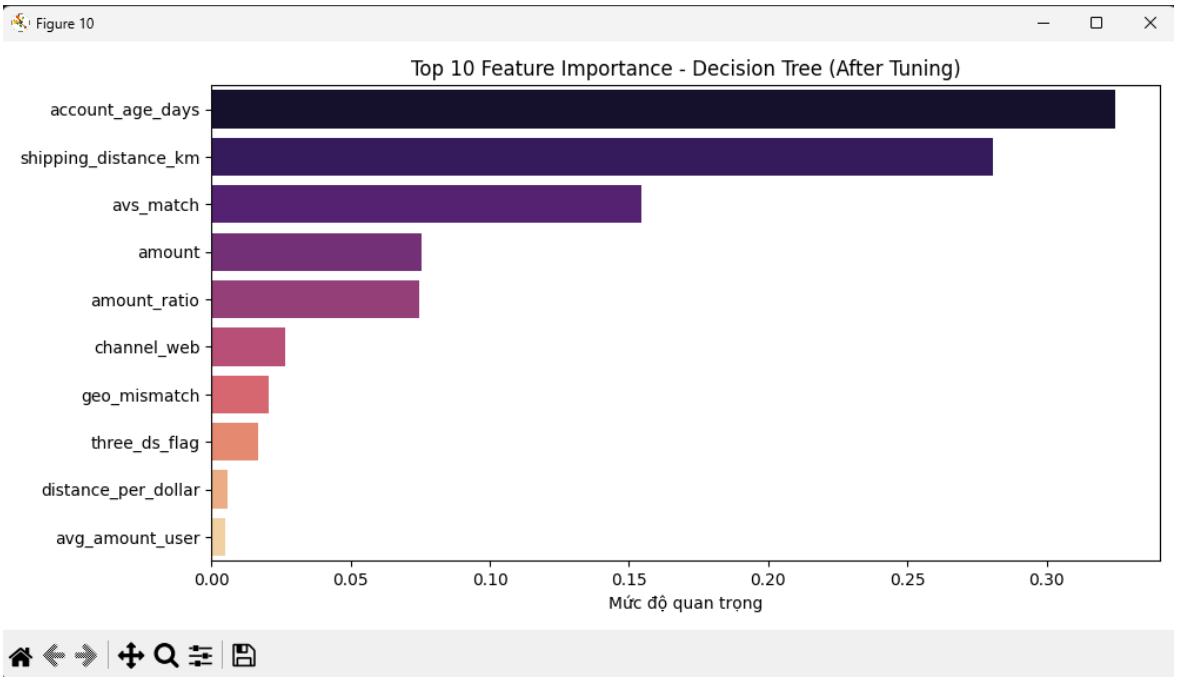
5.2.2. ROC Curve - Decision Tree



Nhìn vào ROC Curve của mô hình Decision Tree, có thể thấy đường cong nằm rất gần góc trên bên trái và cách xa đường chéo ngẫu nhiên, cho thấy mô hình có khả năng phân biệt tốt giữa giao dịch gian lận và không gian lận; giá trị AUC = 0.96 chứng tỏ với xác suất 96% mô hình xếp hạng đúng một giao dịch gian lận cao hơn một giao dịch bình thường, đồng thời tại vùng False Positive Rate thấp mô hình vẫn đạt True Positive Rate cao, điều này đặc biệt quan trọng trong bài toán phát hiện gian lận vì giúp giảm bỏ sót các giao dịch gian lận, tuy nhiên Decision Tree vẫn có nguy cơ overfitting nên kết quả này

cần được so sánh thêm với các mô hình ensemble để đảm bảo tính ổn định khi áp dụng thực tế.

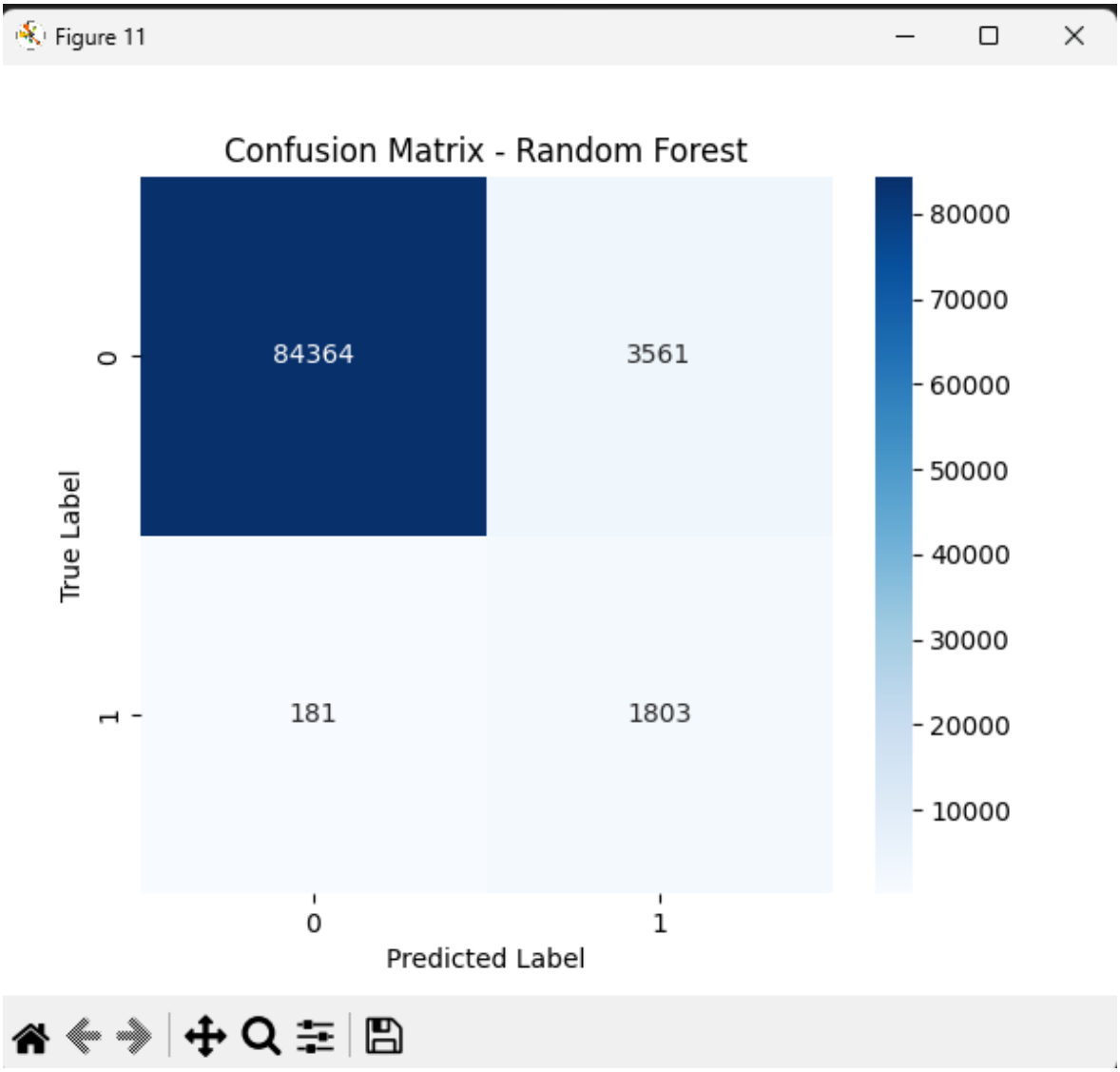
5.2.3. Top 10 Feature Importance - Decision Tree



Biểu đồ Top 10 Feature Importance của mô hình Decision Tree sau khi tuning, có thể thấy mô hình chủ yếu dựa vào các đặc trưng liên quan đến hành vi và mức độ rủi ro của người dùng để đưa ra quyết định, trong đó account_age_days là đặc trưng quan trọng nhất, cho thấy các tài khoản mới tạo có nguy cơ gian lận cao hơn rõ rệt; tiếp theo là shipping_distance_km, phản ánh việc khoảng cách vận chuyển bất thường thường gắn với các giao dịch gian lận. Đặc trưng avs_match cũng có mức ảnh hưởng lớn, cho thấy sự không khớp thông tin địa chỉ thanh toán là dấu hiệu quan trọng để nhận diện gian lận. Các yếu tố liên quan đến giá trị giao dịch như amount và amount_ratio cho thấy những giao dịch có số tiền lớn hoặc bất thường so với lịch sử người dùng dễ bị đánh giá rủi ro cao hơn. Trong khi đó, các đặc trưng như geo_mismatch, three_ds_flag hay channel_web có mức độ quan trọng thấp hơn nhưng vẫn đóng vai trò bổ trợ, giúp mô hình tinh chỉnh quyết định trong các trường hợp biên. Tổng thể, biểu đồ cho thấy Decision Tree sau tuning đã học được các quy luật hợp lý và phù hợp với logic thực tế của bài toán phát hiện gian lận giao dịch điện tử, khi ưu tiên mạnh vào các dấu hiệu hành vi bất thường và độ tin cậy của người dùng.

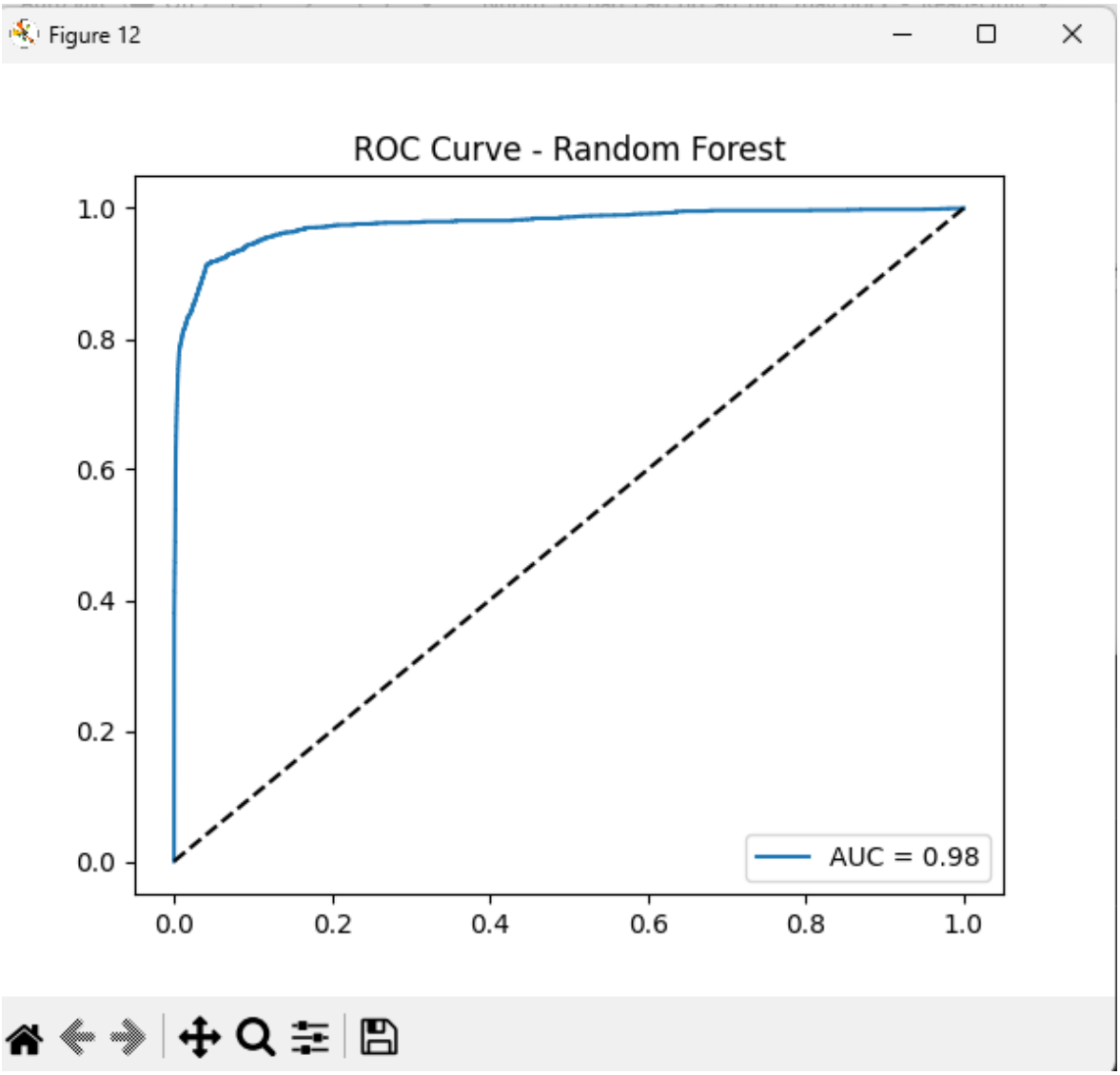
5.3. Kết quả mô hình Random Forest

5.3.1. Confusion Matrix - Random Forest



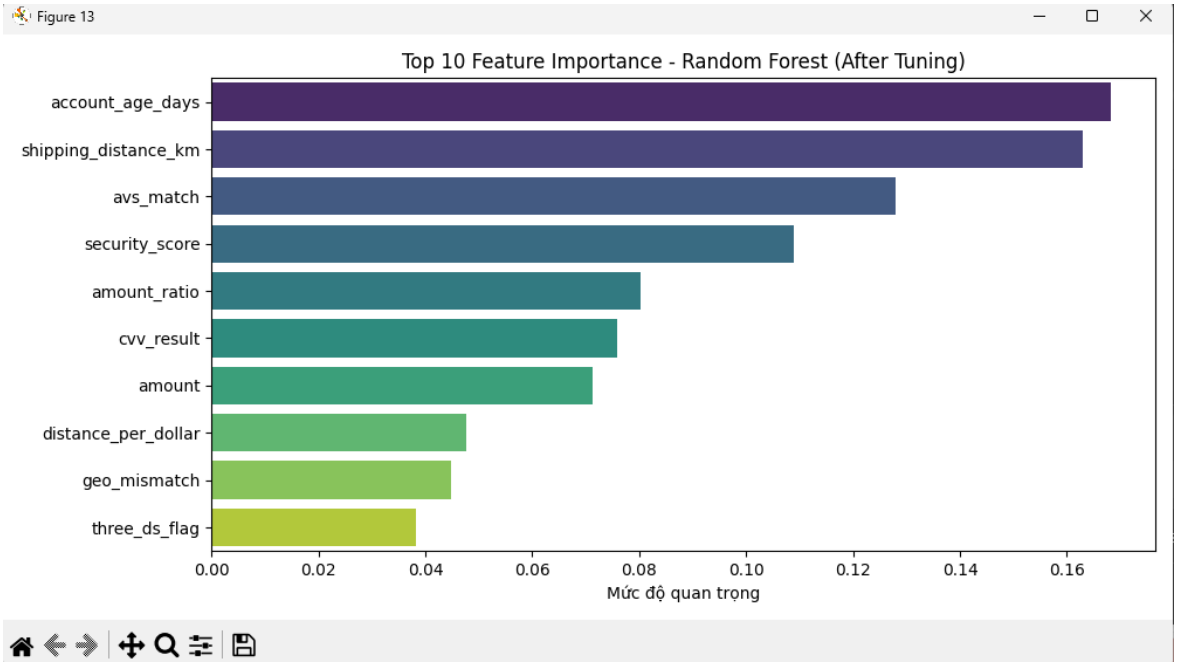
Dựa vào confusion matrix của mô hình Random Forest, có thể thấy mô hình hoạt động tốt hơn so với Decision Tree trong cả hai lớp khi dự đoán đúng 84.364 giao dịch bình thường và chỉ còn 3.561 giao dịch bình thường bị cảnh báo nhầm là gian lận, cho thấy khả năng giảm false positive rõ rệt. Đối với lớp gian lận, mô hình phát hiện đúng 1.803 giao dịch gian lận và chỉ bỏ sót 181 giao dịch (false negative), thấp hơn so với Decision Tree, điều này rất quan trọng trong bài toán phát hiện gian lận vì giúp giảm rủi ro bỏ sót các giao dịch nguy hiểm. Tổng thể, Random Forest cho kết quả cân bằng hơn giữa việc phát hiện gian lận và hạn chế cảnh báo nhầm, phản ánh ưu điểm của mô hình tổ hợp nhiều cây quyết định giúp giảm overfitting và cải thiện độ tổng quát trên dữ liệu mất cân bằng.

5.3.2. ROC Curve - Random Forest



Từ ROC Curve của mô hình Random Forest, có thể khẳng định đây là mô hình có khả năng phân biệt hai lớp giao dịch bình thường và gian lận rất mạnh, thể hiện qua đường cong ROC bám sát góc trên bên trái và cách xa rõ rệt đường chéo biểu diễn mô hình đoán ngẫu nhiên; giá trị AUC = 0.98 cho thấy xác suất mô hình xếp hạng đúng một giao dịch gian lận cao hơn một giao dịch bình thường là rất lớn, gần như tối ưu trong bối cảnh dữ liệu mất cân bằng. Đặc biệt, ở vùng False Positive Rate rất thấp, mô hình vẫn đạt True Positive Rate cao, điều này mang ý nghĩa thực tiễn quan trọng vì hệ thống có thể phát hiện phần lớn giao dịch gian lận mà không làm tăng quá nhiều cảnh báo nhầm đối với giao dịch hợp lệ. Kết quả này phản ánh rõ ưu điểm cốt lõi của Random Forest là khả năng tổng hợp quyết định từ nhiều cây khác nhau, giúp mô hình học được các mối quan hệ phi tuyến phức tạp, giảm overfitting so với Decision Tree đơn lẻ và duy trì độ ổn định khi thay đổi ngưỡng phân loại, từ đó cho thấy Random Forest là mô hình rất phù hợp cho bài toán phát hiện gian lận giao dịch điện tử cả về mặt lý thuyết lẫn hiệu quả thực nghiệm.

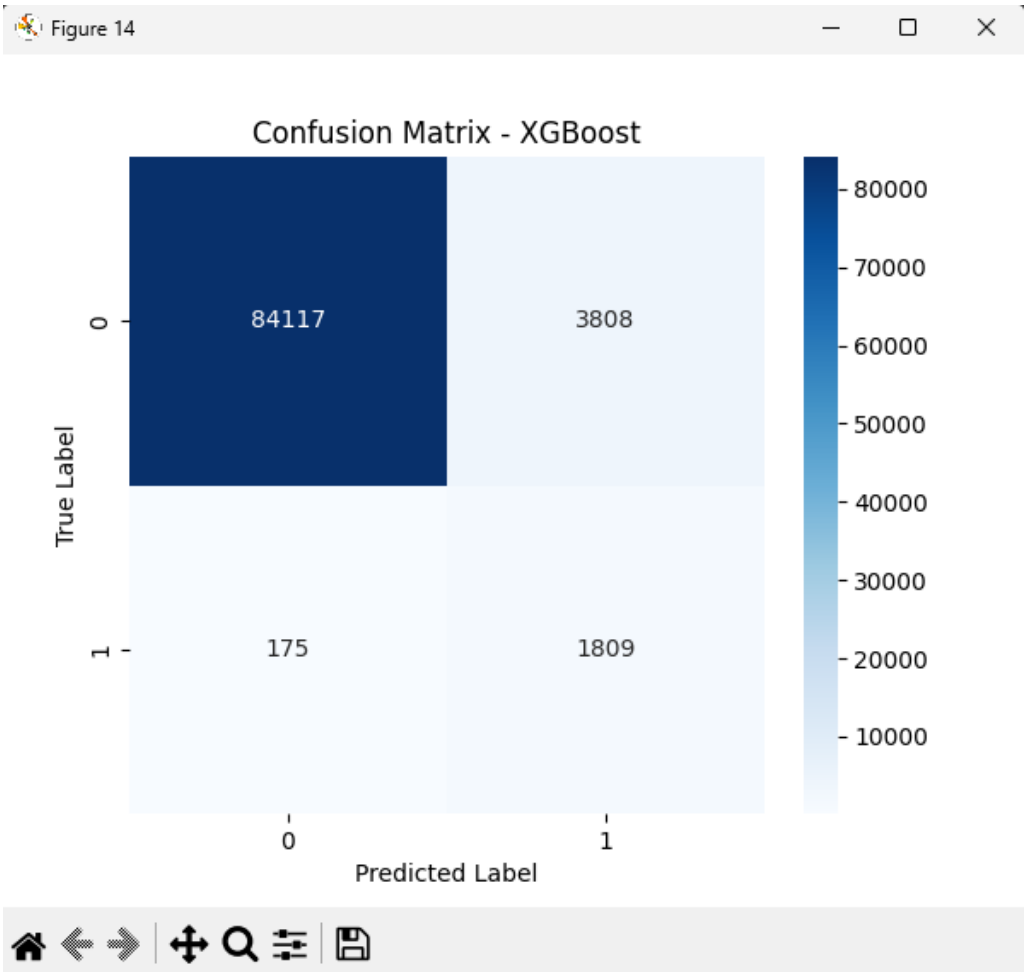
5.3.3. Top 10 Feature Importance Random Forest



Trong biểu đồ Top 10 Feature Importance của mô hình Random Forest sau khi tuning, có thể thấy mô hình vẫn ưu tiên mạnh các đặc trưng phản ánh mức độ tin cậy và hành vi bất thường của người dùng, trong đó account_age_days và shipping_distance_km là hai yếu tố quan trọng nhất, cho thấy các tài khoản mới và khoảng cách vận chuyển bất thường là dấu hiệu rủi ro cao của gian lận. So với Decision Tree, Random Forest phân bổ mức độ quan trọng đồng đều hơn cho nhiều đặc trưng như avs_match, security_score, amount_ratio, cvv_result và amount, phản ánh khả năng tổng hợp thông tin từ nhiều khía cạnh khác nhau của giao dịch thay vì phụ thuộc quá mạnh vào một vài đặc trưng đơn lẻ. Các biến bổ trợ như geo_mismatch, distance_per_dollar và three_ds_flag tuy có mức độ quan trọng thấp hơn nhưng vẫn góp phần giúp mô hình nhận diện chính xác các trường hợp gian lận phức tạp. Tổng thể, biểu đồ cho thấy Random Forest học được các quy luật hợp lý, cân bằng giữa nhiều tín hiệu rủi ro khác nhau, từ đó giải thích vì sao mô hình này cho kết quả ổn định và hiệu quả cao hơn Decision Tree trong bài toán phát hiện gian lận giao dịch điện tử.

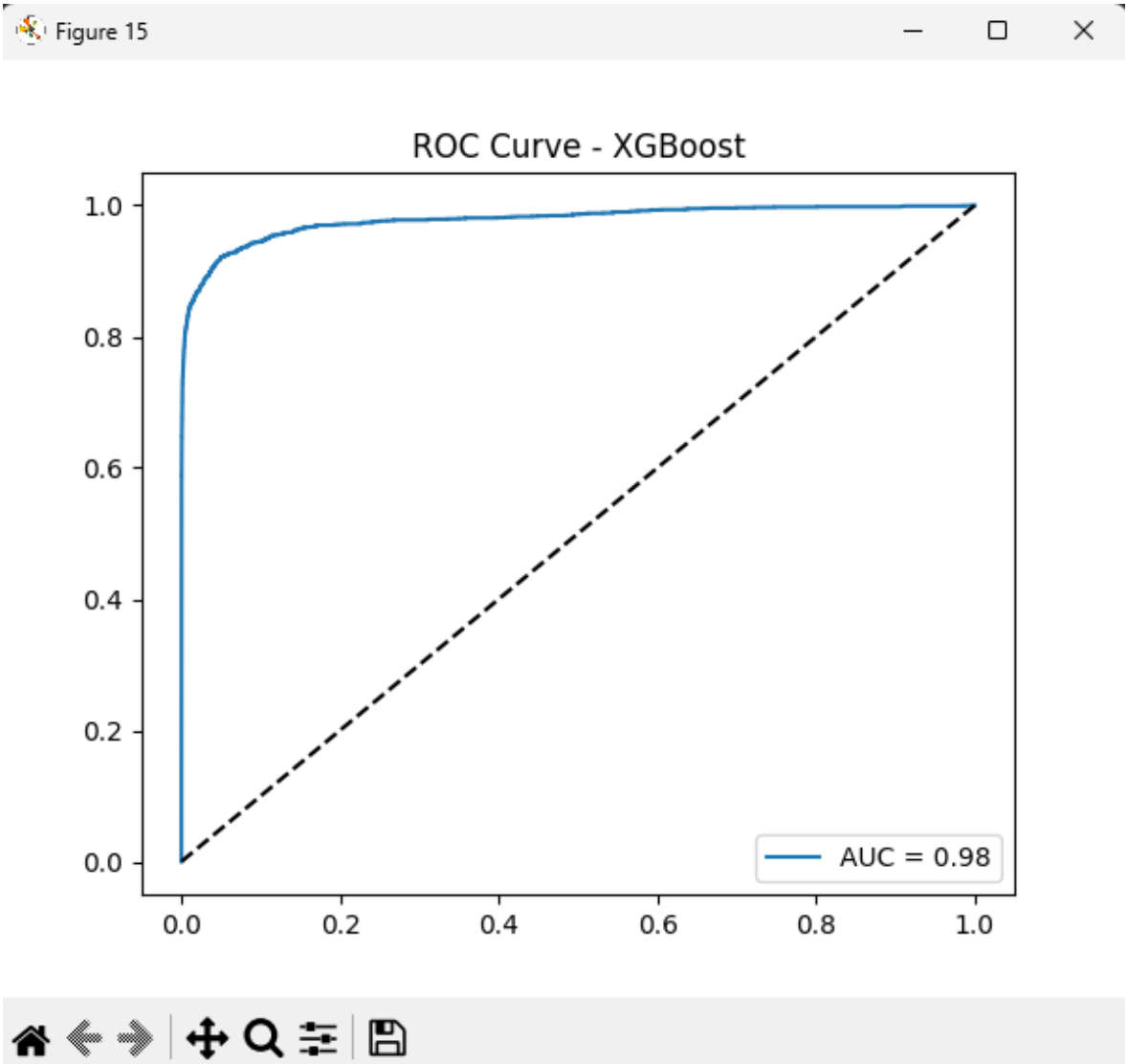
5.4. Kết quả mô hình XGBoost

5.4.1. Confusion Matrix - XGBoost



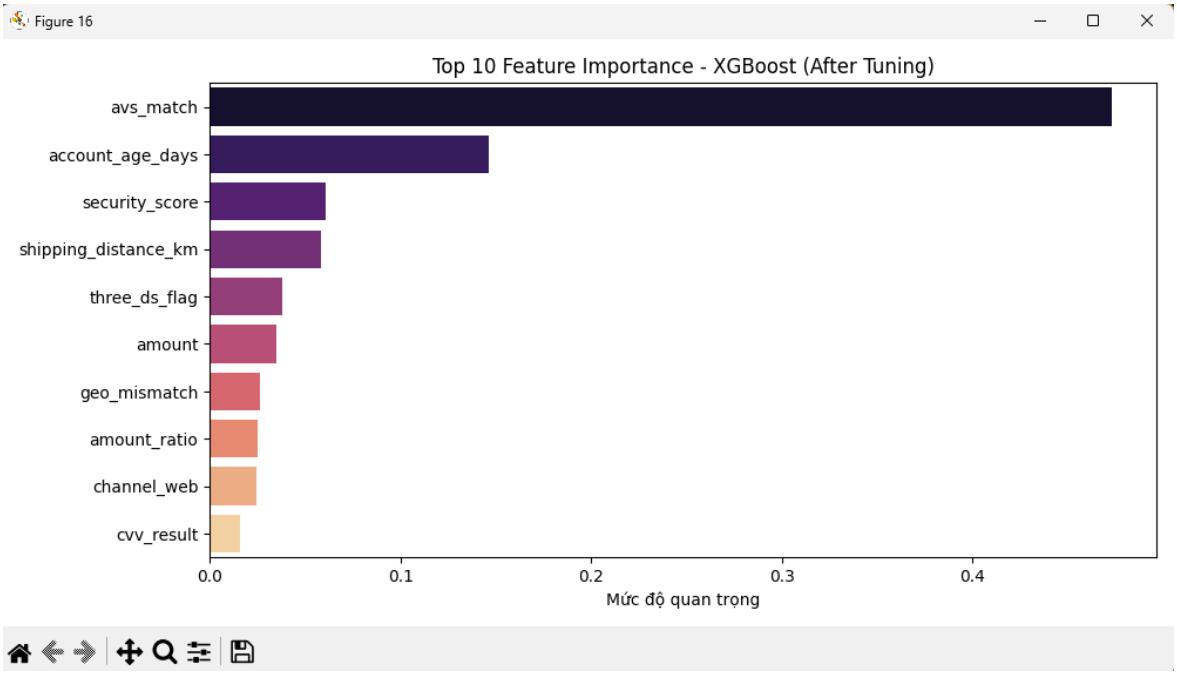
Dựa vào confusion matrix của mô hình XGBoost, có thể thấy mô hình đạt hiệu quả cao và ổn định nhất trong ba mô hình khi dự đoán đúng 84.117 giao dịch bình thường và chỉ có 3.808 giao dịch bình thường bị cảnh báo nhầm, đồng thời phát hiện đúng 1.809 giao dịch gian lận và chỉ bỏ sót 175 trường hợp gian lận, là mức thấp nhất so với Decision Tree và Random Forest. Điều này cho thấy XGBoost có khả năng học được các mẫu phức tạp trong dữ liệu mất cân bằng tốt hơn, vừa giảm false negative – yếu tố đặc biệt quan trọng trong bài toán phát hiện gian lận – vừa kiểm soát false positive ở mức hợp lý. Nhờ cơ chế boosting kết hợp nhiều cây yếu theo hướng tối ưu sai số, XGBoost cho kết quả tổng thể cân bằng và đáng tin cậy nhất, phù hợp để lựa chọn làm mô hình chính cho hệ thống phát hiện gian lận giao dịch điện tử.

5.4.2. ROC Curve - XGBoost



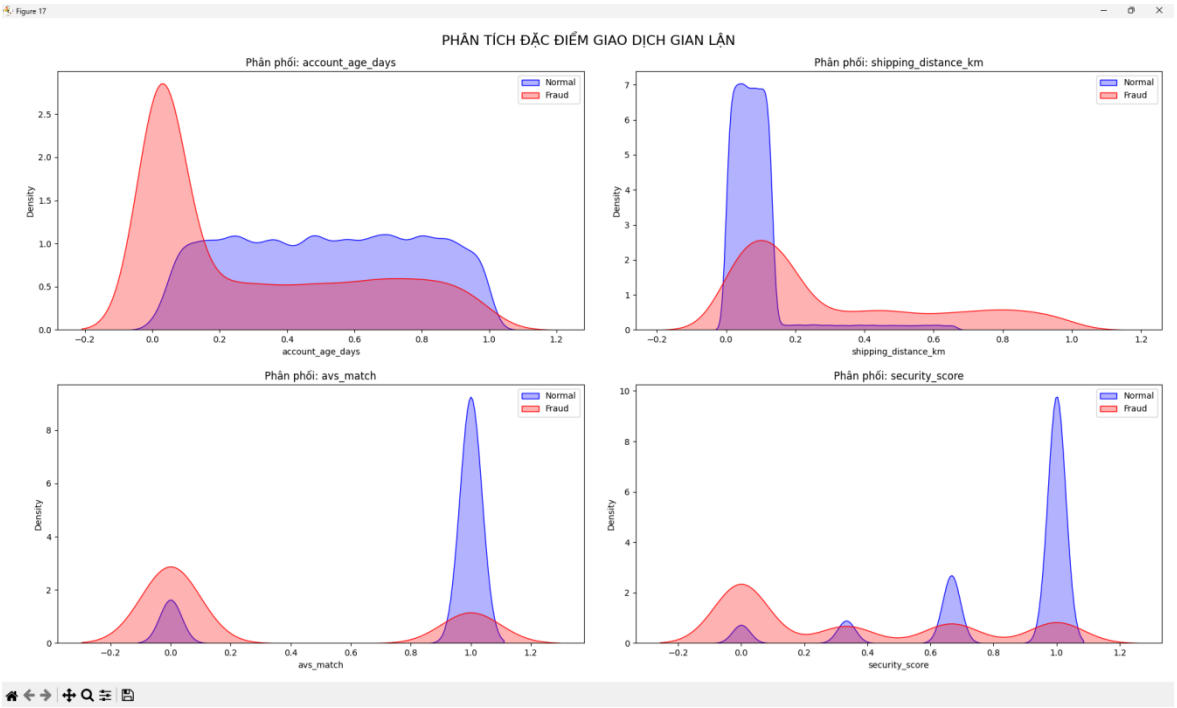
Dựa trên ROC Curve của mô hình XGBoost, có thể thấy đây là mô hình có khả năng phân biệt hai lớp giao dịch bình thường và gian lận rất mạnh, thể hiện qua đường cong ROC bám sát góc trên bên trái và nằm xa đường chéo biểu diễn dự đoán ngẫu nhiên; với giá trị AUC = 0.98, mô hình cho thấy xác suất xếp hạng đúng một giao dịch gian lận cao hơn một giao dịch bình thường là rất lớn, phản ánh hiệu quả phân loại gần mức tối ưu trong bối cảnh dữ liệu mất cân bằng nghiêm trọng. Đặc biệt, tại vùng False Positive Rate rất thấp, XGBoost vẫn duy trì True Positive Rate cao, điều này có ý nghĩa thực tiễn quan trọng vì giúp phát hiện phần lớn giao dịch gian lận mà không làm gia tăng quá nhiều cảnh báo nhầm đối với giao dịch hợp lệ. Kết quả này cho thấy XGBoost học tốt các mối quan hệ phi tuyến phức tạp giữa các đặc trưng nhờ cơ chế boosting, trong đó các cây yếu được huấn luyện tuần tự để tập trung sửa lỗi của các dự đoán trước, từ đó vừa giảm bỏ sót gian lận vừa kiểm soát tốt sai lệch dự đoán. Tổng thể, ROC Curve và AUC của XGBoost khẳng định đây là mô hình có hiệu năng cao, ổn định và phù hợp nhất để triển khai trong hệ thống phát hiện gian lận giao dịch điện tử so với Decision Tree và Random Forest.

5.4.3. Top 10 Feature Importance XGBoost



Dựa trên biểu đồ Top 10 Feature Importance của mô hình XGBoost sau khi tuning, có thể thấy mô hình tập trung rất mạnh vào các đặc trưng phản ánh trực tiếp mức độ xác thực và rủi ro của giao dịch, trong đó avs_match là đặc trưng quan trọng nhất với mức ảnh hưởng vượt trội, cho thấy sự không khớp thông tin địa chỉ thanh toán là tín hiệu then chốt để nhận diện gian lận. Tiếp theo là account_age_days, phản ánh việc các tài khoản mới tạo có xác suất gian lận cao hơn, cùng với security_score và shipping_distance_km, cho thấy XGBoost đặc biệt nhạy với các yếu tố đánh giá độ tin cậy tổng thể và hành vi bất thường trong giao nhận. Các đặc trưng như three_ds_flag, amount và geo_mismatch đóng vai trò bổ trợ, giúp mô hình tinh chỉnh quyết định trong những trường hợp ranh giới giữa gian lận và hợp lệ. So với Decision Tree và Random Forest, XGBoost phân bổ trọng số không đồng đều mà tập trung mạnh vào một số đặc trưng có tính quyết định cao, phản ánh bản chất của thuật toán boosting khi ưu tiên các biến giúp giảm lỗi nhiều nhất qua từng vòng học. Tổng thể, biểu đồ cho thấy XGBoost học được các quy luật rất rõ ràng và sát với logic nghiệp vụ thực tế, giải thích vì sao mô hình này đạt hiệu năng cao nhất và phù hợp nhất để triển khai cho bài toán phát hiện gian lận giao dịch điện tử.

5.5. Phân tích đặc điểm giao dịch gian lận



Dựa trên các biểu đồ phân phối đặc trưng giữa giao dịch bình thường và gian lận, có thể nhận thấy sự khác biệt rất rõ ràng về mặt hành vi: với account_age_days, các giao dịch gian lận chủ yếu tập trung ở nhóm tài khoản mới tạo, trong khi giao dịch bình thường phân bố rộng hơn ở các tài khoản có tuổi đời lớn, cho thấy tài khoản mới là yếu tố rủi ro cao; đối với shipping_distance_km, giao dịch gian lận có xu hướng xuất hiện nhiều ở các khoảng cách vận chuyển lớn và bất thường, trái ngược với giao dịch bình thường thường tập trung quanh khoảng cách ngắn; ở đặc trưng avs_match, giao dịch bình thường chủ yếu có giá trị khớp cao, trong khi giao dịch gian lận tập trung nhiều ở vùng không khớp hoặc khớp thấp, phản ánh việc thông tin địa chỉ thanh toán bị giả mạo; tương tự, security_score cho thấy giao dịch bình thường tập trung ở mức điểm bảo mật cao, còn giao dịch gian lận phân bố rộng ở các mức điểm thấp và trung bình. Tổng thể, các biểu đồ này chứng minh rằng dữ liệu gian lận có đặc điểm hành vi khác biệt rõ rệt so với dữ liệu bình thường, đồng thời giải thích vì sao các đặc trưng như account_age_days, shipping_distance_km, avs_match và security_score có mức độ quan trọng cao trong các mô hình học máy và đóng vai trò then chốt trong việc phát hiện gian lận giao dịch điện tử.

PHẦN 6: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Trong dự án này, nhóm đã xây dựng thành công một hệ thống phát hiện gian lận giao dịch điện tử dựa trên dữ liệu giao dịch thực tế, được triển khai theo một pipeline học máy hoàn chỉnh và sát với bài toán fraud trong môi trường doanh nghiệp. Quy trình thực hiện bao gồm làm sạch dữ liệu, loại bỏ các thuộc tính không mang giá trị dự đoán như ID và user_id, phân tích khám phá dữ liệu (EDA) để hiểu rõ đặc điểm phân bố và mức độ mất cân bằng nghiêm trọng của dữ liệu, thiết kế các đặc trưng mang ý nghĩa nghiệp vụ, xử lý mất cân bằng dữ liệu có kiểm soát, huấn luyện – tinh chỉnh – đánh giá và so sánh nhiều mô hình học máy khác nhau dựa trên các chỉ số phù hợp với bài toán gian lận.

Kết quả EDA cho thấy tỷ lệ giao dịch gian lận chiếm một phần rất nhỏ so với giao dịch bình thường, phản ánh đúng bản chất của bài toán fraud trong thực tế. Gian lận không xuất hiện ngẫu nhiên mà có xu hướng tập trung theo hành vi người dùng, mức độ bảo mật, khoảng cách vận chuyển và sự bất thường về giá trị giao dịch. Điều này cho thấy dữ liệu có cấu trúc hành vi rõ ràng và phù hợp với các mô hình dựa trên cây quyết định và boosting. Các đặc trưng được xây dựng như security_score, amount_ratio, geo_mismatch, distance_per_dollar hay weird_grocery_ship không chỉ mang ý nghĩa kỹ thuật mà còn phản ánh trực tiếp logic nghiệp vụ của gian lận giao dịch. Phân tích tương quan và feature importance từ Decision Tree, Random Forest và XGBoost đều cho thấy các đặc trưng liên quan đến bảo mật, hành vi chi tiêu và khoảng cách vận chuyển đóng vai trò then chốt trong việc phân biệt giao dịch gian lận.

Đối với vấn đề mất cân bằng dữ liệu – thách thức lớn nhất của dự án – nhóm đã áp dụng chiến lược giữ nguyên tập test để phản ánh đúng phân bố thực tế và chỉ thực hiện undersampling có kiểm soát trên tập train với tỷ lệ 1 Fraud : 10 Normal. Cách tiếp cận này giúp tránh rò rỉ dữ liệu, đảm bảo mô hình học được đặc trưng của gian lận mà không bị áp đảo bởi lớp Normal, đồng thời kết quả đánh giá phản ánh đúng khả năng ứng dụng trong thực tế.

Về đánh giá mô hình, Decision Tree có ưu điểm dễ diễn giải và trực quan, nhưng bộc lộ rõ hạn chế về overfitting và khả năng tổng quát hóa, khiến Recall chưa ổn định trên tập test, do đó chỉ phù hợp cho mục đích minh họa. Random Forest sau khi tuning cho kết quả ổn định hơn, giảm overfitting và cải thiện AUC so với Decision Tree, tuy nhiên do cơ chế trung bình các cây độc lập, mô hình vẫn chưa tận dụng tốt các mẫu gian lận khó phát hiện trong bối cảnh dữ liệu hiếm. Trong khi đó, XGBoost cho kết quả vượt trội nhất với AUC cao, đường ROC áp sát góc trên bên trái, Recall tốt hơn và khả năng phân biệt gian lận rõ ràng hơn, cho thấy ưu thế của các mô hình boosting trong việc tập trung học từ các lỗi khó và các giao dịch gian lận hiếm.

Tổng hợp kết quả cho thấy Random Forest là lựa chọn tốt trong nhóm mô hình truyền thống, nhưng chưa đạt mức tối ưu cho bài toán fraud – nơi việc

bỏ sót gian lận gây rủi ro lớn hơn nhiều so với dự đoán nhầm giao dịch bình thường. Dự án chứng minh rằng Feature Engineering và xử lý mất cân bằng dữ liệu đóng vai trò quyết định hơn việc chỉ thay đổi mô hình, đồng thời các mô hình truyền thống như Decision Tree và Random Forest có giới hạn rõ ràng khi đối mặt với gian lận phức tạp và hiếm. Do đó, các mô hình học tăng cường như XGBoost hoặc LightGBM là hướng tiếp cận phù hợp hơn trong thực tế, đặc biệt khi mục tiêu là tối đa hóa Recall và giảm thiểu rủi ro bỏ sót gian lận.

Trong tương lai, hệ thống có thể được mở rộng bằng cách áp dụng các mô hình boosting nâng cao hơn, sử dụng kỹ thuật giải thích mô hình như SHAP để tăng tính minh bạch, kết hợp dữ liệu theo chuỗi thời gian nhằm phát hiện gian lận theo hành vi liên tiếp, và thử nghiệm các phương pháp cân bằng dữ liệu khác như SMOTE kết hợp ensemble. Nhìn chung, dự án không chỉ dừng lại ở việc xây dựng mô hình học máy, mà còn thể hiện rõ tư duy phân tích dữ liệu, hiểu nghiệp vụ và thiết kế hệ thống phát hiện gian lận đúng chuẩn thực tế.

Tài liệu tham khảo

[1] Ian H. Witten, Eibe Frank, Mark A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 4th Edition, Morgan Kaufmann, 2016.

[2] Trevor Hastie, Robert Tibshirani, Jerome Friedman, *The Elements of Statistical Learning*, 2nd Edition, Springer, 2009.

[3] Christopher M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

[4] Géron, A., *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd Edition, O'Reilly Media, 2019.

[5] Breiman, L., "Random Forests", *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[6] Quinlan, J. R., *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers, 1993.

[7] Friedman, J. H., "Greedy Function Approximation: A Gradient Boosting Machine", *Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.

[8] Chen, T., and Guestrin, C., "XGBoost: A Scalable Tree Boosting System", *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, 2016.

[9] Dal Pozzolo, A., Bontempi, G., Snoeck, M., *Handling Imbalanced Data in Fraud Detection*, IEEE Intelligent Systems, 2015.

[10] He, H., and Garcia, E. A., "Learning from Imbalanced Data", *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[11] Pedregosa, F., et al., "Scikit-learn: Machine Learning in Python", *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[12] Kaggle, *Credit Card Fraud Detection Dataset – Discussion and Methodologies*, Kaggle Learn Resources.

Phân chia công việc nhóm

Tên	Làm code	Làm word
Trần Tất Phát	-Tiền xử lý dữ liệu (làm sạch file csv + xóa cột cũ + tạo đặc trưng mới) -Thiết kế web -Decision Tree	-Phân tích mô hình Decision Tree(4.2) -Phân tích biểu đồ RocCurve,ConfusionMatrix,Top 10 đặc trưng của decision tree
Nguyễn Huỳnh Anh Tuấn	-Khai phá dữ liệu EDA -Tổng hợp, gộp code vào 1 project duy nhất -XGBoost	-Phân tích mô hình XGBoost (4.4) -Phân tích biểu đồ RocCurve,ConfusionMatrix,Top 10 đặc trưng của XGBoost và các biểu đồ phần EDA
Đỗ Xuân Hương	-Chia Train-Test và umdersampling -Random forest	-Phân tích mô hình Random forest và tách mẫu (4.1 và 4.3) -Phân tích biểu đồ RocCurve,ConfusionMatrix,Top 10 đặc trưng của Random forest -Làm full báo cáo (Phần 1,2,3,6)