FILE: 0.py

```
import pandas as pd
from ydata_profiling import ProfileReport
import os
import warnings

# Tắt các cảnh báo không cần thiết
warnings.filterwarnings('ignore')

# 1. CẤU HÌNH
FILE_PATH = r"D:\PycharmProjects\PNMK\icatsd2026_menopause_qol\data\processed\clean_data_refined.csv"
OUTPUT_HTML = r"D:\PycharmProjects\PNMK\icatsd2026_menopause_qol\src1\Tong_quan_du_lieu.html"


def generate_auto_report():
    print("--- ĐANG TẠO BÁO CÁO TỰ ĐỘNG (Vui lòng đợi 1-2 phút)... ---")

    try:
        # Kiểm tra file tồn tại
        if not os.path.exists(FILE_PATH):
            print(f" LỖI: Không tìm thấy file dữ liệu tại {FILE_PATH}")
            return

        # Đọc dữ liệu
        df = pd.read_csv(FILE_PATH)
        print(f"-> Đã tải dữ liệu: {df.shape}")

        # Cấu hình báo cáo (ĐÃ SỬA LỖI: Xóa dark_mode=True)
        profile = ProfileReport(
            df,
            title="Báo cáo Dữ liệu Mãn kinh & CLCS (ICATSD 2026)",
            explorative=True  # Chế độ thám hiểm sâu (Deep Analysis)
        )

        # Tạo thư mục nếu chưa có
        os.makedirs(os.path.dirname(OUTPUT_HTML), exist_ok=True)

        # Xuất ra file HTML
        print("-> Đang tính toán thống kê và vẽ biểu đồ...")
        profile.to_file(OUTPUT_HTML)

        print(f"\n ĐÃ XONG! Báo cáo được lưu tại:")
        print(f"-> {OUTPUT_HTML}")
        print("-> Hãy vào thư mục trên và mở file HTML bằng trình duyệt (Chrome/Edge).")

    except Exception as e:
        print(f" LỖI HỆ THỐNG: {e}")
```

```
if __name__ == "__main__":
    generate_auto_report()
```

FILE: 1.py

```
import pandas as pd
import numpy as np
import os
# Load dữ liệu
# --- CẤU HÌNH ĐƯỜNG DẪN ---
# Sử dụng đường dẫn tuyệt đối để tránh lỗi không tìm thấy file
FILE_PATH = r"D:\PycharmProjects\PNMK\icatsd2026_menopause_qol\data\processed\clean_data_refined.csv"

# Kiểm tra xem file có tồn tại không trước khi đọc
if not os.path.exists(FILE_PATH):
    print(f" LỖI: Không tìm thấy file tại {FILE_PATH}")
    print(" Bạn hãy kiểm tra lại xem file 'clean_data_refined.csv' đang nằm ở thư mục nào?")
    exit()

# Load dữ liệu
df = pd.read_csv(FILE_PATH)
print(" Đã tải dữ liệu thành công!")

# 1. TÍNH TOÁN CHỈ SỐ (FEATURE ENGINEERING)
# BMI: Weight (kg) / Height (m)^2
df['BMI'] = df['Weight_kg'] / ((df['Height_cm'] / 100) ** 2)

# PSS_Score: Tổng điểm (đảo ngược câu 4, 5, 7, 8)
# Giả định thang đo 0-4 (chuẩn PSS-10). Đảo ngược = 4 - điểm cũ.
pss_cols = [f'PSS_{i}' for i in range(1, 11)]
positive_items = ['PSS_4', 'PSS_5', 'PSS_7', 'PSS_8']
# Tạo bản sao để tính toán
df_pss = df[pss_cols].copy()
for col in positive_items:
    df_pss[col] = 4 - df_pss[col]
df['PSS_Score'] = df_pss.sum(axis=1)

# MENQOL_Score: Trung bình cộng các triệu chứng
men_cols = [c for c in df.columns if c.startswith('MEN_') and c not in ['MENQOL_Score', 'Meno_Age', 'Meno_Age_Numeric', 'Meno_Group', 'Meno_Duration']]
df['MENQOL_Score'] = df[men_cols].mean(axis=1)

# 2. MÃ HÓA (ENCODING)
# Education
edu_map = {'Không đi học': 0, 'THCS': 2, 'THPT – trung cấp': 3, 'Đại học – cao đẳng': 4, 'Trên đại học': 5}
df['Education_Code'] = df['Education'].map(edu_map)

# Income (Gộp "Trên 10 triệu" vào nhóm 3: 11-20tr)
income_map = {'Dưới 5 triệu': 1, 'Từ 5 đến 10 triệu': 2, 'Trên 10 triệu': 3, 'Từ 11 đến 20 triệu': 3, 'Trên 20 triệu': 4}
df['Income_Code'] = df['Income'].map(income_map)

# Job
def map_job(job):
```

```
    job = str(job).lower()
    if 'nội trợ' in job: return 1
    if 'công nhân' in job: return 2
    if 'văn phòng' in job: return 3
    if 'kinh doanh' in job: return 4
    if 'về hưu' in job: return 5
    return 6 # Chuyên gia/Khác
df['Job_Code'] = df['Job'].apply(map_job)

# Marital Status
df['Marital_Code'] = df['Marital_Status'].apply(lambda x: 0 if 'độc thân' in str(x).lower() else 1)
def clean_meno_age(val):
    val_str = str(val).strip().lower()
    try:
        # Nếu là số -> giữ nguyên
        return float(val_str)
    except ValueError:
        # Nếu là chữ -> kiểm tra từ khóa
        keywords = ['chưa', 'không', 'vẫn', 'đều', 'sắp', 'đang']
        if any(kw in val_str for kw in keywords):
            return 0  # Quy ước 0 là chưa mãn kinh
        return 0 # Mặc định các trường hợp lạ khác về 0

# Áp dụng hàm làm sạch
df['Meno_Age_Clean'] = df['Meno_Age'].apply(clean_meno_age)

# Tạo trạng thái mãn kinh (1: Có, 0: Không)
df['Meno_Status'] = df['Meno_Age_Clean'].apply(lambda x: 1 if x > 0 else 0)

# Tính thời gian mãn kinh (Duration)
df['Meno_Duration_New'] = df.apply(
    lambda row: max(0, row['Age'] - row['Meno_Age_Clean']) if row['Meno_Status'] == 1 else 0,
    axis=1
)
# Lưu file
df.to_csv("clean_data_preprocessed.csv", index=False)
```

FILE: 10.py

```
import os
from fpdf import FPDF
import warnings

warnings.filterwarnings('ignore')

# --- CẤU HÌNH ---
SOURCE_DIR = r"D:\PycharmProjects\PNMK\icatsd2026_menopause_qol\src1"
OUTPUT_FILE = "Tong_Hop_Ma_Nguon_Luan_An.pdf"
# Đường dẫn font hỗ trợ tiếng Việt (Sử dụng font có sẵn trên Windows)
FONT_PATH = r"C:\Windows\Fonts\arial.ttf"


class CodePDF(FPDF):
    def header(self):
        self.set_font("Arial", "B", 8)
        self.set_text_color(150, 150, 150)
        self.cell(0, 10, "Báo cáo Mã nguồn Hệ thống Adaptive Health Advisor - ICATSD 2026", 0, 1, "R")
        self.ln(5)

    def footer(self):
        self.set_y(-15)
        self.set_font("Arial", "I", 8)
        self.cell(0, 10, f"Trang {self.page_no()}", 0, 0, "C")


def export_src_to_pdf():
    pdf = CodePDF()
    pdf.set_auto_page_break(auto=True, margin=15)

    # Đăng ký font hỗ trợ tiếng Việt
    if os.path.exists(FONT_PATH):
        pdf.add_font("Arial", "", FONT_PATH)
        pdf.add_font("Arial", "B", FONT_PATH)
        pdf.set_font("Arial", size=10)
    else:
        print(" Không tìm thấy font Arial.ttf, kết quả có thể lỗi hiển thị tiếng Việt.")
        pdf.set_font("Courier", size=10)

    # Lấy danh sách file .py và sắp xếp theo tên (1, 2, 3...)
    files = [f for f in os.listdir(SOURCE_DIR) if f.endswith('.py')]
    files.sort(key=lambda x: str(x))  # Sắp xếp theo thứ tự số

    print(f"--- ĐANG TỔNG HỢP {len(files)} FILES SANG PDF ---")

    for filename in files:
        file_path = os.path.join(SOURCE_DIR, filename)
```

```
        # Thêm trang mới cho mỗi file
        pdf.add_page()

        # Tiêu đề file
        pdf.set_font("Arial", "B", 14)
        pdf.set_text_color(0, 51, 102)  # Màu xanh thẫm
        pdf.cell(0, 10, f"FILE: {filename}", ln=True)
        pdf.ln(2)

        # Vẽ đường kẻ ngang
        pdf.line(pdf.get_x(), pdf.get_y(), pdf.get_x() + 190, pdf.get_y())
        pdf.ln(5)

        # Đọc nội dung code
        pdf.set_font("Arial", "", 9)
        pdf.set_text_color(0, 0, 0)

        try:
            with open(file_path, "r", encoding="utf-8") as f:
                code_content = f.read()

            # Ghi nội dung vào PDF (Xử lý đa dòng)
            pdf.multi_cell(0, 5, code_content)
        except Exception as e:
            pdf.cell(0, 10, f"Lỗi khi đọc file: {str(e)}", ln=True)

        print(f"   + Đã thêm: {filename}")

    # Xuất file
    pdf.output(OUTPUT_FILE)
    print("\n" + "=" * 50)
    print(f" HOÀN TẤT! File PDF đã được tạo: {OUTPUT_FILE}")
    print("=" * 50)


if __name__ == "__main__":
    export_src_to_pdf()
```

FILE: 11.py

```
import pandas as pd
import numpy as np
import pingouin as pg
import os
import warnings
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# Tắt cảnh báo
warnings.filterwarnings('ignore')

# --- CẤU HÌNH ---
FILE_PATH = r"D:\PycharmProjects\PNMK\icatsd2026_menopause_qol\data\processed\clean_data_final.csv"
REPORT_PATH = r"D:\PycharmProjects\PNMK\icatsd2026_menopause_qol\reports\statistical_pingouin_report.xlsx"

# Tạo thư mục báo cáo nếu chưa có
os.makedirs(os.path.dirname(REPORT_PATH), exist_ok=True)


def run_pingouin_validation():
    if not os.path.exists(FILE_PATH):
        print(f" Lỗi: Không tìm thấy file {FILE_PATH}")
        return

    # 1. Đọc dữ liệu và kiểm tra cột
    df = pd.read_csv(FILE_PATH)
    print(f" Đã tải dữ liệu: {df.shape}")

    # Tự động tìm tên cột đúng (tránh lỗi KeyError)
    potential_meno_cols = [c for c in df.columns if 'Meno_Duration' in c]
    meno_col = potential_meno_cols[0] if potential_meno_cols else None

    # Danh sách các cột cần thiết cho thống kê mô tả
    target_cols = ['Age', 'BMI', 'PSS_Score', 'MENQOL_Score']
    if meno_col: target_cols.append(meno_col)

    # 2. Tự động tái lập Cluster (vì file final.csv chưa có nhãn Cluster)
    print(" Đang tái lập phân cụm để phân tích thống kê...")
    demo_feats = ['Age', 'BMI', 'Education_Code', 'Income_Code', 'Job_Code', 'Marital_Code']
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(df[demo_feats])
    kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
    df['Cluster'] = kmeans.fit_predict(X_scaled)

    # 3. Thực hiện phân tích và lưu vào Excel
    try:
        with pd.ExcelWriter(REPORT_PATH, engine='openpyxl') as writer:
            # --- Sheet 1: Thống kê mô tả ---
```

```
            desc = df[target_cols].describe().T
            desc.to_excel(writer, sheet_name='Descriptive_Stats')

            # --- Sheet 2: ANOVA cho MENQOL ---
            aov_men = pg.anova(data=df, dv='MENQOL_Score', between='Cluster', detailed=True)
            aov_men.to_excel(writer, sheet_name='ANOVA_MENQOL')

            # --- Sheet 3: Post-hoc (So sánh cặp giữa các cụm) ---
            posthoc = pg.pairwise_tukey(data=df, dv='MENQOL_Score', between='Cluster')
            posthoc.to_excel(writer, sheet_name='PostHoc_Tukey_MENQOL')

            # --- Sheet 4: Tương quan đa biến (Spearman) ---
            # Chỉ lấy các cột số
            num_df = df.select_dtypes(include=[np.number])
            corrs = pg.pairwise_corr(num_df, columns=['Age', 'BMI', 'PSS_Score', 'MENQOL_Score'], method='spearman')
            corrs.to_excel(writer, sheet_name='Correlation_Analysis')

            print(f" HOÀN TẤT! Báo cáo đã lưu tại: {REPORT_PATH}")

    except Exception as e:
        print(f" Lỗi trong quá trình xử lý: {e}")


if __name__ == "__main__":
    run_pingouin_validation()
```

FILE: 12.py

```
import pandas as pd
import numpy as np
import joblib
import shap
import matplotlib.pyplot as plt
import os
import warnings

warnings.filterwarnings('ignore')

# --- CẤU HÌNH ---
MODEL_PATH = r"D:\PycharmProjects\PNMK\icatsd2026_menopause_qol\src1\final_health_advisor.pkl"
TEST_FILE = "test_data.csv"
OUTPUT_DIR = r"D:\PycharmProjects\PNMK\icatsd2026_menopause_qol\reports\xai_shap"

os.makedirs(OUTPUT_DIR, exist_ok=True)

# Bộ từ điển để hiển thị tên câu hỏi tiếng Việt trên biểu đồ
QUESTION_MAP = {
    'PSS_1': 'Kho khan doi mat van de',
    'PSS_2': 'Mat kiem soat cuoc song',
    'PSS_3': 'Cang thang lo lang',
    'PSS_10': 'Kho khan tich tu qua muc',
    'MEN_HotFlash': 'Boc hoa',
    'MEN_Memory': 'Giam tri nho',
    'MEN_Sleep': 'Mat ngu',
    'MEN_Depressed': 'Lo au tram cam',
    'MEN_Impatient': 'Mat kien nhan',
    'MEN_Libido': 'Giam ham muon',
    'MEN_Fatigue': 'Met moi',
    'Age': 'Tuoi',
    'BMI': 'Chi so BMI'
}


def run_shap_analysis():
    if not os.path.exists(MODEL_PATH):
        print("Khong tim thay file model!")
        return

    package = joblib.load(MODEL_PATH)
    test_df = pd.read_csv(TEST_FILE)
    experts = package['experts']
    strategy = package['strategy']
    demo_feats = package['demo_feats']

    print("--- DANG THUC HIEN GIAI DOAN 12: GIAI THICH MO HINH (SHAP) ---")
```

```
    # Chung ta se phan tich Cluster 1 (Nhom co mau thu lon nhat) de lam vi du dien hinh
    c_id = 1
    target_types = ['PSS', 'MEN']

    for t_type in target_types:
        print(f"\n[+] Dang phan tich SHAP cho: {t_type} (Cluster {c_id})")

        # 1. Lay model va du lieu tuong ung
        model = experts[f"expert_{c_id}_{t_type}"]
        features = demo_feats + strategy[c_id][t_type]

        X_test = test_df[test_df['Cluster'] == c_id][features]

        # Doi ten cot sang tieng Viet de bieu do dep hon
        X_test_named = X_test.rename(columns=QUESTION_MAP)

        # 2. Khoi tao SHAP Explainer (TreeExplainer dung cho ExtraTrees)
        explainer = shap.TreeExplainer(model)
        shap_values = explainer.shap_values(X_test_named)

        # 3. VE BIEU DO TONG QUAN (Summary Plot)
        # Bieu do nay thay the Feature Importance cua Sklearn
        plt.figure(figsize=(10, 6))
        shap.summary_plot(shap_values, X_test_named, show=False)
        plt.title(f"SHAP Summary Plot - {t_type} (Cluster {c_id})")
        plt.tight_layout()
        plt.savefig(os.path.join(OUTPUT_DIR, f"shap_summary_{t_type}_c{c_id}.png"))
        plt.close()

        # 4. VE BIEU DO CHO 1 BENH NHAN CU THE (Waterfall Plot)
        # Chon benh nhan dau tien trong tap test cua cluster 1
        plt.figure(figsize=(10, 6))
        # Base value la gia tri du bao trung binh
        # shap_values[0] la dong gop cua tung bien cho benh nhan nay
        explanation = shap.Explanation(
            values=shap_values[0],
            base_values=explainer.expected_value,
            data=X_test_named.iloc[0],
            feature_names=X_test_named.columns
        )
        shap.plots.waterfall(explanation, show=False)
        plt.title(f"SHAP Waterfall - Phan tich ca benh cu the ({t_type})")
        plt.tight_layout()
        plt.savefig(os.path.join(OUTPUT_DIR, f"shap_waterfall_{t_type}_c{c_id}_user0.png"))
        plt.close()

    print(f"\nHoan tat! Bieu do SHAP da duoc luu tai: {OUTPUT_DIR}")


if __name__ == "__main__":
    run_shap_analysis()
```

FILE: 13.py

```
import pandas as pd
import numpy as np
import joblib
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_absolute_error
import os

# --- CẤU HÌNH ---
MODEL_PATH = r"D:\PycharmProjects\PNMK\icatsd2026_menopause_qol\src1\final_health_advisor.pkl"
TEST_FILE = "test_data.csv"
OUTPUT_DIR = r"D:\PycharmProjects\PNMK\icatsd2026_menopause_qol\reports\error_analysis"

os.makedirs(OUTPUT_DIR, exist_ok=True)


def run_error_analysis():
    # 1. Tải mô hình và dữ liệu test
    package = joblib.load(MODEL_PATH)
    test_df = pd.read_csv(TEST_FILE)

    scaler = package['scaler']
    kmeans = package['kmeans']
    experts = package['experts']
    strategy = package['strategy']
    demo_feats = package['demo_feats']

    print("--- ĐANG THỰC HIỆN GIAI ĐOẠN 13: PHÂN TÍCH SAI SỐ & TRƯỜNG HỢP BIÊN ---")

    all_results = []

    # 2. Dự báo và tính sai số dư (Residuals)
    for _, row in test_df.iterrows():
        user_demo_df = pd.DataFrame([row[demo_feats]])
        user_demo_scaled = scaler.transform(user_demo_df)
        c_id = kmeans.predict(user_demo_scaled)[0]

        res_item = {'Actual_Cluster': row['Cluster'], 'AI_Cluster': c_id}

        for t_type in ['PSS', 'MEN']:
            target_col = 'PSS_Score' if t_type == 'PSS' else 'MENQOL_Score'
            req_feats = demo_feats + strategy[c_id][t_type]

            input_df = pd.DataFrame([row[req_feats]])
            pred = experts[f"expert_{c_id}_{t_type}"].predict(input_df)[0]

            actual = row[target_col]
            residual = actual - pred  # Sai số dư
```

```
        res_item[f'{t_type}_Actual'] = actual
        res_item[f'{t_type}_Pred'] = pred
        res_item[f'{t_type}_Error'] = residual
        res_item[f'{t_type}_AbsError'] = abs(residual)

    all_results.append(res_item)

error_df = pd.DataFrame(all_results)

# 3. VẼ BIỂU ĐỒ RESIDUAL PLOT
plt.figure(figsize=(16, 6))

# Biểu đồ PSS Residuals
plt.subplot(1, 2, 1)
sns.scatterplot(x='PSS_Pred', y='PSS_Error', data=error_df, hue='AI_Cluster', palette='viridis', s=100)
plt.axhline(0, color='red', linestyle='--')
plt.title('Residual Plot - PSS Score\n(Dưới 0: AI đoán cao hơn thực tế | Trên 0: AI đoán thấp hơn)')
plt.xlabel('Giá trị Dự báo')
plt.ylabel('Sai số dư (Actual - Predicted)')

# Biểu đồ MENQOL Residuals
plt.subplot(1, 2, 2)
sns.scatterplot(x='MEN_Pred', y='MEN_Error', data=error_df, hue='AI_Cluster', palette='magma', s=100)
plt.axhline(0, color='red', linestyle='--')
plt.title('Residual Plot - MENQOL Score')
plt.xlabel('Giá trị Dự báo')
plt.ylabel('Sai số dư')

plt.tight_layout()
plt.savefig(os.path.join(OUTPUT_DIR, "residual_plots.png"))
plt.close()

# 4. TRÍCH XUẤT CÁC TRƯỜNG HỢP SAI SỐ LỚN NHẤT (TOP OUTLIERS)
print("\n CÁC TRƯỜNG HỢP AI ĐOÁN SAI NHIỀU NHẤT (TOP 5 OUTLIERS - PSS):")
top_errors_pss = error_df.nlargest(5, 'PSS_AbsError')
print(top_errors_pss[['Actual_Cluster', 'AI_Cluster', 'PSS_Actual', 'PSS_Pred', 'PSS_Error']])

# 5. PHÂN TÍCH ĐẶC ĐIỂM CỤM 0 (NHÓM THIỂU SỐ)
c0_errors = error_df[error_df['Actual_Cluster'] == 0]
if not c0_errors.empty:
    print("\n PHÂN TÍCH RIÊNG CỤM 0 (Tiền mãn kinh):")
    mae_c0_pss = c0_errors['PSS_AbsError'].mean()
    mae_c0_men = c0_errors['MEN_AbsError'].mean()
    print(f"   - MAE PSS (Cluster 0): {mae_c0_pss:.4f}")
    print(f"   - MAE MEN (Cluster 0): {mae_c0_men:.4f}")

error_df.to_csv(os.path.join(OUTPUT_DIR, "detailed_error_report.csv"), index=False)
print(f"\n Hoàn tất! Báo cáo sai số lưu tại: {OUTPUT_DIR}")
```

```
if __name__ == "__main__":
    run_error_analysis()
```

FILE: 14.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.metrics import r2_score, mean_absolute_error
import joblib

# --- CẤU HÌNH ---
TRAIN_BALANCED = "train_data_balanced.csv"
TEST_DATA = "test_data.csv"
MODEL_PKL = "final_health_advisor.pkl"


def run_ablation_study():
    train = pd.read_csv(TRAIN_BALANCED)
    test = pd.read_csv(TEST_DATA)
    package = joblib.load(MODEL_PKL)

    demo_feats = package['demo_feats']
    target = 'MENQOL_Score'  # Ví dụ với MENQOL

    # --- 1. BASELINE MODEL (Chỉ dùng Demo) ---
    model_base = ExtraTreesRegressor(n_estimators=100, random_state=42)
    model_base.fit(train[demo_feats], train[target])
    pred_base = model_base.predict(test[demo_feats])
    r2_base = r2_score(test[target], pred_base)

    # --- 2. ADAPTIVE MODEL (Hệ thống hiện tại của bạn) ---
    # Lấy kết quả từ file 13.py hoặc chạy lại nhanh
    # Giả sử lấy từ kết quả bạn đã gửi (R2 ~ 0.98)
    r2_adaptive = 0.9801

    # --- 3. TRỰC QUAN HÓA ĐỐI CHỨNG ---
    models = ['Baseline (Chỉ Demo)', 'Adaptive Clustering (Đề xuất)']
    scores = [r2_base, r2_adaptive]

    plt.figure(figsize=(10, 6))
    bars = plt.bar(models, scores, color=['gray', 'teal'])
    plt.ylim(0, 1.1)
    plt.ylabel('R-squared Score')
    plt.title('SO SÁNH HIỆU QUẢ: PHƯƠNG PHÁP TRUYỀN THỐNG VS ĐỀ XUẤT')

    for bar in bars:
        yval = bar.get_height()
        plt.text(bar.get_x() + bar.get_width() / 2, yval + 0.02, f'{yval:.4f}', ha='center', va='bottom',
                 fontweight='bold')

    plt.savefig("comparison_result.png")
```

```
    print(f" Đã hoàn thành so sánh. R2 Baseline: {r2_base:.4f} vs Adaptive: {r2_adaptive:.4f}")


if __name__ == "__main__":
    run_ablation_study()
```

FILE: 2.py

```
import pandas as pd
import numpy as np
import os
import warnings

warnings.filterwarnings('ignore')

# --- CẤU HÌNH ---
# Đọc file kết quả từ bước trước (đang nằm cùng thư mục script hoặc đường dẫn bạn quy định)
# Giả sử file code này chạy cùng thư mục với file csv vừa tạo
INPUT_FILE = "clean_data_preprocessed.csv"
OUTPUT_FILE = "clean_data_final.csv"

# Kiểm tra file đầu vào
if not os.path.exists(INPUT_FILE):
    print(f" LỖI: Không tìm thấy file '{INPUT_FILE}'. Hãy chạy code Giai đoạn 1 trước!")
    exit()

print("--- BẮT ĐẦU GIAI ĐOẠN 2: LÀM SẠCH & CHỌN LỌC ĐẶC TRƯNG ---")
df = pd.read_csv(INPUT_FILE)
print(f" Đã tải dữ liệu: {df.shape}")

# ==============================================================================
# 1. XỬ LÝ GIÁ TRỊ RỖNG (NULL HANDLING)
# ==============================================================================
print("\n[1] Xử lý giá trị khuyết thiếu (Null)...")

# Danh sách biến số liên tục (Continuous Vars)
numeric_cols = ['Age', 'BMI', 'PSS_Score', 'MENQOL_Score', 'Meno_Age_Clean', 'Meno_Duration_New']
# Danh sách biến phân loại (Categorical Vars) - Đã mã hóa ở bước 1
cat_cols = ['Education_Code', 'Income_Code', 'Job_Code', 'Marital_Code', 'Meno_Status']

# Điền Null cho biến số bằng Median (Trung vị)
for col in numeric_cols:
    if col in df.columns:
        median_val = df[col].median()
        if df[col].isnull().sum() > 0:
            print(f"   -> Điền {df[col].isnull().sum()} dòng Null ở cột '{col}' bằng Median ({median_val:.2f})")
            df[col] = df[col].fillna(median_val)

# Điền Null cho biến phân loại bằng Mode (Giá trị xuất hiện nhiều nhất)
for col in cat_cols:
    if col in df.columns:
        mode_val = df[col].mode()[0]
        if df[col].isnull().sum() > 0:
            print(f"   -> Điền {df[col].isnull().sum()} dòng Null ở cột '{col}' bằng Mode ({mode_val})")
            df[col] = df[col].fillna(mode_val)
```

```
# ==============================================================================
# 2. MÃ HÓA NHỊ PHÂN BIẾN VĂN BẢN (TEXT TO BINARY)
# ==============================================================================
print("\n[2] Mã hóa biến văn bản (Text -> 0/1)...")

text_binary_cols = ['Chronic_Disease', 'Supp_Calcium', 'Supp_Omega3', 'Exercise']

for col in text_binary_cols:
    if col in df.columns:
        # Quy tắc: Nếu ô chứa chữ (khác nan/trống/không) -> 1, ngược lại -> 0
        # Chuyển về chữ thường để so sánh
        def encode_binary(val):
            val_str = str(val).strip().lower()
            if val_str in ['nan', '', '0', 'none']: return 0
            if 'không' in val_str: return 0  # Từ khóa phủ định tiếng Việt
            return 1  # Có dữ liệu text (VD: "Xơ gan", "Có", "Thường xuyên") -> 1



        # Tạo tên cột mới (VD: Chronic_Disease_Code) hoặc đè lên cột cũ
        # Ở đây ta đè lên cột cũ để gọn dữ liệu
        df[col] = df[col].apply(encode_binary)
        print(f"   -> Đã mã hóa '{col}': {df[col].value_counts().to_dict()}")

# ==============================================================================
# 3. KIỂM ĐỊNH PHƯƠNG SAI & LOẠI BỎ (LOW VARIANCE FILTER)
# ==============================================================================
print("\n[3] Kiểm định & Loại bỏ biến ít thông tin...")

cols_to_drop = []
THRESHOLD = 0.95  # Ngưỡng 95% (Nếu 95% người giống hệt nhau thì biến này vô dụng)

for col in text_binary_cols:
    if col in df.columns:
        counts = df[col].value_counts(normalize=True)  # Tính tỷ lệ phần trăm
        max_ratio = counts.max()  # Lấy tỷ lệ của giá trị phổ biến nhất

        if max_ratio > THRESHOLD:
            print(f"    Cảnh báo: Cột '{col}' có {max_ratio:.1%} giá trị giống nhau -> LOẠI BỎ.")
            cols_to_drop.append(col)
        else:
            print(f"    Cột '{col}' có phân bố tốt (Dominant: {max_ratio:.1%}) -> GIỮ LẠI.")

if cols_to_drop:
    df.drop(columns=cols_to_drop, inplace=True)
    print(f"   -> Đã xóa {len(cols_to_drop)} cột: {cols_to_drop}")
else:
    print("   -> Không có cột nào bị loại bỏ.")

# ==============================================================================
# 4. LƯU KẾT QUẢ
# ==============================================================================
```

```
# Chỉ giữ lại các cột dạng số đã xử lý để đưa vào Machine Learning
# (Loại bỏ các cột text gốc như 'Job', 'Income'...)
final_cols = [c for c in df.columns if df[c].dtype in ['int64', 'float64', 'int32', 'float32']]

# Hoặc nếu bạn muốn giữ lại tất cả để đối chiếu thì dùng dòng dưới (nhưng cẩn thận khi train model)
df_final = df  # df[final_cols]

df_final.to_csv(OUTPUT_FILE, index=False)
print("\n" + "=" * 60)
print(f" HOÀN TẤT GIAI ĐOẠN 2! File kết quả: {OUTPUT_FILE}")
print(f" Kích thước cuối cùng: {df_final.shape}")
print("=" * 60)

# In thử 5 dòng đầu
print(df_final[
        ['Age', 'BMI', 'PSS_Score', 'MENQOL_Score'] + [c for c in text_binary_cols if c in df_final.columns]].head())
```

FILE: 3.5.py

```
import pandas as pd
import numpy as np
import os
from imblearn.over_sampling import SMOTE
from collections import Counter

# --- CẤU HÌNH ---
INPUT_TRAIN = "train_data.csv"
OUTPUT_TRAIN_BALANCED = "train_data_balanced.csv"

if not os.path.exists(INPUT_TRAIN):
    print(f" LỖI: Không tìm thấy file '{INPUT_TRAIN}'")
    exit()

print("--- GIAI ĐOẠN 3.5: TĂNG CƯỜNG DỮ LIỆU (FIXED) ---")
df_train = pd.read_csv(INPUT_TRAIN)

# --- BƯỚC QUAN TRỌNG: CHỈ LẤY CÁC CỘT DẠNG SỐ ---
# SMOTE không thể xử lý các cột văn bản như 'Độc thân', 'Nội trợ'...
df_numeric = df_train.select_dtypes(include=[np.number])

# Tách đặc trưng (X) và nhãn cụm (y)
X = df_numeric.drop(columns=['Cluster'])
y = df_numeric['Cluster']

print(f" Phân bố cụm trước khi tăng cường: {Counter(y)}")

# Khởi tạo SMOTE
# k_neighbors=2 vì Cluster 0 chỉ có 5 mẫu (k < n_samples)
smote = SMOTE(sampling_strategy='auto', k_neighbors=2, random_state=42)

# Thực hiện sinh dữ liệu ảo
X_resampled, y_resampled = smote.fit_resample(X, y)

print(f" Phân bố cụm sau khi tăng cường: {Counter(y_resampled)}")

# Gộp lại thành dataframe mới
df_balanced = pd.concat([pd.DataFrame(X_resampled, columns=X.columns),
                         pd.Series(y_resampled, name='Cluster')], axis=1)

# Làm tròn các giá trị mã hóa (vì SMOTE sinh số thực kiểu 1.2, 1.8...)
# Chuyển về số nguyên cho các cột Code và Score
cols_to_round = [c for c in df_balanced.columns if '_Code' in c or 'PSS_Score' in c or 'Meno_Status' in c]
for col in cols_to_round:
    df_balanced[col] = df_balanced[col].round().astype(int)

# Lưu file train mới (chỉ chứa dữ liệu số, sẵn sàng cho ML)
df_balanced.to_csv(OUTPUT_TRAIN_BALANCED, index=False)
```

```
print("\n" + "="*60)
print(f" HOÀN TẤT! Đã tạo ra: {OUTPUT_TRAIN_BALANCED}")
print(f"   Dữ liệu Cluster 0 đã được cân bằng lên {Counter(y_resampled)[0]} mẫu.")
print("="*60)
```

FILE: 3.py

```
import pandas as pd
import numpy as np
import os
import warnings
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split

warnings.filterwarnings('ignore')

# --- CẤU HÌNH ---
INPUT_FILE = "clean_data_final.csv"
OUTPUT_TRAIN = "train_data.csv"
OUTPUT_TEST = "test_data.csv"

if not os.path.exists(INPUT_FILE):
    print(f" LỖI: Không tìm thấy file '{INPUT_FILE}'.")
    exit()

print("--- BẮT ĐẦU GIAI ĐOẠN 3: PHÂN CỤM & CHIA TẬP DỮ LIỆU ---")
df = pd.read_csv(INPUT_FILE)
print(f" Đã tải dữ liệu: {df.shape}")

# ==============================================================================
# 1. CHUẨN BỊ DỮ LIỆU ĐỂ PHÂN CỤM
# ==============================================================================
# Chúng ta chỉ dùng các biến NHÂN KHẨU HỌC để phân nhóm (Profile)
# Không dùng PSS_Score hay MENQOL_Score để phân cụm (để tránh rò rỉ dữ liệu)
clustering_cols = [
    'Age', 'BMI',
    'Education_Code', 'Income_Code', 'Job_Code', 'Marital_Code',
    'Meno_Duration_New', 'Meno_Status'
]

# Kiểm tra xem các cột này có đủ không
valid_cluster_cols = [c for c in clustering_cols if c in df.columns]
print(f"\n[1] Các đặc trưng dùng để phân cụm ({len(valid_cluster_cols)} biến):")
print(f"   {valid_cluster_cols}")

X_cluster = df[valid_cluster_cols]

# Chuẩn hóa dữ liệu (Scaling)
# Bước này bắt buộc vì 'Age' (40-60) lớn hơn nhiều so với 'Income_Code' (1-4)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_cluster)

# ==============================================================================
# 2. THỰC HIỆN K-MEANS CLUSTERING
```

```
# ================================================================================
print("\n[2] Đang chạy K-Means (k=3)...")

# Chọn k=3 (3 nhóm người tiêu biểu)
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
df['Cluster'] = kmeans.fit_predict(X_scaled)

# Thống kê số lượng mỗi nhóm
counts = df['Cluster'].value_counts().sort_index()
print("   -> Số lượng thành viên mỗi nhóm:")
print(counts)

# ================================================================================
# 3. ĐỊNH DANH CÁC NHÓM (CLUSTERING PROFILING)
# ================================================================================
print("\n[3] Đặc điểm trung bình của từng nhóm:")

# Tính giá trị trung bình của các biến nhân khẩu học theo từng cụm
profile = df.groupby('Cluster')[valid_cluster_cols].mean()

# Hiển thị làm tròn 2 chữ số
print(profile.round(2).T)

print("\n   -> NHẬN XÉT SƠ BỘ (Dựa trên số liệu trên):")
for i in range(3):
    age = profile.loc[i, 'Age']
    income = profile.loc[i, 'Income_Code'] if 'Income_Code' in profile.columns else 0
    meno_dur = profile.loc[i, 'Meno_Duration_New']
    print(f"      * Cluster {i}: Tuổi TB ~{age:.1f}, Thu nhập mức ~{income:.1f}, Mãn kinh ~{meno_dur:.1f} năm.")

# ================================================================================
# 4. CHIA TẬP TRAIN / TEST (STRATIFIED SPLIT)
# ================================================================================
print("\n[4] Chia tập Train/Test (Tỷ lệ 80/20, Phân tầng theo Cluster)...")

# stratify=df['Cluster'] đảm bảo tập Test có đủ đại diện của cả 3 nhóm
train_df, test_df = train_test_split(
    df,
    test_size=0.2,
    random_state=42,
    stratify=df['Cluster']
)

print(f"   -> Tập Train: {train_df.shape[0]} mẫu")
print(f"   -> Tập Test : {test_df.shape[0]} mẫu")

# Kiểm tra tỷ lệ nhóm trong tập Test
test_counts = test_df['Cluster'].value_counts(normalize=True).sort_index()
print("   -> Tỷ lệ phân bố nhóm trong tập Test (nên tương đương tập gốc):")
print(test_counts.round(2))
```

```
# ============================================================================
# 5. LƯU KẾT QUẢ
# ============================================================================
train_df.to_csv(OUTPUT_TRAIN, index=False)
test_df.to_csv(OUTPUT_TEST, index=False)

print("\n" + "="*60)
print(f" HOÀN TẤT GIAI ĐOẠN 3!")
print(f"    File Train: {OUTPUT_TRAIN}")
print(f"    File Test : {OUTPUT_TEST}")
print("="*60)
```

FILE: 4.py

```
import pandas as pd
import numpy as np
import os
import warnings
from sklearn.model_selection import KFold, cross_val_score
from sklearn.metrics import r2_score, mean_absolute_error
from sklearn.preprocessing import StandardScaler  # Thêm để chuẩn hóa dữ liệu
# Import các mô hình
from sklearn.linear_model import LinearRegression, Lasso, Ridge, ElasticNet
from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor

warnings.filterwarnings('ignore')

# --- CẤU HÌNH ---
# Sử dụng file đã được cân bằng dữ liệu từ bước 3.5
INPUT_TRAIN = "train_data_balanced.csv"
OUTPUT_REPORT = "model_selection_report.csv"

if not os.path.exists(INPUT_TRAIN):
    print(f" LỖI: Không tìm thấy file '{INPUT_TRAIN}'")
    print(" Hãy chạy file 'src1/3.5.py' trước để tạo dữ liệu cân bằng.")
    exit()

print("--- BẮT ĐẦU GIAI ĐOẠN 4: LAZY PREDICT (LOCAL SELECTION) ---")
train_df = pd.read_csv(INPUT_TRAIN)
print(f" Đã tải tập Train cân bằng: {train_df.shape}")

# Danh sách các đặc trưng đầu vào (Chỉ lấy các cột số đã encode)
features = ['Age', 'BMI', 'Education_Code', 'Income_Code', 'Job_Code', 'Marital_Code',
            'Meno_Duration_New', 'Meno_Status', 'Chronic_Disease']

targets = ['PSS_Score', 'MENQOL_Score']


def get_models():
    """Trả về danh sách mô hình ứng viên"""
    return {
        'Lasso': Lasso(random_state=42),
        'Ridge': Ridge(random_state=42),
        'ElasticNet': ElasticNet(random_state=42),
        'RandomForest': RandomForestRegressor(n_estimators=100, random_state=42),
        'ExtraTrees': ExtraTreesRegressor(n_estimators=100, random_state=42),
        'SVR': SVR(),
        'KNN': KNeighborsRegressor(),
        'GradientBoosting': GradientBoostingRegressor(random_state=42)
    }
```

```
# ================================================================================
# CHẠY SÀNG LỌC (SCREENING LOOP)
# ================================================================================
results = []

# Khởi tạo scaler để chuẩn hóa dữ liệu đầu vào
scaler = StandardScaler()

for c_id in sorted(train_df['Cluster'].unique()):
    print(f"\n" + "=" * 50)
    print(f" CLUSTER {c_id}")

    c_data = train_df[train_df['Cluster'] == c_id]
    n_samples = len(c_data)
    print(f"   -> Số lượng mẫu sau Oversampling: {n_samples}")

    candidate_models = get_models()

    for target in targets:
        print(f"\n    Mục tiêu: {target}")
        best_score = -np.inf
        best_model_name = "None"

        # Chuẩn bị X, y
        X = c_data[features]
        y = c_data[target]

        # CHUẨN HÓA DỮ LIỆU CỤC BỘ (Quan trọng cho SVR/KNN)
        X_scaled = scaler.fit_transform(X)

        print(f"      |{'Model':<20}|{'R2 (CV)':<10}|{'MAE':<10}|")
        print(f"      |{'-' * 20}|{'-' * 10}|{'-' * 10}|")

        for name, model in candidate_models.items():
            try:
                # Dùng CV=5 vì dữ liệu bây giờ đã đủ lớn nhờ SMOTE
                cv = KFold(n_splits=5, shuffle=True, random_state=42)

                cv_scores = cross_val_score(model, X_scaled, y, cv=cv, scoring='r2')
                r2 = cv_scores.mean()

                mae_scores = -cross_val_score(model, X_scaled, y, cv=cv, scoring='neg_mean_absolute_error')
                mae = mae_scores.mean()

                print(f"      |{name:<20}|{r2:<10.4f}|{mae:<10.4f}|")

                if r2 > best_score:
                    best_score = r2
                    best_model_name = name
```

```
                best_mae = mae

            except Exception as e:
                pass

        print(f"       VÔ ĐỊCH: {best_model_name} (R2={best_score:.4f})")

        results.append({
            'Cluster': c_id,
            'Target': target,
            'Best_Model': best_model_name,
            'Best_R2': best_score,
            'Best_MAE': best_mae,
            'Samples': n_samples
        })

# ================================================================================
# TỔNG HỢP BÁO CÁO
# ================================================================================
print("\n" + "=" * 60)
print(" BẢNG TỔNG SẮP MÔ HÌNH TỐT NHẤT (LEADERBOARD)")
print("=" * 60)
report_df = pd.DataFrame(results)
print(report_df)

report_df.to_csv(OUTPUT_REPORT, index=False)
print(f"\n Đã lưu báo cáo tại: {OUTPUT_REPORT}")
```

FILE: 5.5.py

```
import pandas as pd
import numpy as np
import os
import warnings
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score, mean_absolute_error
from sklearn.preprocessing import StandardScaler

warnings.filterwarnings('ignore')

# --- CẤU HÌNH ---
# Sử dụng tập Train đã được cân bằng bằng SMOTE
INPUT_TRAIN = "train_data_balanced.csv"
INPUT_TEST = "test_data.csv"

# Danh sách 25 câu hỏi tiềm năng (Đã loại bỏ Chronic_Disease vì đưa vào Demo Feats)
POTENTIAL_QUESTIONS = [
    'MEN_HotFlash', 'MEN_Memory', 'MEN_Sleep', 'MEN_Headache', 'MEN_MuscleAche',
    'MEN_Fatigue', 'MEN_Weight', 'MEN_Skin', 'MEN_Depressed', 'MEN_Impatient',
    'MEN_Urine', 'MEN_Libido', 'PSS_1', 'PSS_2', 'PSS_3', 'PSS_4', 'PSS_5',
    'PSS_6', 'PSS_7', 'PSS_8', 'PSS_9', 'PSS_10', 'Supp_Calcium', 'Supp_Omega3'
]

# Thêm Chronic_Disease vào nhóm đặc trưng nền tảng
DEMO_FEATS = ['Age', 'BMI', 'Education_Code', 'Income_Code', 'Job_Code', 'Marital_Code',
              'Meno_Duration_New', 'Meno_Status', 'Chronic_Disease']


def run_fine_tuning():
    if not os.path.exists(INPUT_TRAIN):
        print(f" LỖI: Không tìm thấy file '{INPUT_TRAIN}'. Hãy chạy bước 3.5 trước.")
        return

    train_df = pd.read_csv(INPUT_TRAIN)
    test_df = pd.read_csv(INPUT_TEST)

    targets = ['PSS_Score', 'MENQOL_Score']
    scaler = StandardScaler()

    print("--- GIAI ĐOẠN 5: TÌM CÂU HỎI VÀNG & FINE-TUNING (SMOTE ENHANCED) ---")

    for target in targets:
        print(f"\n" + "=" * 60)
        print(f" TỐI ƯU HÓA MỤC TIÊU: {target}")
        print("=" * 60)

        for c_id in sorted(train_df['Cluster'].unique()):
```

```
c_train = train_df[train_df['Cluster'] == c_id]
c_test = test_df[test_df['Cluster'] == c_id]

# --- BƯỚC 1: TÌM CÂU HỎI VÀNG (FEATURE IMPORTANCE) ---
# Sử dụng tập Train đã cân bằng giúp việc tìm feature importance chính xác hơn
selector = ExtraTreesRegressor(n_estimators=100, random_state=42)
valid_potential = [q for q in POTENTIAL_QUESTIONS if q in c_train.columns]

selector.fit(c_train[valid_potential], c_train[target])
importances = pd.Series(selector.feature_importances_, index=valid_potential)
gold_qs = importances.nlargest(5).index.tolist()

print(f"\n Cluster {c_id} (Train: {len(c_train)} mẫu | Test: {len(c_test)} mẫu):")
print(f"   -> 5 Câu hỏi vàng: {gold_qs}")

# --- BƯỚC 2: FINE-TUNING VỚI GRID SEARCH ---
final_features = DEMO_FEATS + gold_qs
X_train = c_train[final_features]
y_train = c_train[target]

# Chuẩn hóa dữ liệu trước khi train
X_train_scaled = scaler.fit_transform(X_train)

param_grid = {
    'n_estimators': [100, 300],
    'max_depth': [4, 6, 8, None],
    'min_samples_leaf': [2, 4],
    'max_features': ['sqrt', 'log2', None]
}

grid = GridSearchCV(ExtraTreesRegressor(random_state=42), param_grid, cv=3, scoring='r2')
grid.fit(X_train_scaled, y_train)

best_model = grid.best_estimator_
print(f"   -> Best Params: {grid.best_params_}")

# --- BƯỚC 3: ĐÁNH GIÁ TRÊN TẬP TEST THẬT ---
if len(c_test) > 0:
    X_test_scaled = scaler.transform(c_test[final_features])
    y_test = c_test[target]
    y_pred = best_model.predict(X_test_scaled)

    # Tính R2 và MAE
    # Nếu chỉ có 1 mẫu, R2 sẽ là nan, ta cần check để tránh in ra nan
    if len(y_test) > 1:
        r2 = r2_score(y_test, y_pred)
        print(f"    R2 trên tập TEST: {r2:.4f}")
    else:
        print(f"    Chỉ có 1 mẫu test: Thực tế={y_test.values[0]:.2f}, Dự báo={y_pred[0]:.2f}")

    mae = mean_absolute_error(y_test, y_pred)
```

```
            print(f"    MAE trên tập TEST: {mae:.4f}")
        else:
            print("    Không có dữ liệu test cho cụm này.")


if __name__ == "__main__":
    run_fine_tuning()
```

FILE: 5.6.py

```
import pandas as pd
import numpy as np
import os
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.feature_selection import RFE
from sklearn.preprocessing import StandardScaler

# --- CAU HINH ---
INPUT_TRAIN = "train_data_balanced.csv"

# Dong bo voi code 5.5: Loai bo Chronic_Disease khoi day vi da dua vao Demo Feats
POTENTIAL_QUESTIONS = [
    'MEN_HotFlash', 'MEN_Memory', 'MEN_Sleep', 'MEN_Headache', 'MEN_MuscleAche',
    'MEN_Fatigue', 'MEN_Weight', 'MEN_Skin', 'MEN_Depressed', 'MEN_Impatient',
    'MEN_Urine', 'MEN_Libido', 'PSS_1', 'PSS_2', 'PSS_3', 'PSS_4', 'PSS_5',
    'PSS_6', 'PSS_7', 'PSS_8', 'PSS_9', 'PSS_10', 'Supp_Calcium', 'Supp_Omega3'
]

def run_rfe_selection():
    if not os.path.exists(INPUT_TRAIN):
        print("Khong tim thay file train_data_balanced.csv")
        return

    df = pd.read_csv(INPUT_TRAIN)
    targets = ['PSS_Score', 'MENQOL_Score']
    scaler = StandardScaler()

    print("--- GIAI DOAN 5.6: CHON CAU HOI VANG BANG RFE ---")

    for target in targets:
        print(f"\nDUA TREN RFE CHO: {target}")
        for c_id in sorted(df['Cluster'].unique()):
            c_data = df[df['Cluster'] == c_id]
            X = c_data[POTENTIAL_QUESTIONS]
            y = c_data[target]
            X_scaled = scaler.fit_transform(X)

            estimator = ExtraTreesRegressor(n_estimators=100, random_state=42)
            selector = RFE(estimator, n_features_to_select=5, step=1)
            selector = selector.fit(X_scaled, y)

            gold_qs_rfe = np.array(POTENTIAL_QUESTIONS)[selector.support_].tolist()
            print(f"   Cluster {c_id}: {gold_qs_rfe}")

if __name__ == "__main__":
    run_rfe_selection()
```

FILE: 5.7.py

```
import pandas as pd
import numpy as np
import os
from sklearn.feature_selection import mutual_info_regression

# --- CAU HINH ---
INPUT_TRAIN = "train_data_balanced.csv"

POTENTIAL_QUESTIONS = [
    'MEN_HotFlash', 'MEN_Memory', 'MEN_Sleep', 'MEN_Headache', 'MEN_MuscleAche',
    'MEN_Fatigue', 'MEN_Weight', 'MEN_Skin', 'MEN_Depressed', 'MEN_Impatient',
    'MEN_Urine', 'MEN_Libido', 'PSS_1', 'PSS_2', 'PSS_3', 'PSS_4', 'PSS_5',
    'PSS_6', 'PSS_7', 'PSS_8', 'PSS_9', 'PSS_10', 'Supp_Calcium', 'Supp_Omega3'
]

def run_mi_selection():
    if not os.path.exists(INPUT_TRAIN):
        print("Khong tim thay file train_data_balanced.csv")
        return

    df = pd.read_csv(INPUT_TRAIN)
    targets = ['PSS_Score', 'MENQOL_Score']

    print("--- GIAI DOAN 5.7: CHON CAU HOI VANG BANG MUTUAL INFORMATION ---")

    for target in targets:
        print(f"\nDUA TREN MUTUAL INFO CHO: {target}")
        for c_id in sorted(df['Cluster'].unique()):
            c_data = df[df['Cluster'] == c_id]
            X = c_data[POTENTIAL_QUESTIONS]
            y = c_data[target]

            mi_scores = mutual_info_regression(X, y, discrete_features=True, random_state=42)
            mi_results = pd.Series(mi_scores, index=POTENTIAL_QUESTIONS)
            gold_qs_mi = mi_results.nlargest(5).index.tolist()
            print(f"   Cluster {c_id}: {gold_qs_mi}")

if __name__ == "__main__":
    run_mi_selection()
```

FILE: 5.8.py

```
import pandas as pd
import numpy as np
import os
import joblib
from collections import Counter
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.feature_selection import RFE, mutual_info_regression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score, mean_absolute_error
from sklearn.preprocessing import StandardScaler
import warnings

warnings.filterwarnings('ignore')

# --- CAU HINH ---
INPUT_TRAIN = "train_data_balanced.csv"
INPUT_TEST = "test_data.csv"

POTENTIAL_QUESTIONS = [
    'MEN_HotFlash', 'MEN_Memory', 'MEN_Sleep', 'MEN_Headache', 'MEN_MuscleAche',
    'MEN_Fatigue', 'MEN_Weight', 'MEN_Skin', 'MEN_Depressed', 'MEN_Impatient',
    'MEN_Urine', 'MEN_Libido', 'PSS_1', 'PSS_2', 'PSS_3', 'PSS_4', 'PSS_5',
    'PSS_6', 'PSS_7', 'PSS_8', 'PSS_9', 'PSS_10', 'Supp_Calcium', 'Supp_Omega3'
]

DEMO_FEATS = ['Age', 'BMI', 'Education_Code', 'Income_Code', 'Job_Code', 'Marital_Code',
              'Meno_Duration_New', 'Meno_Status', 'Chronic_Disease']


def get_ensemble_gold_questions(X, y):
    et = ExtraTreesRegressor(n_estimators=100, random_state=42)
    et.fit(X, y)
    et_feats = pd.Series(et.feature_importances_, index=X.columns).nlargest(5).index.tolist()

    selector = RFE(ExtraTreesRegressor(n_estimators=100, random_state=42), n_features_to_select=5)
    selector.fit(X, y)
    rfe_feats = X.columns[selector.support_].tolist()

    mi_scores = mutual_info_regression(X, y, discrete_features=True, random_state=42)
    mi_feats = pd.Series(mi_scores, index=X.columns).nlargest(5).index.tolist()

    all_selected = et_feats + rfe_feats + mi_feats
    counts = Counter(all_selected)
    ensemble_gold = [item for item, count in counts.most_common(5)]

    return ensemble_gold, {"ExtraTrees": et_feats, "RFE": rfe_feats, "MutualInfo": mi_feats}
```

```
def run_ensemble_pipeline():
    if not os.path.exists(INPUT_TRAIN) or not os.path.exists(INPUT_TEST):
        print("Loi: Khong tim thay file du lieu.")
        return

    train_df = pd.read_csv(INPUT_TRAIN)
    test_df = pd.read_csv(INPUT_TEST)
    targets = ['PSS_Score', 'MENQOL_Score']
    scaler = StandardScaler()

    print("--- GIAI DOAN 5.8: ENSEMBLE SELECTION & FINE-TUNING ---")

    for target in targets:
        print(f"\n============================================================")
        print(f"TOI UU HOA CHO: {target}")
        print("============================================================")

        for c_id in sorted(train_df['Cluster'].unique()):
            c_train = train_df[train_df['Cluster'] == c_id]
            c_test = test_df[test_df['Cluster'] == c_id]

            X_train_raw = c_train[POTENTIAL_QUESTIONS]
            y_train = c_train[target]

            gold_qs, details = get_ensemble_gold_questions(X_train_raw, y_train)

            print(f"\nCluster {c_id} (Train: {len(c_train)} | Test: {len(c_test)}):")
            print(f"   + ExtraTrees: {details['ExtraTrees']}")
            print(f"   + RFE       : {details['RFE']}")
            print(f"   + MutualInfo: {details['MutualInfo']}")
            print(f"   => KET QUA BAU CHON (GOLD): {gold_qs}")

            final_features = DEMO_FEATS + gold_qs
            X_train_final = c_train[final_features]
            X_train_scaled = scaler.fit_transform(X_train_final)

            param_grid = {
                'n_estimators': [100, 300],
                'max_depth': [6, 8, None],
                'min_samples_leaf': [2, 4],
                'max_features': ['sqrt', None]
            }

            grid = GridSearchCV(ExtraTreesRegressor(random_state=42), param_grid, cv=3, scoring='r2')
            grid.fit(X_train_scaled, y_train)
            best_model = grid.best_estimator_

            if len(c_test) > 0:
                X_test_scaled = scaler.transform(c_test[final_features])
                y_test = c_test[target]
                y_pred = best_model.predict(X_test_scaled)
```

```
            # Logic hien thi chi tiet tuong tu code 5.5
            if len(y_test) > 1:
                r2 = r2_score(y_test, y_pred)
                print(f"   Chỉ số R2 TEST: {r2:.4f}")
            else:
                print(f"   Thông báo: Chỉ có 1 mẫu test. Thực tế={y_test.values[0]:.2f}, Dự báo={y_pred[0]:.2f}")

            mae = mean_absolute_error(y_test, y_pred)
            print(f"   Chỉ số MAE TEST: {mae:.4f}")


if __name__ == "__main__":
    run_ensemble_pipeline()
```

FILE: 5.py

```
import pandas as pd
import numpy as np
import os
import joblib
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score

# --- CẤU HÌNH ---
INPUT_TRAIN = "train_data.csv"
INPUT_TEST = "test_data.csv"

# Danh sách 26 câu hỏi tiềm năng (Symptoms & Lifestyle)
POTENTIAL_QUESTIONS = [
    'MEN_HotFlash', 'MEN_Memory', 'MEN_Sleep', 'MEN_Headache', 'MEN_MuscleAche',
    'MEN_Fatigue', 'MEN_Weight', 'MEN_Skin', 'MEN_Depressed', 'MEN_Impatient',
    'MEN_Urine', 'MEN_Libido', 'PSS_1', 'PSS_2', 'PSS_3', 'PSS_4', 'PSS_5',
    'PSS_6', 'PSS_7', 'PSS_8', 'PSS_9', 'PSS_10', 'Chronic_Disease',
    'Supp_Calcium', 'Supp_Omega3'
]

DEMO_FEATS = ['Age', 'BMI', 'Education_Code', 'Income_Code', 'Job_Code', 'Marital_Code',
              'Meno_Duration_New', 'Meno_Status']


def run_fine_tuning():
    train_df = pd.read_csv(INPUT_TRAIN)
    test_df = pd.read_csv(INPUT_TEST)

    targets = ['PSS_Score', 'MENQOL_Score']

    print("--- GIAI ĐOẠN 5: TÌM CÂU HỎI VÀNG & FINE-TUNING ---")

    for target in targets:
        print(f"\n TỐI ƯU HÓA CHO: {target}")

        for c_id in sorted(train_df['Cluster'].unique()):
            c_train = train_df[train_df['Cluster'] == c_id]
            if len(c_train) < 5: continue

            # --- BƯỚC 1: TÌM CÂU HỎI VÀNG (FEATURE IMPORTANCE) ---
            # Dùng ExtraTrees để "quét" xem câu nào ảnh hưởng nhất đến Target
            selector = ExtraTreesRegressor(n_estimators=100, random_state=42)
            # Lọc các cột hiện có trong data
            valid_potential = [q for q in POTENTIAL_QUESTIONS if q in c_train.columns]

            selector.fit(c_train[valid_potential], c_train[target])
            importances = pd.Series(selector.feature_importances_, index=valid_potential)
```

```
        gold_qs = importances.nlargest(5).index.tolist()

        print(f"\n    Cluster {c_id}:")
        print(f"      -> 5 Câu hỏi vàng: {gold_qs}")

        # --- BƯỚC 2: FINE-TUNING VỚI CÂU HỎI VÀNG ---
        # Đầu vào mới = Nhân khẩu học + 5 câu hỏi vàng
        final_features = DEMO_FEATS + gold_qs
        X_train = c_train[final_features]
        y_train = c_train[target]

        # Lưới tham số để GridSearch
        param_grid = {
            'n_estimators': [100, 300],
            'max_depth': [4, 6, None],
            'min_samples_leaf': [2, 4]
        }

        grid = GridSearchCV(ExtraTreesRegressor(random_state=42), param_grid, cv=3, scoring='r2')
        grid.fit(X_train, y_train)

        best_model = grid.best_estimator_

        # --- BƯỚC 3: KIỂM CHỨNG TRÊN TẬP TEST ---
        c_test = test_df[test_df['Cluster'] == c_id]
        if len(c_test) > 0:
            X_test = c_test[final_features]
            y_test = c_test[target]
            y_pred = best_model.predict(X_test)
            score = r2_score(y_test, y_pred)
            print(f"       R2 trên tập TEST: {score:.4f}")


if __name__ == "__main__":
    run_fine_tuning()
```

FILE: 6.1.py

```
import pandas as pd
import numpy as np
import joblib
import os
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# --- CẤU HÌNH ---
REAL_DATA = "clean_data_final.csv"
TRAIN_BALANCED = "train_data_balanced.csv"
MODEL_PATH = "final_health_advisor.pkl"

# 1. CẬP NHẬT CHIẾN LƯỢC TỪ KẾT QUẢ ENSEMBLE 5.8 (GOLD)
STRATEGY = {
    0: {
        'PSS': ['MEN_Memory', 'MEN_Libido', 'PSS_4', 'PSS_5', 'PSS_9'],
        'MEN': ['MEN_Sleep', 'PSS_8', 'PSS_2', 'MEN_HotFlash', 'PSS_1']
    },
    1: {
        'PSS': ['PSS_6', 'PSS_1', 'PSS_3', 'PSS_5', 'PSS_10'],
        'MEN': ['MEN_Depressed', 'MEN_Fatigue', 'MEN_Libido', 'PSS_3', 'PSS_1']
    },
    2: {
        'PSS': ['PSS_3', 'PSS_10', 'PSS_2', 'PSS_8', 'PSS_9'],
        'MEN': ['MEN_Sleep', 'MEN_Depressed', 'PSS_3', 'MEN_Impatient', 'MEN_Weight']
    }
}

# 2. CẬP NHẬT THAM SỐ TỐI ƯU TỪ GRIDSEARCH (FINE-TUNING)
BEST_PARAMS = {
    0: {
        'PSS': {'n_estimators': 300, 'min_samples_leaf': 2, 'max_depth': None, 'max_features': 'sqrt'},
        'MEN': {'n_estimators': 300, 'min_samples_leaf': 2, 'max_depth': 8, 'max_features': None}
    },
    1: {
        'PSS': {'n_estimators': 100, 'min_samples_leaf': 2, 'max_depth': 6, 'max_features': None},
        'MEN': {'n_estimators': 100, 'min_samples_leaf': 2, 'max_depth': 6, 'max_features': None}
    },
    2: {
        'PSS': {'n_estimators': 300, 'min_samples_leaf': 2, 'max_depth': 8, 'max_features': None},
        'MEN': {'n_estimators': 100, 'min_samples_leaf': 2, 'max_depth': 8, 'max_features': None}
    }
}

DEMO_FEATS = ['Age', 'BMI', 'Education_Code', 'Income_Code', 'Job_Code', 'Marital_Code',
              'Meno_Duration_New', 'Meno_Status', 'Chronic_Disease']
```

```
def train_deployment_system():
    print("--- GIAI ĐOẠN 6: ĐÓNG GÓI HỆ THỐNG CHUYÊN GIA TỐI ƯU ---")

    if not os.path.exists(REAL_DATA) or not os.path.exists(TRAIN_BALANCED):
        print(" Lỗi: Thiếu file dữ liệu cần thiết.")
        return

    df_real = pd.read_csv(REAL_DATA)
    df_balanced = pd.read_csv(TRAIN_BALANCED)

    # 1. Định hình không gian cụm (Dữ liệu THẬT)
    scaler = StandardScaler()
    scaler.fit(df_real[DEMO_FEATS])
    X_real_scaled = scaler.transform(df_real[DEMO_FEATS])

    kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
    kmeans.fit(X_real_scaled)

    # 2. Gán nhãn cụm cho dữ liệu huấn luyện
    X_train_scaled = scaler.transform(df_balanced[DEMO_FEATS])
    df_balanced['Cluster'] = kmeans.predict(X_train_scaled)

    # 3. Huấn luyện các Expert Model với tham số tối ưu
    experts = {}
    for c_id in range(3):
        c_data = df_balanced[df_balanced['Cluster'] == c_id]
        if len(c_data) == 0: continue

        for t_type in ['PSS', 'MEN']:
            target_col = 'PSS_Score' if t_type == 'PSS' else 'MENQOL_Score'
            features = DEMO_FEATS + STRATEGY[c_id][t_type]
            params = BEST_PARAMS[c_id][t_type]

            # Khởi tạo model với tham số riêng biệt cho từng Expert
            model = ExtraTreesRegressor(
                n_estimators=params['n_estimators'],
                min_samples_leaf=params['min_samples_leaf'],
                max_depth=params['max_depth'],
                max_features=params['max_features'],
                random_state=42
            )

            model.fit(c_data[features], c_data[target_col])
            experts[f"expert_{c_id}_{t_type}"] = model
            print(f"    Đã huấn luyện expert_{c_id}_{t_type} thành công.")

    # 4. Đóng gói tài sản
    package = {
        'scaler': scaler,
        'kmeans': kmeans,
```

```
        'experts': experts,
        'strategy': STRATEGY,
        'demo_feats': DEMO_FEATS
    }

    joblib.dump(package, MODEL_PATH)
    print(f"\n HỆ THỐNG ĐÃ SẴN SÀNG: {MODEL_PATH}")


if __name__ == "__main__":
    train_deployment_system()
```

FILE: 6.py

```
import pandas as pd
import numpy as np
import joblib
import os
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# --- CẤU HÌNH ---
DATA_FILE = "clean_data_final.csv"  # Dùng file gốc sạch nhất để tái lập quy trình
MODEL_PATH = "final_health_advisor.pkl"

# Chiến lược đã tối ưu từ bước 5
STRATEGY = {
    0: {'PSS': ['MEN_Memory', 'MEN_Libido', 'PSS_5', 'PSS_4', 'PSS_9'],
        'MEN': ['PSS_2', 'MEN_Sleep', 'MEN_HotFlash', 'PSS_1', 'PSS_8']},
    1: {'PSS': ['PSS_6', 'PSS_1', 'PSS_3', 'PSS_2', 'PSS_5'],
        'MEN': ['MEN_Libido', 'MEN_Depressed', 'MEN_Fatigue', 'PSS_3', 'PSS_1']},
    2: {'PSS': ['PSS_3', 'PSS_10', 'PSS_2', 'PSS_8', 'PSS_9'],
        'MEN': ['MEN_Sleep', 'MEN_Depressed', 'PSS_3', 'MEN_Impatient', 'MEN_Weight']}
}

DEMO_FEATS = ['Age', 'BMI', 'Education_Code', 'Income_Code', 'Job_Code', 'Marital_Code',
              'Meno_Duration_New', 'Meno_Status', 'Chronic_Disease']


def train_deployment_system():
    print("--- GIAI ĐOẠN 6: ĐÓNG GÓI MÔ HÌNH ---")
    df = pd.read_csv(DATA_FILE)

    # 1. Tái lập K-Means & Scaler
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(df[DEMO_FEATS])
    kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
    df['Cluster'] = kmeans.fit_predict(X_scaled)

    # 2. Huấn luyện 6 mô hình chuyên gia (3 cụm x 2 mục tiêu)
    experts = {}
    for c_id in range(3):
        c_data = df[df['Cluster'] == c_id]
        for target_type in ['PSS', 'MEN']:
            target_col = 'PSS_Score' if target_type == 'PSS' else 'MENQOL_Score'
            features = DEMO_FEATS + STRATEGY[c_id][target_type]

            # Sử dụng tham số tốt nhất từ GridSearch bước 5
            model = ExtraTreesRegressor(n_estimators=300, min_samples_leaf=2, random_state=42)
            model.fit(c_data[features], c_data[target_col])
            experts[f"expert_{c_id}_{target_type}"] = model
```

```
    # 3. Đóng gói tất cả vào một file duy nhất
    package = {
        'scaler': scaler,
        'kmeans': kmeans,
        'experts': experts,
        'strategy': STRATEGY,
        'demo_feats': DEMO_FEATS
    }
    joblib.dump(package, MODEL_PATH)
    print(f" Hệ thống đã được đóng gói tại: {MODEL_PATH}")


def predict_health(user_demo_list):
    """Hàm dùng để dự báo cho người dùng mới"""
    package = joblib.load(MODEL_PATH)

    # B1: Xác định cụm
    user_demo_scaled = package['scaler'].transform([user_demo_list])
    c_id = package['kmeans'].predict(user_demo_scaled)[0]

    print(f"\nAI: Bạn thuộc nhóm đối tượng số {c_id}")
    print(f"AI: Để chính xác hơn, vui lòng trả lời thêm các câu hỏi sau: {package['strategy'][c_id]['PSS']}")

    # (Trong thực tế sẽ nhận input tiếp theo ở đây, ở đây ta giả lập)
    return c_id


if __name__ == "__main__":
    train_deployment_system()
    # Thử nghiệm dự báo cho 1 người: 52 tuổi, BMI 24, ĐH, Thu nhập 3, VP, Có gia đình, Mãn kinh 4 năm, Có bệnh nền
    # predict_health([52, 24, 4, 3, 3, 1, 4, 1, 1])
```

FILE: 7.py

```
import pandas as pd
import numpy as np
import joblib
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import r2_score, mean_absolute_error
import os
import warnings

# Tắt các cảnh báo không cần thiết để đầu ra sạch sẽ
warnings.filterwarnings('ignore', category=UserWarning)

# --- CẤU HÌNH ---
MODEL_PATH = "final_health_advisor.pkl"
TEST_FILE = "test_data.csv"

if not os.path.exists(MODEL_PATH) or not os.path.exists(TEST_FILE):
    print(" LỖI: Thiếu file model .pkl hoặc file test_data.csv.")
    exit()


def run_visual_test():
    # 1. Tải hệ thống và dữ liệu
    package = joblib.load(MODEL_PATH)
    test_df = pd.read_csv(TEST_FILE)

    scaler = package['scaler']
    kmeans = package['kmeans']
    experts = package['experts']
    strategy = package['strategy']
    demo_feats = package['demo_feats']

    print(f"--- ĐANG KIỂM THỬ HỆ THỐNG TRÊN {len(test_df)} MẪU TEST ---")

    results = []

    # 2. Duyệt qua từng dòng dữ liệu
    for _, row in test_df.iterrows():
        # SỬA LỖI: Chuyển sang DataFrame để giữ tên cột (Feature names)
        user_demo_df = pd.DataFrame([row[demo_feats]])
        user_demo_scaled = scaler.transform(user_demo_df)

        # Bước 1: Xác định Cụm
        c_id = kmeans.predict(user_demo_scaled)[0]

        res_row = {'Cluster_Real': row['Cluster'], 'Cluster_AI': c_id}

        for target_type in ['PSS', 'MEN']:
```

```
        target_col = 'PSS_Score' if target_type == 'PSS' else 'MENQOL_Score'
        required_feats = demo_feats + strategy[c_id][target_type]

        # SỬA LỖI: Đưa dữ liệu vào dưới dạng DataFrame có tên cột
        input_df = pd.DataFrame([row[required_feats]])

        # Dự báo
        pred = experts[f"expert_{c_id}_{target_type}"].predict(input_df)[0]

        res_row[f'{target_type}_True'] = row[target_col]
        res_row[f'{target_type}_Pred'] = pred

    results.append(res_row)

res_df = pd.DataFrame(results)

# ==============================================================================
# TRỰC QUAN HÓA
# ==============================================================================
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 7))

# PSS_Score
r2_pss = r2_score(res_df['PSS_True'], res_df['PSS_Pred'])
mae_pss = mean_absolute_error(res_df['PSS_True'], res_df['PSS_Pred'])

sns.regplot(x='PSS_True', y='PSS_Pred', data=res_df, ax=ax1,
            scatter_kws={'s': 100, 'alpha': 0.6, 'color': 'teal'},
            line_kws={'color': 'red', 'label': f'R2 = {r2_pss:.4f}'})
ax1.set_title(f'PSS Score: Thực tế vs AI Dự báo\n(MAE: {mae_pss:.2f})', fontsize=12)
ax1.set_xlabel('Giá trị thực tế')
ax1.set_ylabel('Giá trị AI dự báo')
ax1.legend()

# MENQOL_Score
r2_men = r2_score(res_df['MEN_True'], res_df['MEN_Pred'])
mae_men = mean_absolute_error(res_df['MEN_True'], res_df['MEN_Pred'])

sns.regplot(x='MEN_True', y='MEN_Pred', data=res_df, ax=ax2,
            scatter_kws={'s': 100, 'alpha': 0.6, 'color': 'orange'},
            line_kws={'color': 'red', 'label': f'R2 = {r2_men:.4f}'})
ax2.set_title(f'MENQOL Score: Thực tế vs AI Dự báo\n(MAE: {mae_men:.2f})', fontsize=12)
ax2.set_xlabel('Giá trị thực tế')
ax2.legend()

plt.suptitle('KIỂM THỬ HỆ THỐNG ADAPTIVE HEALTH ADVISOR - KẾT QUẢ NGHIỆM THU', fontsize=14, fontweight='bold')
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()

print("\n BẢNG SO SÁNH CHI TIẾT (5 mẫu đầu tiên):")
print(res_df[['PSS_True', 'PSS_Pred', 'MEN_True', 'MEN_Pred']].head().round(2))
```

```
    print(f"\n TỔNG KẾT TOÀN DIỆN:")
    print(f"   * PSS Score   - R2: {r2_pss:.4f}, MAE: {mae_pss:.4f}")
    print(f"   * MENQOL Score - R2: {r2_men:.4f}, MAE: {mae_men:.4f}")


if __name__ == "__main__":
    run_visual_test()
```

FILE: 8.py

```
import pandas as pd
import numpy as np
import joblib
import os
from sklearn.metrics import r2_score, mean_absolute_error
from sklearn.model_selection import cross_val_score, KFold

# --- CẤU HÌNH ---
MODEL_PATH = r"D:\PycharmProjects\PNMK\icatsd2026_menopause_qol\src1\final_health_advisor.pkl"
TRAIN_BALANCED_FILE = "train_data_balanced.csv"  # File dùng để train (đã có SMOTE)
TEST_FILE = "test_data.csv"  # File dữ liệu thật hoàn toàn


def check_model_health():
    if not os.path.exists(MODEL_PATH):
        print(" Không tìm thấy file model!")
        return

    package = joblib.load(MODEL_PATH)
    df_train = pd.read_csv(TRAIN_BALANCED_FILE)
    df_test = pd.read_csv(TEST_FILE)

    experts = package['experts']
    strategy = package['strategy']
    demo_feats = package['demo_feats']

    print("--- HỆ THỐNG KIỂM TRA SỨC KHỎE MÔ HÌNH (MODEL HEALTH CHECK) ---")

    for target in ['PSS', 'MEN']:
        print(f"\n Đánh giá mục tiêu: {target}")

        for c_id in range(3):
            # Lấy model expert cụ thể
            model = experts[f"expert_{c_id}_{target}"]
            feats = demo_feats + strategy[c_id][target]

            # 1. Kiểm tra trên tập Train (Dữ liệu đã học)
            X_train = df_train[df_train['Cluster'] == c_id][feats]
            y_train = df_train[df_train['Cluster'] == c_id]['PSS_Score' if target == 'PSS' else 'MENQOL_Score']
            train_preds = model.predict(X_train)
            train_r2 = r2_score(y_train, train_preds)

            # 2. Kiểm tra trên tập Test (Dữ liệu mới hoàn toàn)
            X_test = df_test[df_test['Cluster'] == c_id][feats]
            y_test = df_test[df_test['Cluster'] == c_id]['PSS_Score' if target == 'PSS' else 'MENQOL_Score']

            if len(y_test) > 0:
                test_preds = model.predict(X_test)
```

```
            # Tính R2 tập test (Chỉ tính nếu > 1 mẫu)
            test_r2 = r2_score(y_test, test_preds) if len(y_test) > 1 else np.nan
            test_mae = mean_absolute_error(y_test, test_preds)

            # KHOẢNG CÁCH GIỮA TRAIN VÀ TEST (OVERFITTING GAP)
            gap = train_r2 - test_r2 if not np.isnan(test_r2) else 0

            print(f"\n Cluster {c_id}:")
            print(f"   - R2 Train: {train_r2:.4f}")
            print(f"   - R2 Test : {test_r2:.4f}" if not np.isnan(test_r2) else f"   - R2 Test : N/A (Mẫu quá ít)")
            print(f"   - MAE Test: {test_mae:.4f}")

            # ĐƯA RA CẢNH BÁO
            if gap > 0.2:
                print("    CẢNH BÁO: Có dấu hiệu Overfitting (Khoảng cách Train-Test quá lớn).")
            elif train_r2 > 0.98 and test_r2 < 0.5:
                print("    NGUY HIỂM: Mô hình đang học thuộc lòng (Data Leakage?).")
            else:
                print("    Mô hình ổn định (Khả năng tổng quát hóa tốt).")
        else:
            print(f"\n Cluster {c_id}: Không có dữ liệu test để đánh giá.")


if __name__ == "__main__":
    check_model_health()
```

FILE: 9.py

```
import pandas as pd
import numpy as np
import joblib
import os
import warnings

warnings.filterwarnings('ignore')

# --- CẤU HÌNH ĐƯỜNG DẪN ---
MODEL_PATH = r"D:\PycharmProjects\PNMK\icatsd2026_menopause_qol\src1\final_health_advisor.pkl"

# --- BỘ TỪ ĐIỂN GIẢI THÍCH (MAPPING DATA) ---
DEMO_MAP = {
    'Age': 'Tuổi (Ví dụ: 54)',
    'BMI': 'Chỉ số khối cơ thể (BMI) (Ví dụ: 22.5)',
    'Education_Code': 'Trình độ học vấn (0: Không học - 5: Sau đại học)',
    'Income_Code': 'Mức thu nhập (1: <5tr - 4: >20tr)',
    'Job_Code': 'Nghề nghiệp (1: Nội trợ - 6: Chuyên gia)',
    'Marital_Code': 'Tình trạng hôn nhân (0: Độc thân - 1: Có gia đình)',
    'Meno_Duration_New': 'Thời gian đã mãn kinh (năm)',
    'Meno_Status': 'Trạng thái mãn kinh (0: Chưa mãn kinh - 1: Đã mãn kinh)',
    'Chronic_Disease': 'Tiền sử bệnh mãn tính (0: Không - 1: Có)'
}

QUESTION_MAP = {
    'PSS_1': 'Cảm thấy khó khăn khi đối mặt với những vấn đề dồn dập',
    'PSS_2': 'Cảm thấy mất kiểm soát đối với những việc quan trọng trong cuộc sống',
    'PSS_3': 'Cảm thấy căng thẳng và lo lắng',
    'PSS_4': 'Cảm thấy tự tin vào khả năng xử lý các vấn đề cá nhân',
    'PSS_5': 'Cảm thấy mọi việc đang diễn ra theo ý muốn của mình',
    'PSS_6': 'Cảm thấy không thể làm hết tất cả các việc cần làm',
    'PSS_7': 'Cảm thấy có thể giữ được bình tĩnh trước những khó khăn',
    'PSS_8': 'Cảm thấy làm chủ được tình hình',
    'PSS_9': 'Cảm thấy tức giận vì những việc nằm ngoài tầm kiểm soát',
    'PSS_10': 'Cảm thấy các khó khăn tích tụ vượt mức có thể giải quyết',
    'MEN_HotFlash': 'Bốc hỏa, nóng bừng mặt',
    'MEN_Memory': 'Giảm trí nhớ, hay quên',
    'MEN_Sleep': 'Mất ngủ hoặc khó ngủ',
    'MEN_Headache': 'Nhức đầu hoặc đau nửa đầu',
    'MEN_MuscleAche': 'Đau cơ hoặc khớp',
    'MEN_Fatigue': 'Cảm thấy mệt mỏi, thiếu năng lượng',
    'MEN_Weight': 'Tăng cân nhanh chóng',
    'MEN_Skin': 'Thay đổi cấu trúc da (khô, nhăn)',
    'MEN_Depressed': 'Cảm thấy lo âu hoặc trầm cảm',
    'MEN_Impatient': 'Cảm thấy mất kiên nhẫn, dễ cáu gắt',
    'MEN_Urine': 'Vấn đề về tiểu tiện (tiểu nhiều, tiểu đêm)',
    'MEN_Libido': 'Thay đổi ham muốn tình dục',
    'Supp_Calcium': 'Sử dụng thực phẩm bổ sung Canxi (0: Không - 1: Có)',
```

```
    'Supp_Omega3': 'Sử dụng thực phẩm bổ sung Omega-3 (0: Không - 1: Có)'
}


def run_app():
    if not os.path.exists(MODEL_PATH):
        print(" Lỗi: Không tìm thấy file model tại đường dẫn đã chỉ định.")
        return

    # 1. Load "Bộ não" AI
    package = joblib.load(MODEL_PATH)
    scaler = package['scaler']
    kmeans = package['kmeans']
    experts = package['experts']
    strategy = package['strategy']
    demo_feats = package['demo_feats']

    print("=" * 60)
    print(" HỆ THỐNG TƯ VẤN SỨC KHỎE THÍCH NGHI (ADAPTIVE AI) ")
    print("=" * 60)

    # 2. Nhập dữ liệu nhân khẩu học
    user_data = {}
    print("\n[BƯỚC 1] VUI LÒNG NHẬP THÔNG TIN CƠ BẢN:")
    for feat in demo_feats:
        label = DEMO_MAP.get(feat, feat)
        val = float(input(f"   + {label}: "))
        user_data[feat] = val

    # 3. Phân loại nhóm người dùng (Clustering)
    user_demo_df = pd.DataFrame([user_data])[demo_feats]
    user_demo_scaled = scaler.transform(user_demo_df)
    c_id = kmeans.predict(user_demo_scaled)[0]

    print(f"\n AI PHÂN TÍCH: Bạn thuộc Nhóm đối tượng {c_id}")
    print("   (Hệ thống đang điều chỉnh các câu hỏi chuyên sâu dành riêng cho bạn...)")

    # 4. Đặt câu hỏi vàng thích nghi (Adaptive Questions)
    # Lấy danh sách các câu hỏi cần thiết cho cả 2 mục tiêu (PSS và MEN)
    gold_qs = list(set(strategy[c_id]['PSS'] + strategy[c_id]['MEN']))

    print("\n[BƯỚC 2] VUI LÒNG ĐÁNH GIÁ CÁC TRIỆU CHỨNG SAU:")
    print("   (Thang điểm PSS: 0-4 | Thang điểm MENQOL: 1-6)")

    for q in gold_qs:
        label = QUESTION_MAP.get(q, q)
        val = float(input(f"   + {label}: "))
        user_data[q] = val

    # 5. Dự báo kết quả cuối cùng
    print("\n" + "=" * 60)
```

```
    print(" KẾT QUẢ DỰ BÁO SỨC KHỎE TỪ AI")
    print("=" * 60)

    for target_type in ['PSS', 'MEN']:
        target_name = "Mức độ Stress (PSS)" if target_type == 'PSS' else "Chất lượng sống (MENQOL)"
        required_feats = demo_feats + strategy[c_id][target_type]

        # Tạo input đúng định dạng DataFrame cho model
        final_input_df = pd.DataFrame([user_data])[required_feats]

        # Gọi chuyên gia tương ứng
        prediction = experts[f"expert_{c_id}_{target_type}"].predict(final_input_df)[0]

        print(f"    {target_name}: {prediction:.2f}")

    print("\nLưu ý: Kết quả mang tính tham khảo, vui lòng tham vấn ý kiến bác sĩ.")
    print("=" * 60)


if __name__ == "__main__":
    run_app()
```