# -\*- coding: utf-8 -\*-  
"""Health AI.ipynb  
  
Automatically generated by Colab.  
  
Original file is located at  
https://colab.research.google.com/drive/1lSXPAPzGfHTFNr7wGVjDbrPSHT4KBHDb  
"""  
  
!pip install streamlit pyngrok pandas numpy plotly  
  
import io  
import json  
import base64  
from datetime import datetime  
  
import numpy as np  
import pandas as pd  
import plotly.express as px  
import gradio as gr  
  
# ---------------------------  
# Utilities & risk model (same logic as your Streamlit code)  
# ---------------------------  
def sigmoid(x):  
return 1 / (1 + np.exp(-x))  
  
def compute\_bmi(height\_cm: float, weight\_kg: float) -> float:  
try:  
if height\_cm is None or weight\_kg is None or float(height\_cm) <= 0:  
return np.nan  
return float(weight\_kg) / ((float(height\_cm) / 100) \*\* 2)  
except Exception:  
return np.nan  
  
def normalize(x, mean, std, lo=None, hi=None):  
if x is None or pd.isna(x):  
return 0.0  
z = (float(x) - mean) / (std if std > 0 else 1.0)  
z = float(np.clip(z, -3, 3))  
if lo is not None and hi is not None:  
return float(np.clip(z, lo, hi))  
return z  
  
def risk\_model(features: dict):  
age = features.get("age", 35)  
bmi = features.get("bmi", np.nan)  
sys\_bp = features.get("systolic\_bp", 120)  
dia\_bp = features.get("diastolic\_bp", 80)  
fbg = features.get("fasting\_glucose", 95)  
hba1c = features.get("hba1c", 5.3)  
chol = features.get("total\_chol", 180)  
hdl = features.get("hdl", 50)  
ldl = features.get("ldl", 110)  
smoker = 1.0 if features.get("smoker", False) else 0.0  
fam = 1.0 if features.get("family\_history", False) else 0.0  
act = features.get("activity\_minutes\_per\_week", 60)  
  
n\_age = normalize(age, 45, 12)  
n\_bmi = normalize(bmi, 26, 5)  
n\_sys = normalize(sys\_bp, 125, 15)  
n\_dia = normalize(dia\_bp, 80, 10)  
n\_fbg = normalize(fbg, 95, 12)  
n\_a1c = normalize(hba1c, 5.4, 0.5)  
n\_chol = normalize(chol, 190, 35)  
n\_hdl = -normalize(hdl, 55, 12)  
n\_ldl = normalize(ldl, 120, 30)  
n\_act = -normalize(act, 120, 60)  
  
d\_score = (  
0.9 \* n\_bmi  
+ 1.2 \* n\_fbg  
+ 1.2 \* n\_a1c  
+ 0.5 \* n\_age  
+ 0.5 \* n\_act  
+ 0.6 \* fam  
+ 0.3 \* smoker  
- 0.2  
)  
diabetes = float(sigmoid(d\_score))  
  
h\_score = (  
0.8 \* n\_age  
+ 0.8 \* n\_chol  
+ 0.7 \* n\_ldl  
+ 0.6 \* n\_sys  
+ 0.3 \* n\_dia  
+ 0.8 \* smoker  
+ 0.6 \* fam  
+ 0.5 \* n\_bmi  
+ 0.5 \* n\_act  
- 0.3  
+ 0.8 \* n\_hdl  
)  
heart = float(sigmoid(h\_score))  
  
ht\_score = (  
1.2 \* n\_sys  
+ 0.8 \* n\_dia  
+ 0.6 \* n\_bmi  
+ 0.6 \* n\_age  
+ 0.4 \* smoker  
+ 0.5 \* fam  
+ 0.4 \* n\_act  
- 0.2  
)  
htn = float(sigmoid(ht\_score))  
  
return {  
"diabetes\_risk": round(diabetes, 3),  
"heart\_disease\_risk": round(heart, 3),  
"hypertension\_risk": round(htn, 3),  
}  
  
def risk\_band(p):  
if p >= 0.67:  
return "High"  
if p >= 0.34:  
return "Moderate"  
return "Low"  
  
def make\_care\_plan(features, risks):  
bmi = features.get("bmi", np.nan)  
act = features.get("activity\_minutes\_per\_week", 0)  
sys\_bp = features.get("systolic\_bp", 120)  
fbg = features.get("fasting\_glucose", 95)  
hba1c = features.get("hba1c", 5.3)  
smoker = features.get("smoker", False)  
  
d\_band = risk\_band(risks["diabetes\_risk"])  
h\_band = risk\_band(risks["heart\_disease\_risk"])  
t\_band = risk\_band(risks["hypertension\_risk"])  
  
plan = []  
plan.append("⚠ This autogenerated plan is not medical advice. Consult a clinician.")  
  
if pd.notna(bmi):  
if bmi >= 30:  
plan.append("• Weight management: target 5–10% weight loss over 6–12 months.")  
elif bmi >= 25:  
plan.append("• Aim to reach BMI < 25 with gradual lifestyle changes.")  
else:  
plan.append("• Maintain current healthy weight with balanced nutrition.")  
  
if act < 150:  
plan.append("• Physical activity: build to ≥150 min/week moderate activity (+ 2 strength days).")  
else:  
plan.append("• Keep ≥150 min/week activity; add flexibility/strength work 2–3×/week.")  
  
if smoker:  
plan.append("• Smoking: begin a cessation plan (set quit date, NRT or meds per clinician).")  
  
if d\_band in ["Moderate", "High"] or fbg >= 100 or hba1c >= 5.7:  
plan.append("• Glucose management: reduce refined carbs/sugary drinks; prefer high-fiber foods.")  
plan.append("• Monitor fasting glucose weekly; discuss HbA1c testing cadence (e.g., q3–6 months).")  
  
if t\_band in ["Moderate", "High"] or sys\_bp >= 130:  
plan.append("• BP: limit sodium (~1.5–2 g/day), follow DASH-style diet, regular home BP logs.")  
plan.append("• Sleep hygiene: 7–9 h/night; screen for sleep apnea if snoring/daytime fatigue.")  
  
if h\_band in ["Moderate", "High"]:  
plan.append("• Lipids/heart: prioritize unsaturated fats, fish 2×/week, soluble fiber.")  
plan.append("• Add stress management: daily 10-min breathing/meditation; social support.")  
  
plan.append("• Schedule preventive care visit; share home logs (BP/weight/glucose) with clinician.")  
return "\n".join(plan)  
  
def to\_download\_bytes(dataframe):  
buf = io.StringIO()  
dataframe.to\_csv(buf, index=False)  
return buf.getvalue().encode()  
  
# ---------------------------  
# Gradio app functions  
# ---------------------------  
def predict\_single(  
age, sex, height\_cm, weight\_kg, systolic\_bp, diastolic\_bp,  
fasting\_glucose, hba1c, total\_chol, hdl, ldl,  
smoker, family\_history, activity\_minutes\_per\_week  
):  
bmi = compute\_bmi(height\_cm, weight\_kg)  
inputs = {  
"age": age,  
"sex": sex,  
"height\_cm": height\_cm,  
"weight\_kg": weight\_kg,  
"bmi": bmi,  
"systolic\_bp": systolic\_bp,  
"diastolic\_bp": diastolic\_bp,  
"fasting\_glucose": fasting\_glucose,  
"hba1c": hba1c,  
"total\_chol": total\_chol,  
"hdl": hdl,  
"ldl": ldl,  
"smoker": smoker,  
"family\_history": family\_history,  
"activity\_minutes\_per\_week": activity\_minutes\_per\_week,  
}  
risks = risk\_model(inputs)  
plan = make\_care\_plan(inputs, risks)  
  
# build display dataframe  
df = pd.DataFrame({  
"Condition": ["Diabetes", "Heart Disease", "Hypertension"],  
"Risk": [risks["diabetes\_risk"], risks["heart\_disease\_risk"], risks["hypertension\_risk"]],  
"Band": [risk\_band(risks["diabetes\_risk"]), risk\_band(risks["heart\_disease\_risk"]), risk\_band(risks["hypertension\_risk"])]  
})  
  
# bar figure  
fig = px.bar(df, x="Condition", y="Risk", color="Band", range\_y=[0,1], text=df["Risk"].map(lambda x: f"{x:.2f}"))  
fig.update\_layout(yaxis\_title="Risk (0–1)", xaxis\_title="")  
  
# create report bytes for download  
report = {  
"timestamp": datetime.utcnow().isoformat() + "Z",  
"inputs": inputs,  
"risks": risks,  
"plan": plan,  
}  
txt = (  
"Healthcare Assistant Report (Demo)\n"  
"=================================\n\n"  
f"Timestamp (UTC): {report['timestamp']}\n\n"  
"Inputs:\n"  
+ json.dumps(report["inputs"], indent=2)  
+ "\n\nRisks:\n"  
+ json.dumps(report["risks"], indent=2)  
+ "\n\nPlan:\n"  
+ report["plan"]  
+ "\n"  
).encode()  
  
return fig, df, plan, txt  
  
def dashboard\_from\_csv(file\_obj):  
# read csv into dataframe  
df = pd.read\_csv(file\_obj)  
sample\_cols = [  
"patient\_id","age","sex","height\_cm","weight\_kg","systolic\_bp","diastolic\_bp",  
"fasting\_glucose","hba1c","total\_chol","hdl","ldl","smoker","family\_history","activity\_minutes\_per\_week"  
]  
for col in sample\_cols:  
if col not in df.columns:  
df[col] = np.nan  
  
df["bmi"] = df.apply(lambda r: compute\_bmi(r.get("height\_cm", np.nan), r.get("weight\_kg", np.nan)), axis=1)  
  
for bcol in ["smoker", "family\_history"]:  
if bcol in df.columns:  
df[bcol] = df[bcol].map(lambda x: 1 if str(x).strip().lower() in ["1","true","yes","y"] else 0)  
  
def row\_risks(r):  
feats = {  
"age": r.get("age", np.nan),  
"sex": r.get("sex", "Other"),  
"height\_cm": r.get("height\_cm", np.nan),  
"weight\_kg": r.get("weight\_kg", np.nan),  
"bmi": r.get("bmi", np.nan),  
"systolic\_bp": r.get("systolic\_bp", np.nan),  
"diastolic\_bp": r.get("diastolic\_bp", np.nan),  
"fasting\_glucose": r.get("fasting\_glucose", np.nan),  
"hba1c": r.get("hba1c", np.nan),  
"total\_chol": r.get("total\_chol", np.nan),  
"hdl": r.get("hdl", np.nan),  
"ldl": r.get("ldl", np.nan),  
"smoker": bool(r.get("smoker", 0)),  
"family\_history": bool(r.get("family\_history", 0)),  
"activity\_minutes\_per\_week": r.get("activity\_minutes\_per\_week", np.nan),  
}  
return risk\_model(feats)  
  
risks\_list = df.apply(row\_risks, axis=1)  
df["diabetes\_risk"] = risks\_list.map(lambda d: d["diabetes\_risk"])  
df["heart\_disease\_risk"] = risks\_list.map(lambda d: d["heart\_disease\_risk"])  
df["hypertension\_risk"] = risks\_list.map(lambda d: d["hypertension\_risk"])  
  
df["diabetes\_band"] = df["diabetes\_risk"].map(risk\_band)  
df["heart\_band"] = df["heart\_disease\_risk"].map(risk\_band)  
df["hypertension\_band"] = df["hypertension\_risk"].map(risk\_band)  
  
# Overview metrics  
metrics = {  
"patients": len(df),  
"avg\_bmi": float(df["bmi"].mean()) if len(df) else float("nan"),  
"avg\_systolic": float(df["systolic\_bp"].mean()) if len(df) else float("nan"),  
"avg\_fasting\_glucose": float(df["fasting\_glucose"].mean()) if len(df) else float("nan"),  
}  
  
# Figures  
fig\_diab = px.histogram(df, x="diabetes\_risk", nbins=20, title="Diabetes Risk")  
fig\_heart = px.histogram(df, x="heart\_disease\_risk", nbins=20, title="Heart Disease Risk")  
fig\_htn = px.histogram(df, x="hypertension\_risk", nbins=20, title="Hypertension Risk")  
fig\_scatter = px.scatter(df, x="bmi", y="fasting\_glucose", color="diabetes\_band", title="BMI vs Fasting Glucose")  
  
csv\_bytes = to\_download\_bytes(df)  
return metrics, fig\_diab, fig\_heart, fig\_htn, fig\_scatter, df.head(50), csv\_bytes  
  
# ---------------------------  
# Build Gradio UI  
# ---------------------------  
with gr.Blocks(title="Healthcare Assistant (Demo)") as demo:  
gr.Markdown("## 🩺 Healthcare Assistant — Demo (not medical advice)")  
with gr.Row():  
with gr.Column(scale=3):  
gr.Markdown("### Patient Intake")  
age = gr.Number(value=35, label="Age", interactive=True)  
sex = gr.Dropdown(choices=["Female","Male","Other"], value="Female", label="Sex")  
height\_cm = gr.Number(value=170, label="Height (cm)")  
weight\_kg = gr.Number(value=70.0, label="Weight (kg)")  
systolic\_bp = gr.Number(value=122, label="Systolic BP")  
diastolic\_bp = gr.Number(value=80, label="Diastolic BP")  
fasting\_glucose = gr.Number(value=95, label="Fasting Glucose (mg/dL)")  
hba1c = gr.Number(value=5.3, label="HbA1c (%)")  
total\_chol = gr.Number(value=180, label="Total Cholesterol (mg/dL)")  
hdl = gr.Number(value=50, label="HDL (mg/dL)")  
ldl = gr.Number(value=110, label="LDL (mg/dL)")  
activity\_minutes\_per\_week = gr.Number(value=60, label="Activity (min/week)")  
smoker = gr.Checkbox(label="Currently smokes", value=False)  
family\_history = gr.Checkbox(label="Family history of cardiometabolic disease", value=False)  
predict\_btn = gr.Button("Predict Risks")  
  
with gr.Column(scale=4):  
gr.Markdown("### Predictions & Care Plan")  
out\_plot = gr.Plot(label="Risk Chart")  
out\_table = gr.Dataframe(value=pd.DataFrame(), label="Risk Table")  
out\_plan = gr.Textbox(lines=10, label="Autogenerated Care Plan")  
download\_report = gr.File(label="Download Report (.txt)")  
  
# CSV dashboard area  
gr.Markdown("## 📊 Population Dashboard (Upload CSV)")  
csv\_uploader = gr.File(label="Upload CSV (patients)", file\_count="single", file\_types=[".csv"])  
dashboard\_btn = gr.Button("Generate Dashboard")  
with gr.Row():  
col1 = gr.Column()  
col2 = gr.Column()  
metrics\_text = gr.JSON(value={})  
fig1 = gr.Plot()  
fig2 = gr.Plot()  
fig3 = gr.Plot()  
fig4 = gr.Plot()  
preview\_table = gr.Dataframe(value=pd.DataFrame(), label="Data Preview (first 50 rows)")  
download\_csv = gr.File(label="Download Enriched CSV")  
  
# Sample CSV download  
sample\_df = pd.DataFrame({  
"patient\_id": [f"P{i:03d}" for i in range(1, 31)],  
"age": np.random.randint(25, 75, 30),  
"sex": np.random.choice(["Female","Male","Other"], 30, p=[0.48,0.48,0.04]),  
"height\_cm": np.random.normal(168, 10, 30).round(0).clip(145, 195),  
"weight\_kg": np.random.normal(75, 15, 30).round(1).clip(45, 140),  
"systolic\_bp": np.random.normal(128, 16, 30).round(0).clip(90, 200),  
"diastolic\_bp": np.random.normal(82, 10, 30).round(0).clip(55, 120),  
"fasting\_glucose": np.random.normal(98, 15, 30).round(0).clip(65, 240),  
"hba1c": np.random.normal(5.6, 0.7, 30).round(1).clip(4.5, 11.0),  
"total\_chol": np.random.normal(190, 35, 30).round(0).clip(110, 320),  
"hdl": np.random.normal(52, 12, 30).round(0).clip(25, 90),  
"ldl": np.random.normal(120, 30, 30).round(0).clip(60, 220),  
"smoker": np.random.choice([0,1], 30, p=[0.75,0.25]),  
"family\_history": np.random.choice([0,1], 30, p=[0.7,0.3]),  
"activity\_minutes\_per\_week": np.random.normal(110, 70, 30).round(0).clip(0, 600),  
})  
sample\_df["bmi"] = sample\_df.apply(lambda r: compute\_bmi(r["height\_cm"], r["weight\_kg"]), axis=1)  
sample\_csv\_bytes = sample\_df.to\_csv(index=False).encode()  
sample\_download = gr.File(label="sample") # Initialize the sample\_download component  
  
# Bind events  
def on\_predict\_click(  
age\_v, sex\_v, height\_v, weight\_v, sys\_v, dia\_v, fbg\_v, a1c\_v, chol\_v, hdl\_v, ldl\_v, act\_v, smoker\_v, fam\_v  
):  
fig, df, plan\_text, txt\_bytes = predict\_single(  
age\_v, sex\_v, height\_v, weight\_v, sys\_v, dia\_v, fbg\_v, a1c\_v, chol\_v, hdl\_v, ldl\_v, smoker\_v, fam\_v, act\_v  
)  
# prepare downloadable txt file (gradio expects a file-like or path; we return bytes via BytesIO)  
report\_io = io.BytesIO(txt\_bytes)  
report\_io.name = "healthcare\_plan\_report.txt"  
return fig, df, plan\_text, report\_io  
  
predict\_btn.click(  
on\_predict\_click,  
inputs=[age, sex, height\_cm, weight\_kg, systolic\_bp, diastolic\_bp, fasting\_glucose, hba1c, total\_chol, hdl, ldl, activity\_minutes\_per\_week, smoker, family\_history],  
outputs=[out\_plot, out\_table, out\_plan, download\_report]  
)  
  
def on\_dashboard\_click(file\_obj):  
if file\_obj is None:  
return {"patients":0,"avg\_bmi":None,"avg\_systolic":None,"avg\_fasting\_glucose":None}, None, None, None, None, pd.DataFrame(), None  
metrics, f1, f2, f3, f4, head50, csv\_bytes = dashboard\_from\_csv(file\_obj.name if hasattr(file\_obj, "name") else file\_obj)  
csv\_io = io.BytesIO(csv\_bytes)  
csv\_io.name = "patients\_enriched.csv"  
return metrics, f1, f2, f3, f4, head50, csv\_io  
  
dashboard\_btn.click(  
on\_dashboard\_click,  
inputs=[csv\_uploader],  
outputs=[metrics\_text, fig1, fig2, fig3, fig4, preview\_table, download\_csv]  
)  
  
# Sample CSV download button (gradio way)  
# we provide a downloadable link by creating a downloadable BytesIO  
def get\_sample\_file():  
b = io.BytesIO(sample\_csv\_bytes)  
b.name = "sample\_patients.csv"  
return b  
  
sample\_dl\_btn = gr.Button("Download Sample CSV")  
sample\_dl\_btn.click(get\_sample\_file, inputs=None, outputs=[sample\_download]) # Use the component directly  
  
demo.launch(share=True)