# healthcare\_app.py

# ---------------------------------------------

# Streamlit Healthcare Assistant

# - Predicts risks for Diabetes, Heart Disease, Hypertension

# - Generates a simple care plan

# - Builds dashboards from uploaded CSV data

#

# DISCLAIMER: This tool is for educational use only and is NOT medical advice.

# Always consult licensed clinicians for diagnosis and treatment.

# ---------------------------------------------

import io

import json

import base64

import numpy as np

import pandas as pd

import plotly.express as px

import streamlit as st

from datetime import datetime

# ---------------------------

# Utility & risk calculations

# ---------------------------

def sigmoid(x):

return 1 / (1 + np.exp(-x))

def compute\_bmi(height\_cm: float, weight\_kg: float) -> float:

if height\_cm is None or weight\_kg is None or height\_cm <= 0:

return np.nan

return weight\_kg / ((height\_cm / 100) \*\* 2)

def clip\_num(x, lo, hi):

try:

return float(np.clip(float(x), lo, hi))

except Exception:

return np.nan

def normalize(x, mean, std, lo=None, hi=None):

"""Z-score then clip to [-3, 3] for stability."""

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):

"""

Simple transparent rule-based scoring mapped through sigmoid to produce

pseudo-probabilities. This is NOT a medical model.

Inputs (features):

age, sex, height\_cm, weight\_kg, bmi, systolic\_bp, diastolic\_bp,

fasting\_glucose, hba1c, total\_chol, hdl, ldl,

smoker(bool), family\_history(bool), activity\_minutes\_per\_week

Returns dict of risks in [0,1].

"""

# Extract features with defaults

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)

# Normalizations (rough population-ish priors)

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) # higher HDL reduces risk

n\_ldl = normalize(ldl, 120, 30)

n\_act = -normalize(act, 120, 60) # more activity reduces risk

# Diabetes risk score

# drivers: BMI, fasting glucose, HbA1c, age, activity, family history

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 # baseline

)

diabetes = float(sigmoid(d\_score))

# Heart disease risk score

# drivers: age, total chol, LDL, HDL (protective), BP, smoking, family history, activity

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 # already negative

)

heart = float(sigmoid(h\_score))

# Hypertension risk score

# drivers: systolic/diastolic BP, BMI, age, smoking, family history, activity

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)

# Key risk bands

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.")

# General

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).")

# Condition-specific

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.")

# Follow-up

plan.append("• Schedule preventive care visit; share home logs (BP/weight/glucose) with clinician.")

return "\n".join(plan)

def to\_download\_button(data: bytes, filename: str, label: str):

b64 = base64.b64encode(data).decode()

href = f'<a href="data:file/txt;base64,{b64}" download="{filename}">{label}</a>'

return href

# ---------------------------

# Streamlit UI

# ---------------------------

st.set\_page\_config(page\_title="Healthcare Assistant", page\_icon="🩺", layout="wide")

st.title("🩺 Healthcare Assistant (Demo)")

st.caption("Educational tool — not for medical use.")

with st.sidebar:

st.header("Patient Intake")

colA, colB = st.columns(2)

with colA:

age = st.number\_input("Age", min\_value=0, max\_value=120, value=35, step=1)

height\_cm = st.number\_input("Height (cm)", min\_value=80, max\_value=250, value=170)

systolic\_bp = st.number\_input("Systolic BP", min\_value=70, max\_value=250, value=122)

fasting\_glucose = st.number\_input("Fasting Glucose (mg/dL)", min\_value=50, max\_value=400, value=95)

total\_chol = st.number\_input("Total Cholesterol (mg/dL)", min\_value=80, max\_value=400, value=180)

hdl = st.number\_input("HDL (mg/dL)", min\_value=10, max\_value=120, value=50)

with colB:

sex = st.selectbox("Sex", ["Female", "Male", "Other"])

weight\_kg = st.number\_input("Weight (kg)", min\_value=20.0, max\_value=250.0, value=70.0, step=0.1)

diastolic\_bp = st.number\_input("Diastolic BP", min\_value=40, max\_value=150, value=80)

hba1c = st.number\_input("HbA1c (%)", min\_value=4.0, max\_value=15.0, value=5.3, step=0.1)

ldl = st.number\_input("LDL (mg/dL)", min\_value=30, max\_value=300, value=110)

activity\_minutes\_per\_week = st.number\_input("Activity (min/week)", min\_value=0, max\_value=2000, value=60, step=10)

smoker = st.checkbox("Currently smokes")

family\_history = st.checkbox("Family history of cardiometabolic disease")

calc\_bmi = compute\_bmi(height\_cm, weight\_kg)

st.markdown(f"\*\*BMI:\*\* {calc\_bmi:.1f}" if not pd.isna(calc\_bmi) else "\*\*BMI:\*\* —")

if st.button("Predict Risks"):

st.session\_state["inputs"] = {

"age": age,

"sex": sex,

"height\_cm": height\_cm,

"weight\_kg": weight\_kg,

"bmi": calc\_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,

}

st.session\_state["risks"] = risk\_model(st.session\_state["inputs"])

st.session\_state["plan"] = make\_care\_plan(st.session\_state["inputs"], st.session\_state["risks"])

# Main layout

tab\_pred, tab\_plan, tab\_dash, tab\_help = st.tabs(

["🧪 Predictions", "🗂 Care Plan", "📊 Dashboards", "ℹ️ Help & Template"]

)

with tab\_pred:

st.subheader("Predicted Risks")

if "risks" in st.session\_state:

risks = st.session\_state["risks"]

df\_risk = 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"]),

],

}

)

c1, c2 = st.columns([2, 1])

with c1:

fig = px.bar(

df\_risk, x="Condition", y="Risk", color="Band", range\_y=[0, 1],

text=df\_risk["Risk"].map(lambda x: f"{x:.2f}")

)

fig.update\_layout(yaxis\_title="Risk (0–1)", xaxis\_title="", uniformtext\_minsize=10, uniformtext\_mode="hide")

st.plotly\_chart(fig, use\_container\_width=True)

with c2:

st.dataframe(df\_risk, use\_container\_width=True)

st.success("Predictions generated. See the Care Plan tab for suggestions.")

else:

st.info("Use the sidebar to enter patient details and click \*\*Predict Risks\*\*.")

with tab\_plan:

st.subheader("Autogenerated Care Plan (Demo)")

if "plan" in st.session\_state and "inputs" in st.session\_state:

st.text(st.session\_state["plan"])

# Download plan with inputs and risks as a text file

report = {

"timestamp": datetime.utcnow().isoformat() + "Z",

"inputs": st.session\_state["inputs"],

"risks": st.session\_state["risks"],

"plan": st.session\_state["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()

st.download\_button(

"Download Plan & Report (.txt)",

data=txt,

file\_name="healthcare\_plan\_report.txt",

mime="text/plain",

)

else:

st.info("No plan yet. Generate predictions first.")

with tab\_dash:

st.subheader("Population Dashboard")

st.caption("Upload a CSV of patients to visualize risks and metrics. Columns are auto-detected.")

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"

]

uploaded = st.file\_uploader("Upload CSV", type=["csv"])

if uploaded is not None:

df = pd.read\_csv(uploaded)

# Ensure required columns exist; create if missing

for col in sample\_cols:

if col not in df.columns:

df[col] = np.nan

# Compute BMI

df["bmi"] = df.apply(lambda r: compute\_bmi(r["height\_cm"], r["weight\_kg"]), axis=1)

# Coarse type fixes

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)

# Risk predictions per row

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"])

# Banding

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)

st.markdown("### Overview")

k1, k2, k3, k4 = st.columns(4)

k1.metric("Patients", len(df))

k2.metric("Avg BMI", f"{df['bmi'].mean():.1f}")

k3.metric("Avg Systolic BP", f"{df['systolic\_bp'].mean():.0f}")

k4.metric("Avg Fasting Glucose", f"{df['fasting\_glucose'].mean():.0f}")

st.markdown("### Risk Distribution")

c1, c2 = st.columns(2)

with c1:

fig1 = px.histogram(df, x="diabetes\_risk", nbins=20, title="Diabetes Risk")

st.plotly\_chart(fig1, use\_container\_width=True)

fig2 = px.histogram(df, x="heart\_disease\_risk", nbins=20, title="Heart Disease Risk")

st.plotly\_chart(fig2, use\_container\_width=True)

with c2:

fig3 = px.histogram(df, x="hypertension\_risk", nbins=20, title="Hypertension Risk")

st.plotly\_chart(fig3, use\_container\_width=True)

band\_counts = df[["diabetes\_band","heart\_band","hypertension\_band"]].melt(var\_name="type", value\_name="band")

fig4 = px.bar(band\_counts.value\_counts().reset\_index().rename(columns={"count":"n"}),

x="type", y="n", color="band", title="Risk Bands Count")

st.plotly\_chart(fig4, use\_container\_width=True)

st.markdown("### Relationships")

r1, r2 = st.columns(2)

with r1:

fig5 = px.scatter(df, x="bmi", y="fasting\_glucose", color="diabetes\_band", title="BMI vs Fasting Glucose")

st.plotly\_chart(fig5, use\_container\_width=True)

with r2:

fig6 = px.scatter(df, x="systolic\_bp", y="diastolic\_bp", color="hypertension\_band", title="BP Scatter")

st.plotly\_chart(fig6, use\_container\_width=True)

st.markdown("### Data Preview")

st.dataframe(df.head(50), use\_container\_width=True)

# Download enriched dataset

out = io.StringIO()

df.to\_csv(out, index=False)

st.download\_button(

"Download Enriched CSV",

data=out.getvalue().encode(),

file\_name="patients\_enriched.csv",

mime="text/csv",

)

else:

st.info("Upload a CSV to see cohort dashboards.")

with tab\_help:

st.subheader("How to Use")

st.markdown(

"""

1) \*\*Enter patient details\*\* in the sidebar → click \*\*Predict Risks\*\*

2) See \*\*Predictions\*\* and \*\*Care Plan\*\*

3) Upload a CSV in \*\*Dashboards\*\* to visualize a population

\*\*CSV Template Columns\*\* (case-sensitive):

- `patient\_id` (str/int)

- `age` (int), `sex` (Female/Male/Other)

- `height\_cm`, `weight\_kg`

- `systolic\_bp`, `diastolic\_bp`

- `fasting\_glucose`, `hba1c`

- `total\_chol`, `hdl`, `ldl`

- `smoker` (1/0, yes/no/true/false)

- `family\_history` (1/0, yes/no/true/false)

- `activity\_minutes\_per\_week`

"""

)

# Provide a downloadable sample CSV

sample = 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["bmi"] = sample.apply(lambda r: compute\_bmi(r["height\_cm"], r["weight\_kg"]), axis=1)

csv\_bytes = sample.to\_csv(index=False).encode()

st.download\_button(

"Download Sample CSV",

data=csv\_bytes,

file\_name="sample\_patients.csv",

mime="text/csv",

)

# Footer disclaimer

st.write("---")

st.caption("© 2025 Demo. For education only. Not a medical device or diagnostic tool.")