```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
author: fran-pellegrino
"""
print("Deliverable 1: Perceptron Model")

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

from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import classification_report, roc_auc_score, roc_curve, confusion_matrix
from sklearn.utils.class_weight import compute_class_weight

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization, InputLayer
from tensorflow.keras.callbacks import EarlyStopping
import random
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error
from tensorflow.keras.optimizers import Adam
from math import sqrt


# ------------------------------------------------
# 1. Load Data
# ------------------------------------------------

DATA_PATH = "diabetes.csv"

COLS = [
    "diabetes", "high_bp", "high_chol", "bmi", "smoker",
    "stroke", "heart_attack", "phys_active", "fruit_daily", "veg_daily",
    "heavy_drinker", "has_healthcare", "not_afford_doctor", "general_health",
    "mental_health_days", "physical_health_days", "hard_climb_stairs",
    "sex", "age_bracket", "education_bracket", "income_bracket", "zodiac"
]

df = pd.read_csv(DATA_PATH, header=0, names=COLS)

# Drop zodiac
df = df.drop(columns=["zodiac"])

# ------------------------------------------------
# 2. Preprocessing
# ------------------------------------------------

target = "diabetes"

 #  B i n a r y  0 / 1  v a r i a b l e s  !’  p a s s t h r o u g h
binary_cols = [
    "high_bp","high_chol","smoker","stroke","heart_attack","phys_active",
    "fruit_daily","veg_daily","heavy_drinker","has_healthcare",
    "not_afford_doctor","hard_climb_stairs"
]

 #  C o n t i n u o u s  v a r i a b l e s  !’  i m p u t e  +  s c a l e
numeric_cols = ["bmi", "general_health", "mental_health_days", "physical_health_days"]

 #  C a t e g o r i c a l  v a r i a b l e s  !’  o n e - h o t  e n c o d e
categorical_cols = ["sex", "age_bracket", "education_bracket", "income_bracket"]

# Split X / y
X = df.drop(columns=[target])
y = df[target].astype(int).values

# ColumnTransformer with scaling
preprocess = ColumnTransformer(
    transformers=[
        ("num", Pipeline([
            ("imputer", SimpleImputer(strategy="median")),
            ("scaler", StandardScaler())
        ]), numeric_cols),

        ("cat", OneHotEncoder(drop="first", handle_unknown="ignore"), categorical_cols),
    ],
    remainder="passthrough"
)

# Train-test split
X_train_raw, X_test_raw, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=13639406, stratify=y
)

# Fit-transform preprocessing
X_train = preprocess.fit_transform(X_train_raw)
X_test = preprocess.transform(X_test_raw)

# Class weights for balancing
classes = np.unique(y_train)
cw = compute_class_weight(class_weight="balanced", classes=classes, y=y_train)
class_weight_dict = {c: w for c, w in zip(classes, cw)}

# ------------------------------------------------
# 3. Perceptron (SGDClassifier w/ perceptron loss)
# ------------------------------------------------

np.random.seed(13639406)

clf = SGDClassifier(
    loss="perceptron",
    penalty=None,
    learning_rate="constant",
    eta0=0.01,            # small LR for stability
    max_iter=5000,        # more epochs for convergence
    tol=1e-6,
    random_state=13639406,
    class_weight=class_weight_dict
)

clf.fit(X_train, y_train)

# Predictions
y_pred = clf.predict(X_test)
y_score = clf.decision_function(X_test)

# ------------------------------------------------
# 4. Metrics
# ------------------------------------------------

auc = roc_auc_score(y_test, y_score)

print("\n========== PERCEPTRON MODEL RESULTS ==========\n")
print(classification_report(y_test, y_pred, digits=4))
print("AUC =", round(auc, 4))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))

# ------------------------------------------------
# 5. FIGURE 1 — ROC Curve
# ------------------------------------------------

fpr, tpr, thresholds = roc_curve(y_test, y_score)

plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, linewidth=2, label=f"AUC = {auc:.3f}")
plt.plot([0,1],[0,1],'k--')

plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Perceptron ROC Curve")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

# ------------------------------------------------
# 6. FIGURE 2 — Feature Weight Force Plot
# ------------------------------------------------

# 1. Collect feature names
num_features = numeric_cols
cat_features = preprocess.named_transformers_['cat'].get_feature_names_out(categorical_cols)
passed_through = binary_cols  # come after num+cat

feature_names = list(num_features) + list(cat_features) + passed_through

coeffs = clf.coef_[0]
sorted_idx = np.argsort(coeffs)

plt.figure(figsize=(10,12))
sns.barplot(
    x=coeffs[sorted_idx],
    y=np.array(feature_names)[sorted_idx],
    palette="coolwarm"
)
plt.title("Perceptron Feature Influence (After Standard Scaling)")
plt.xlabel("Coefficient Value")
plt.ylabel("Feature")
plt.grid(True, axis='x')
plt.tight_layout()
plt.show()

print("\nPerceptron training completed.\n")
# -----------------------------------------------------------------------------------------------

print("\n\nDeliverable 2: FNN")

# reproducibility
SEED = 13639406
np.random.seed(SEED)
tf.random.set_seed(SEED)
random.seed(SEED)

# Ensure X_train, X_test, y_train, y_test exist (from previous perceptron code)
# X_train, X_test, y_train, y_test are assumed to be numpy arrays from ColumnTransformer

# Quick shape print
print("FFNN experiments — data shapes:", X_train.shape, X_test.shape, y_train.shape, y_test.shape)

# Architectures map: number of hidden layers -> list of units per hidden layer
arch_map = {
    0: [],              # no hidden layers -> logistic/linear output directly from inputs
    1: [64],            # one hidden layer
    2: [64, 32],        # two hidden layers
    3: [128, 64, 32]    # three hidden layers
}

hidden_layer_counts = [0, 1]
activations = ["relu", "sigmoid", "linear"]  # 'linear' == no activation in hidden layers

results = []  # store dicts: {'layers': n, 'activation': a, 'auc': x, 'model': model_ref(optional)'}

# Training hyperparams
BATCH_SIZE = 256
EPOCHS = 100
PATIENCE = 8
VERBOSE = 0  # set to 1 for training logs

for n_layers in hidden_layer_counts:
    units_list = arch_map[n_layers]
    for act in activations:
        tf.keras.backend.clear_session()
        # Build model
        model = Sequential()
        model.add(InputLayer(input_shape=(X_train.shape[1],)))
        # Add hidden layers if any
        for units in units_list:
            # If activation is 'linear' we pass activation=None (Dense defaults to linear)
            if act == "linear":
                model.add(Dense(units, activation=None))
            else:
                model.add(Dense(units, activation=act))
            # Add small batchnorm + dropout for stability (still linear if act='linear')
            model.add(BatchNormalization())
            model.add(Dropout(0.2))
        # Output layer (single unit, sigmoid)
        model.add(Dense(1, activation="sigmoid"))

        model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])
        es = EarlyStopping(monitor="val_loss", patience=PATIENCE, restore_best_weights=True, verbose
=0)

        # Fit
        history = model.fit(
            X_train, y_train,
            validation_split=0.1,
            epochs=EPOCHS,
            batch_size=BATCH_SIZE,
            class_weight=class_weight_dict,
            callbacks=[es],
            verbose=VERBOSE
        )

        # Predict probabilities on test set
        y_prob = model.predict(X_test).ravel()
        auc_val = roc_auc_score(y_test, y_prob)

        print(f"Layers={n_layers:>1d}, activation={act:6s} -> AUC = {auc_val:.4f}")

        results.append({
            "n_layers": n_layers,
            "activation": act,
            "auc": float(auc_val),
            "history": history,   # optional: keep if you want to inspect
            # "model": model      # avoid storing heavy model objects unless you want them
        })

# Convert results to DataFrame for plotting
res_df = pd.DataFrame(results)
print("\nSummary results:\n", res_df[["n_layers","activation","auc"]])

# ----------------------------
# FIGURE 1: AUC vs Number of Hidden Layers (one line per activation)
# ----------------------------
plt.figure(figsize=(8,6))
sns.lineplot(data=res_df, x="n_layers", y="auc", hue="activation", marker="o")
plt.xticks(hidden_layer_counts)
plt.xlabel("Number of hidden layers")
plt.ylabel("Test AUC")
plt.title("AUC vs Number of Hidden Layers (per activation)")
plt.grid(True)
plt.tight_layout()
plt.legend(title="activation")
plt.show()

# ----------------------------
# FIGURE 2: Grouped Bar Chart — Activation dependence per hidden-layer count
# ----------------------------
plt.figure(figsize=(9,6))
# pivot for grouped bars
pivot = res_df.pivot(index="n_layers", columns="activation", values="auc")
pivot.plot(kind="bar", figsize=(9,6))
plt.xlabel("Number of hidden layers")
plt.ylabel("Test AUC")
plt.title("AUC by activation function and number of hidden layers")
plt.xticks(rotation=0)
plt.ylim(0.0, 1.0)
plt.grid(axis='y')
plt.legend(title="activation")
plt.tight_layout()
plt.show()

# ----------------------------
# Additional creative visual: Heatmap of AUC (activation x layers)
# ----------------------------
plt.figure(figsize=(6,4))
heat = pivot.T  # activation rows, n_layers columns
sns.heatmap(heat, annot=True, fmt=".4f", cmap="viridis")
plt.xlabel("Number of hidden layers")
plt.title("Heatmap: AUC (activation × layers)")
plt.tight_layout()
plt.show()

# ----------------------------
# Print final ordered table
# ----------------------------
print("\nFinal ranked results (by AUC desc):")
print(res_df.sort_values("auc", ascending=False).reset_index(drop=True)[["n_layers","activation","au
c"]])

print("\n FNN training completed.\n")


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

print("\n\nDeliverable 3: DNN")

# ----------------------------
# 1. Reproducibility
# ----------------------------
np.random.seed(13639406)
random.seed(13639406)
tf.random.set_seed(13639406)

# ----------------------------
# 2. Deep Network Architecture
# ----------------------------
deep_model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    BatchNormalization(),
    Dense(32, activation='relu'),
    BatchNormalization(),
    Dense(1, activation='sigmoid')  # output layer
])

deep_model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=[tf.keras.metrics.AUC(name='auc')]
)

# ----------------------------
# 3. Early stopping
# ----------------------------
early_stop = EarlyStopping(
    monitor='val_auc',
    mode='max',
    patience=10,
    restore_best_weights=True,
    verbose=1
)

# ----------------------------
# 4. Train-test split (reuse previous)
# ----------------------------
history = deep_model.fit(
    X_train, y_train,
    validation_split=0.2,
    epochs=100,
    batch_size=256,
    callbacks=[early_stop],
    verbose=2
)

# ----------------------------
# 5. Evaluate model
# ----------------------------
y_pred_prob = deep_model.predict(X_test).ravel()
y_pred = (y_pred_prob >= 0.5).astype(int)
auc = roc_auc_score(y_test, y_pred_prob)

print("\n========== DEEP NEURAL NETWORK RESULTS ==========\n")
print(classification_report(y_test, y_pred, digits=4))
print("AUC:", round(auc, 4))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

# ----------------------------
# 6. FIGURE 1 — ROC Curve with threshold sweep
# ----------------------------
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)

plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, label=f'AUC = {auc:.3f}', linewidth=2)
plt.plot([0,1], [0,1], 'k--')

# Annotate a few threshold points
for thr in [0.1, 0.25, 0.5, 0.75, 0.9]:
    idx = np.argmin(np.abs(thresholds - thr))
    plt.scatter(fpr[idx], tpr[idx], s=70)
    plt.text(fpr[idx]+0.02, tpr[idx]-0.02, f"thr={thr:.2f}")

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Deep Network ROC Curve with Threshold Annotations')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# ----------------------------
# 7. FIGURE 2 — Feature Importance via Permutation
# ----------------------------

# Since deep_model isn't an sklearn estimator, let's approximate feature influence
# using correlation with predictions as a proxy
feature_corr = np.array([np.corrcoef(X_test[:,i], y_pred_prob)[0,1] for i in range(X_test.shape[1])]
)

plt.figure(figsize=(10,12))
sns.barplot(
    x=feature_corr,
    y=feature_names,
    palette="vlag"
)
plt.title("Deep Network Feature Influence (Correlation with Predictions)")
plt.xlabel("Correlation with Model Output")
plt.ylabel("Feature")
plt.grid(True, axis='x')
plt.tight_layout()
plt.show()

print("\nDeep Network complete.\n")

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

print("Deliverable 4: FNN for BMI Prediction")

# ----------------------------
# Preprocessing for BMI prediction
# ----------------------------
target = "bmi"

# Identify features
binary_cols = [
    "high_bp","high_chol","smoker","stroke","heart_attack","phys_active",
    "fruit_daily","veg_daily","heavy_drinker","has_healthcare",
    "not_afford_doctor","hard_climb_stairs"
]

numeric_cols = ["general_health", "mental_health_days", "physical_health_days"]
categorical_cols = ["sex", "age_bracket", "education_bracket", "income_bracket"]

# Split X, y
X = df.drop(columns=[target])
y = df[target].values

# Train-test split
X_train_raw, X_test_raw, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=13639406
)

# Redefine preprocessing pipeline
preprocess_bmi = ColumnTransformer(
    transformers=[
        ("num", SimpleImputer(strategy="median"), numeric_cols),
        ("cat", OneHotEncoder(drop="first", handle_unknown="ignore"), categorical_cols),
    ],
    remainder="passthrough"  # keep binary columns
)

X_train = preprocess_bmi.fit_transform(X_train_raw)
X_test = preprocess_bmi.transform(X_test_raw)

# ----------------------------
# 3. Neural Network Design
# ----------------------------
activations = ["relu", "sigmoid", "linear"]
rmse_results = {}

for act in activations:
    np.random.seed(13639406)
    tf.random.set_seed(13639406)

    nn_model = Sequential([
        Dense(64, activation=act, input_shape=(X_train.shape[1],)),
        Dense(32, activation=act),
        Dense(1, activation='linear')  # output layer for regression
    ])

    nn_model.compile(
        optimizer='adam',
        loss='mse',
        metrics=[]
    )

    history = nn_model.fit(
        X_train, y_train,
        validation_split=0.2,
        epochs=100,
        batch_size=256,
        verbose=0,
        callbacks=[EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)]
    )

    y_pred_scaled = nn_model.predict(X_test).ravel()
    rmse = sqrt(mean_squared_error(y_test, y_pred_scaled))
    rmse_results[act] = rmse

    if act == "relu":  # keep one example network for figure generation
        best_model = nn_model
        best_history = history

print("\nNeural network used for BMI prediction: Feedforward Neural Network with 2 hidden layers")
print("RMSE by activation function:")
for act, rmse in rmse_results.items():
    print(f"  {act}: {rmse:.4f}")

# ----------------------------
# 4. FIGURE 1 — RMSE vs Activation Function
# ----------------------------
plt.figure(figsize=(6,4))
sns.barplot(x=list(rmse_results.keys()), y=list(rmse_results.values()), palette="viridis")
plt.ylabel("RMSE (scaled BMI)")
plt.xlabel("Activation Function")
plt.title("Effect of Activation Function on BMI Prediction RMSE")
plt.grid(True, axis='y')
plt.tight_layout()
plt.show()

# ----------------------------
# 5. FIGURE 2 — RMSE vs Epochs (Learning Curve)
# ----------------------------
plt.figure(figsize=(8,6))
plt.plot(best_history.history['loss'], label='Training Loss', linewidth=2)
plt.plot(best_history.history['val_loss'], label='Validation Loss', linewidth=2)
plt.xlabel("Epoch")
plt.ylabel("MSE Loss (scaled)")
plt.title(f"Training vs Validation Loss Curve (Activation={act})")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```