```
# ==========================================
# Early Stage Diabetes Risk Prediction Project
# ANN + Baselines (LR, SVM, RF, KNN) with CV, val-threshold tuning, calibration
# ==========================================

import os
import json
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from typing import Dict

# Reproducibility
RANDOM_STATE = 42
random.seed(RANDOM_STATE)
np.random.seed(RANDOM_STATE)

# Sklearn
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_validate
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score, roc_auc_score,
    classification_report, confusion_matrix, RocCurveDisplay, PrecisionRecallDisplay
)
from sklearn.calibration import calibration_curve, CalibratedClassifierCV
from sklearn.isotonic import IsotonicRegression
from sklearn.inspection import permutation_importance

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier

# TensorFlow / Keras
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.callbacks import EarlyStopping
```

```
# ----------------------------
# Config
# ----------------------------
CSV_PATH = '/content/dataset.csv'  # set to path
RESULTS_DIR = 'results_diabetes'
os.makedirs(RESULTS_DIR, exist_ok=True)
N_SPLITS = 5
```

```
# ----------------------------
# Load & Preprocess
# ----------------------------
df = pd.read_csv(CSV_PATH)
df.columns = [c.strip() for c in df.columns]

yes_no_cols = [
    'Polyuria','Polydipsia','sudden weight loss','weakness','Polyphagia',
    'Genital thrush','visual blurring','Itching','Irritability','delayed healing',
    'partial paresis','muscle stiffness','Alopecia','Obesity'
]

# Map Yes/No → 1/0 (robust to whitespace/case)
for c in yes_no_cols:
    df[c] = (df[c].astype(str).str.strip().str.title()
             .map({'Yes': 1, 'No': 0}))

df['Gender'] = (df['Gender'].astype(str).str.strip().str.title()
                .map({'Male': 1, 'Female': 0}))
```

```
df['class']  = (df['class'].astype(str).str.strip().str.title()
                .map({'Positive': 1, 'Negative': 0}))

# Sanity check
missing = df[yes_no_cols + ['Gender','class']].isna().sum().sum()
if missing != 0:
    raise ValueError(f"Found {missing} missing values after mapping—clean the data before proceeding.")

X = df.drop(columns=['class'])
y = df['class']

num_cols = ['Age']
bin_cols = [c for c in X.columns if c not in num_cols]

# Train/Val/Test split
X_train_full, X_test, y_train_full, y_test = train_test_split(
    X, y, test_size=0.20, stratify=y, random_state=RANDOM_STATE
)
X_train, X_val, y_train, y_val = train_test_split(
    X_train_full, y_train_full, test_size=0.20, stratify=y_train_full, random_state=RANDOM_STATE
)

# ColumnTransformer: scale Age only, passthrough binaries
ct = ColumnTransformer(
    [('num', StandardScaler(), num_cols), ('bin', 'passthrough', bin_cols)],
    remainder='drop'
)
```

```
# ----------------------------
# Helpers
# ----------------------------
def evaluate_model(name: str, y_true, y_pred, y_score) -> Dict:
    """Return metrics dict; y_score is prob or decision function for ROC-AUC."""
    return {
        'Model': name,
        'Accuracy': accuracy_score(y_true, y_pred),
        'Precision': precision_score(y_true, y_pred),
        'Recall': recall_score(y_true, y_pred),
        'F1': f1_score(y_true, y_pred),
        'ROC_AUC': roc_auc_score(y_true, y_score)
    }

def select_threshold_with_constraints(
    y_true: np.ndarray,
    y_score: np.ndarray,
    target_recall: float = 0.98,
    min_precision: float = 0.95
) -> float:
    """
    Choose threshold on validation predictions to achieve recall >= target_recall
    and precision >= min_precision. If no candidate exists, fall back to max F1.
    """
    from sklearn.metrics import precision_recall_curve
    prec, rec, thr = precision_recall_curve(y_true, y_score)
    candidates = [(p, r, t, (2*p*r)/(p+r+1e-12))
                  for p, r, t in zip(prec, rec, thr)
                  if (r >= target_recall and p >= min_precision)]
    if candidates:
        best = max(candidates, key=lambda x: x[3])  # best F1 among candidates
        return best[2]
    # fallback to max F1
    f1 = 2*prec*rec/(prec+rec+1e-12)
    idx = np.argmax(f1)
    return thr[idx] if idx < len(thr) else 0.5

def plot_ann_diagnostics(y_true, y_score, y_pred, out_dir: str):
    # ROC
    fig_roc, ax_roc = plt.subplots(figsize=(6,5))
    RocCurveDisplay.from_predictions(y_true, y_score, ax=ax_roc)
    ax_roc.set_title('ANN ROC Curve (Test)')
    fig_roc.tight_layout()
    fig_roc.savefig(os.path.join(out_dir, 'ann_roc.png'), dpi=200)
    plt.close(fig_roc)
```

```
    # Precision-Recall
    fig_pr, ax_pr = plt.subplots(figsize=(6,5))
    PrecisionRecallDisplay.from_predictions(y_true, y_score, ax=ax_pr)
    ax_pr.set_title('ANN Precision-Recall Curve (Test)')
    fig_pr.tight_layout()
    fig_pr.savefig(os.path.join(out_dir, 'ann_pr.png'), dpi=200)
    plt.close(fig_pr)

    # Calibration / Reliability
    prob_true, prob_pred = calibration_curve(y_true, y_score, n_bins=10, strategy='uniform')
    fig_cal, ax_cal = plt.subplots(figsize=(6,5))
    ax_cal.plot(prob_pred, prob_true, marker='o')
    ax_cal.plot([0,1],[0,1], '--', color='gray')
    ax_cal.set_xlabel('Predicted probability (mean per bin)')
    ax_cal.set_ylabel('Observed frequency')
    ax_cal.set_title('ANN Reliability Diagram (Test)')
    fig_cal.tight_layout()
    fig_cal.savefig(os.path.join(out_dir, 'ann_calibration.png'), dpi=200)
    plt.close(fig_cal)

    # Confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    fig_cm, ax_cm = plt.subplots(figsize=(5,4))
    im = ax_cm.imshow(cm, cmap='Blues')
    ax_cm.set_xticks([0,1]); ax_cm.set_yticks([0,1])
    ax_cm.set_xticklabels(['Negative','Positive'])
    ax_cm.set_yticklabels(['Negative','Positive'])
    for (i,j), val in np.ndenumerate(cm):
        ax_cm.text(j, i, f'{val}', ha='center', va='center', color='black')
    ax_cm.set_title('ANN Confusion Matrix (Test)')
    ax_cm.set_xlabel('Predicted'); ax_cm.set_ylabel('True')
    fig_cm.colorbar(im, ax=ax_cm)
    fig_cm.tight_layout()
    fig_cm.savefig(os.path.join(out_dir, 'ann_confusion_matrix.png'), dpi=200)
    plt.close(fig_cm)
```

```
# ------------------------------
# Baselines: 5-fold CV (mean ± std)
# For CV: use SVC(probability=True) to avoid nested CV inside calibration.
# ------------------------------
baselines_cv = []

pipe_lr = Pipeline([('prep', ct), ('clf', LogisticRegression(max_iter=1000, random_state=RANDOM_STATE))])
pipe_svm_cv = Pipeline([('prep', ct), ('clf', SVC(kernel='rbf', probability=True, random_state=RANDOM_STATE))])
pipe_rf = Pipeline([('prep', ct), ('clf', RandomForestClassifier(n_estimators=300, random_state=RANDOM_STATE))])
pipe_knn = Pipeline([('prep', ct), ('clf', KNeighborsClassifier(n_neighbors=5))])

def cv_scores(name, estimator, X, y):
    scoring = ['accuracy','precision','recall','f1','roc_auc']
    skf = StratifiedKFold(n_splits=N_SPLITS, shuffle=True, random_state=RANDOM_STATE)
    out = cross_validate(estimator, X, y, scoring=scoring, cv=skf, n_jobs=-1, return_train_score=False)
    return {
        'Model': name,
        **{f'{m}_mean': float(np.mean(out[f'test_{m}'])) for m in scoring},
        **{f'{m}_std' : float(np.std(out[f'test_{m}']))  for m in scoring}
    }

baselines_cv.append(cv_scores("Logistic Regression", pipe_lr, X, y))
baselines_cv.append(cv_scores("SVM (RBF)",           pipe_svm_cv, X, y))
baselines_cv.append(cv_scores("Random Forest",       pipe_rf, X, y))
baselines_cv.append(cv_scores("KNN (k=5)",           pipe_knn, X, y))

cv_df = pd.DataFrame(baselines_cv)
cv_df.to_excel(os.path.join(RESULTS_DIR, 'baselines_cv_results.xlsx'), index=False)
```

```
# ------------------------------
# Fit baselines on train_full and evaluate on test (one-shot)
# SVM on test: calibrated probabilities (isotonic).
# ------------------------------
def fit_and_eval_on_test(name, estimator):
    estimator.fit(X_train_full, y_train_full)
    # proba or decision score
```

```
    if hasattr(estimator, 'predict_proba'):
        y_score = estimator.predict_proba(X_test)[:, 1]
    elif hasattr(estimator, 'decision_function'):
        y_score = estimator.decision_function(X_test)
    else:
        raise ValueError(f"{name} does not support probability/decision scores.")
    y_pred = estimator.predict(X_test)
    return evaluate_model(name, y_test, y_pred, y_score)

test_results_map = {}
test_results_map["Logistic Regression"] = fit_and_eval_on_test("Logistic Regression", pipe_lr)
test_results_map["Random Forest"]       = fit_and_eval_on_test("Random Forest", pipe_rf)
test_results_map["KNN (k=5)"]           = fit_and_eval_on_test("KNN (k=5)", pipe_knn)

# SVM (calibrated) — fit separately and evaluate
svm_base = Pipeline([('prep', ct), ('clf', SVC(kernel='rbf', probability=False, random_state=RANDOM_STATE))])
svm_cal  = CalibratedClassifierCV(svm_base, cv=N_SPLITS, method='isotonic')
svm_cal.fit(X_train_full, y_train_full)
y_score_svm = svm_cal.predict_proba(X_test)[:,1]
y_pred_svm  = svm_cal.predict(X_test)
test_results_map["SVM (RBF, calibrated)"] = evaluate_model("SVM (RBF, calibrated)", y_test, y_pred_svm, y_score_svm)
```

```
# ----------------------------
# ANN: CV (5-fold) for robustness
# ----------------------------
def ann_build(input_dim: int) -> tf.keras.Model:
    tf.random.set_seed(RANDOM_STATE)
    model = models.Sequential([
        layers.Input(shape=(input_dim,)),
        layers.Dense(32, activation='relu'),
        layers.Dropout(0.30),
        layers.Dense(16, activation='relu'),
        layers.Dropout(0.20),
        layers.Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy', tf.keras.metrics.AUC(name='auc')])
    return model

def ann_cv_scores(X: pd.DataFrame, y: pd.Series, n_splits=5) -> Dict:
    skf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=RANDOM_STATE)
    metrics = {'accuracy':[], 'precision':[], 'recall':[], 'f1':[], 'roc_auc':[]}
    for train_idx, test_idx in skf.split(X, y):
        X_tr, X_te = X.iloc[train_idx], X.iloc[test_idx]
        y_tr, y_te = y.iloc[train_idx], y.iloc[test_idx]

        # Fit ColumnTransformer on training fold
        ct_fold = ColumnTransformer(
            [('num', StandardScaler(), num_cols), ('bin', 'passthrough', bin_cols)],
            remainder='drop'
        )
        X_tr_s = ct_fold.fit_transform(X_tr)
        X_te_s = ct_fold.transform(X_te)

        # Internal validation split for early-stopping
        X_tr_s2, X_val_s2, y_tr2, y_val2 = train_test_split(
            X_tr_s, y_tr, test_size=0.2, stratify=y_tr, random_state=RANDOM_STATE
        )

        model = ann_build(X_tr_s.shape[1])
        early = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
        model.fit(X_tr_s2, y_tr2, validation_data=(X_val_s2, y_val2),
                  epochs=100, batch_size=32, callbacks=[early], verbose=0)

        # Threshold on validation = max F1 (fold-wise)
        y_val_score = model.predict(X_val_s2).ravel()
        from sklearn.metrics import precision_recall_curve
        prec, rec, thr = precision_recall_curve(y_val2, y_val_score)
        f1 = 2*prec*rec/(prec+rec+1e-12)
        idx = np.argmax(f1)
        thr_best = thr[idx] if idx < len(thr) else 0.5

        # Evaluate on fold test
```

```
        y_te_score = model.predict(X_te_s).ravel()
        y_te_pred  = (y_te_score >= thr_best).astype(int)

        metrics['accuracy'].append(accuracy_score(y_te, y_te_pred))
        metrics['precision'].append(precision_score(y_te, y_te_pred))
        metrics['recall'].append(recall_score(y_te, y_te_pred))
        metrics['f1'].append(f1_score(y_te, y_te_pred))
        metrics['roc_auc'].append(roc_auc_score(y_te, y_te_score))

    return {
        'Model': 'ANN (CV)',
        'accuracy_mean': float(np.mean(metrics['accuracy'])),
        'accuracy_std' : float(np.std(metrics['accuracy'])),
        'precision_mean': float(np.mean(metrics['precision'])),
        'precision_std' : float(np.std(metrics['precision'])),
        'recall_mean': float(np.mean(metrics['recall'])),
        'recall_std' : float(np.std(metrics['recall'])),
        'f1_mean': float(np.mean(metrics['f1'])),
        'f1_std' : float(np.std(metrics['f1'])),
        'roc_auc_mean': float(np.mean(metrics['roc_auc'])),
        'roc_auc_std' : float(np.std(metrics['roc_auc']))
    }

# Compute ANN CV metrics and append to CV table
ann_cv = ann_cv_scores(X, y, n_splits=N_SPLITS)
cv_df = pd.concat([cv_df, pd.DataFrame([ann_cv])], ignore_index=True)
cv_df.to_excel(os.path.join(RESULTS_DIR, 'baselines_plus_ann_cv_results.xlsx'), index=False)
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/3 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/stepWARNING:tensorflow:5 out of the last 15 calls to <function TensorFlowTrainer.make_predict_f
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/3 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/stepWARNING:tensorflow:5 out of the last 15 calls to <function TensorFlowTrainer.make_predict_f
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
```

```
# ANN: Final training on Train with Validation
# - Isotonic calibration of probabilities
# - Threshold selection with clinical constraints
# ------------------------------
# Fit transformer on training (not on validation/test)
ct_ann = ColumnTransformer(
    [('num', StandardScaler(), num_cols), ('bin', 'passthrough', bin_cols)],
    remainder='drop'
)
X_train_s = ct_ann.fit_transform(X_train)
X_val_s   = ct_ann.transform(X_val)
X_test_s  = ct_ann.transform(X_test)

model_ann = ann_build(X_train_s.shape[1])
early = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
history = model_ann.fit(X_train_s, y_train, validation_data=(X_val_s, y_val),
                        epochs=100, batch_size=32, callbacks=[early], verbose=1)

# Raw validation probabilities
y_val_score_raw = model_ann.predict(X_val_s).ravel()

# Optional: Calibrate ANN probabilities via Isotonic Regression
iso = IsotonicRegression(out_of_bounds='clip')
iso.fit(y_val_score_raw, y_val.astype(int))
y_val_score_cal = iso.predict(y_val_score_raw)

# Choose threshold on calibrated validation scores with constraints
best_thr = select_threshold_with_constraints(
    y_true=y_val,
    y_score=y_val_score_cal,
    target_recall=0.98,
    min_precision=0.95
)
```

```
# Test evaluation using calibrated probabilities and chosen threshold
y_test_score_raw = model_ann.predict(X_test_s).ravel()
y_test_score_cal = iso.predict(y_test_score_raw)  # calibrated
y_test_pred      = (y_test_score_cal >= best_thr).astype(int)

ann_metrics = evaluate_model("ANN (thr tuned on val; calibrated)", y_test, y_test_pred, y_test_score_cal)
ann_report  = classification_report(y_test, y_test_pred, output_dict=True)
ann_cm      = confusion_matrix(y_test, y_test_pred)

# Plots
plot_ann_diagnostics(y_test, y_test_score_cal, y_test_pred, out_dir=RESULTS_DIR)
```

```
Epoch 73/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9835 - auc: 0.9964 - loss: 0.0674 - val_accuracy: 0.9286 - val_auc: 0.97
Epoch 74/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9844 - auc: 0.9973 - loss: 0.0657 - val_accuracy: 0.9286 - val_auc: 0.97
Epoch 75/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9743 - auc: 0.9973 - loss: 0.0749 - val_accuracy: 0.9286 - val_auc: 0.96
Epoch 76/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9704 - auc: 0.9968 - loss: 0.0714 - val_accuracy: 0.9286 - val_auc: 0.96
Epoch 77/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9801 - auc: 0.9980 - loss: 0.0596 - val_accuracy: 0.9286 - val_auc: 0.96
Epoch 78/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9684 - auc: 0.9970 - loss: 0.0700 - val_accuracy: 0.9286 - val_auc: 0.96
Epoch 79/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9776 - auc: 0.9959 - loss: 0.0721 - val_accuracy: 0.9286 - val_auc: 0.96
Epoch 80/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9916 - auc: 0.9978 - loss: 0.0574 - val_accuracy: 0.9286 - val_auc: 0.97
Epoch 81/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9764 - auc: 0.9973 - loss: 0.0698 - val_accuracy: 0.9286 - val_auc: 0.97
Epoch 82/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9854 - auc: 0.9979 - loss: 0.0635 - val_accuracy: 0.9286 - val_auc: 0.97
Epoch 83/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9915 - auc: 0.9985 - loss: 0.0605 - val_accuracy: 0.9286 - val_auc: 0.97
Epoch 84/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9855 - auc: 0.9980 - loss: 0.0615 - val_accuracy: 0.9286 - val_auc: 0.97
Epoch 85/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9886 - auc: 0.9973 - loss: 0.0561 - val_accuracy: 0.9286 - val_auc: 0.97
Epoch 86/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9900 - auc: 0.9993 - loss: 0.0481 - val_accuracy: 0.9286 - val_auc: 0.97
Epoch 87/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9769 - auc: 0.9987 - loss: 0.0576 - val_accuracy: 0.9286 - val_auc: 0.97
Epoch 88/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9849 - auc: 0.9977 - loss: 0.0651 - val_accuracy: 0.9286 - val_auc: 0.97
Epoch 89/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9922 - auc: 0.9992 - loss: 0.0494 - val_accuracy: 0.9286 - val_auc: 0.97
Epoch 90/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9858 - auc: 0.9995 - loss: 0.0453 - val_accuracy: 0.9286 - val_auc: 0.97
Epoch 91/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9872 - auc: 0.9989 - loss: 0.0517 - val_accuracy: 0.9286 - val_auc: 0.97
Epoch 92/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9911 - auc: 0.9987 - loss: 0.0540 - val_accuracy: 0.9286 - val_auc: 0.97
Epoch 93/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9809 - auc: 0.9982 - loss: 0.0575 - val_accuracy: 0.9286 - val_auc: 0.97
Epoch 94/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9865 - auc: 0.9986 - loss: 0.0499 - val_accuracy: 0.9286 - val_auc: 0.97
Epoch 95/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9813 - auc: 0.9992 - loss: 0.0462 - val_accuracy: 0.9286 - val_auc: 0.97
Epoch 96/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9828 - auc: 0.9983 - loss: 0.0496 - val_accuracy: 0.9286 - val_auc: 0.97
Epoch 97/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9883 - auc: 0.9989 - loss: 0.0458 - val_accuracy: 0.9286 - val_auc: 0.97
Epoch 98/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9908 - auc: 0.9992 - loss: 0.0413 - val_accuracy: 0.9286 - val_auc: 0.97
Epoch 99/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9796 - auc: 0.9991 - loss: 0.0555 - val_accuracy: 0.9286 - val_auc: 0.97
Epoch 100/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9837 - auc: 0.9987 - loss: 0.0529 - val_accuracy: 0.9286 - val_auc: 0.97
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
```

```
# Feature importance (RF) for interpretability
# -----------------------------
pipe_rf.fit(X_train_full, y_train_full)
perm = permutation_importance(pipe_rf, X_test, y_test, n_repeats=10, random_state=RANDOM_STATE, n_jobs=-1)
fi = pd.DataFrame({
    'feature': num_cols + bin_cols,
    'mean_importance': perm.importances_mean,
    'std_importance': perm.importances_std
}).sort_values(by='mean_importance', ascending=False)
```

```
fi.to_excel(os.path.join(RESULTS_DIR, 'rf_permutation_importance.xlsx'), index=False)

# Top 10 (Quick Plot)
top_n = 10
fig_fi, ax_fi = plt.subplots(figsize=(8,5))
fi.head(top_n).plot(kind='barh', x='feature', y='mean_importance',
                    xerr=fi.head(top_n)['std_importance'], ax=ax_fi, color='teal')
ax_fi.invert_yaxis()
ax_fi.set_title('Random Forest Permutation Importance (Top 10)')
ax_fi.set_xlabel('Mean decrease in score')
fig_fi.tight_layout()
fig_fi.savefig(os.path.join(RESULTS_DIR, 'rf_permutation_importance.png'), dpi=200)
plt.close(fig_fi)
```

```
# ------------------------------
# Save consolidated outputs
# ------------------------------
# CV results
print("\n=== Baselines + ANN (5-fold CV) mean±std ===")
print(cv_df[['Model','accuracy_mean','accuracy_std','precision_mean','precision_std',
             'recall_mean','recall_std','f1_mean','f1_std','roc_auc_mean','roc_auc_std']])
cv_df.to_excel(os.path.join(RESULTS_DIR, 'cv_summary.xlsx'), index=False)

# Test-set table (one-shot fits) with idempotent dict
test_results_map["SVM (RBF, calibrated)"] = test_results_map.pop("SVM (RBF, calibrated)") if "SVM (RBF, calibrated)" in test_resu
test_results_map["ANN (thr tuned on val; calibrated)"] = ann_metrics
test_df = pd.DataFrame(test_results_map.values()).sort_values(by='ROC_AUC', ascending=False)
test_df['Notes'] = ''
test_df.loc[test_df['Model'] == 'ANN (thr tuned on val; calibrated)',
            'Notes'] = f'Calibrated (Isotonic); Threshold={best_thr:.4f} chosen on validation with recall≥0.98 & precision≥0.95'

print("\n=== Test-set comparison (one-shot fits) ===")
print(test_df)

test_df.to_excel(os.path.join(RESULTS_DIR, 'test_set_comparison.xlsx'), index=False)

# ANN details JSON
ann_detail = {
    'ann_metrics_test': ann_metrics,
    'ann_threshold_validation': float(best_thr),
    'ann_classification_report_test': ann_report,
    'ann_confusion_matrix_test': ann_cm.tolist()
}
with open(os.path.join(RESULTS_DIR, 'ann_details.json'), 'w') as f:
    json.dump(ann_detail, f, indent=2)

# Class counts for the report
print("\n=== Class counts ===")
print(df['class'].value_counts())
```

```
=== Baselines + ANN (5-fold CV) mean±std ===
                Model  accuracy_mean  accuracy_std  precision_mean  \
0  Logistic Regression       0.928846      0.011538        0.947398
1            SVM (RBF)       0.967308      0.007692        0.978064
2        Random Forest       0.982692      0.012756        0.987589
3            KNN (k=5)       0.913462      0.019231        0.969878
4             ANN (CV)       0.953846      0.016543        0.954437

   precision_std  recall_mean  recall_std   f1_mean    f1_std  roc_auc_mean  \
0       0.024740     0.937500    0.019764  0.941974  0.008981      0.976562
1       0.007215     0.968750    0.017116  0.973249  0.006626      0.997266
2       0.011422     0.984375    0.017116  0.985874  0.010516      0.998750
3       0.015251     0.887500    0.037500  0.926219  0.018024      0.978516
4       0.017853     0.971875    0.025000  0.962778  0.013500      0.989531

   roc_auc_std
0     0.009188
1     0.001694
2     0.001109
3     0.008328
4     0.009242

=== Test-set comparison (one-shot fits) ===
                              Model  Accuracy  Precision    Recall  \
4  ANN (thr tuned on val; calibrated)  0.971154   0.955224  1.000000
```

```
1                      Random Forest  0.980769   0.984375  0.984375
3              SVM (RBF, calibrated)  0.990385   0.984615  1.000000
0                Logistic Regression  0.942308   0.983333  0.921875
2                          KNN (k=5)  0.932692   0.983051  0.906250

         F1   ROC_AUC                                            Notes
4  0.977099  0.999219  Calibrated (Isotonic); Threshold=0.5000 chosen...
1  0.984375  0.998828
3  0.992248  0.998437
0  0.951613  0.990625
2  0.943089  0.961523

=== Class counts ===
class
1    320
0    200
Name: count, dtype: int64
```