CODE PRINTOUT

// **eda_preprocessing.m** //

```
clc;
clear all;
close all;

%% 1. LOAD DATA
data = readtable("D:\ML_Coursework\Medical_Env_Discomfort_Dataset.csv");

X = data(:,1:end-1);   % 11 environmental features
Y = categorical(data.over_discomfort);  % Target (0/1)
Xmat = table2array(X);
fprintf("Rows: %d | Features: %d\n\n", size(Xmat,1), size(Xmat,2));

%% 2. SUMMARY STATISTICS
fprintf("   Summary Statistics   \n");
summaryTable = table( ...
   X.Properties.VariableNames', ...
   mean(Xmat)', ...
   std(Xmat)', ...
   'VariableNames', {'Feature','Mean','StdDev'});
disp(summaryTable);

%% 3. CLASS IMBALANCE
fprintf("\n    Class Distribution    \n");
classCounts = countcats(Y);
disp(table(categories(Y), classCounts,'VariableNames',{'Class','Count'}));

figure;
bar(classCounts);
set(gca,'XTickLabel',categories(Y));
title('Class Distribution (Comfort vs Discomfort)');
xlabel('Class');
ylabel('Count');

%% 4. FEATURE HISTOGRAMS
figure;
tiledlayout(4,3);

for i =1:width(X)
   nexttile;
   histogram(X{:,i});
   title(X.Properties.VariableNames{i},'Interpreter','none');
end
sgtitle("Histograms of All Environmental Features");

%% 5. PEARSON CORRELATION HEATMAP
corrMatrix= corr(Xmat,'Rows','pairwise');

figure;
heatmap(X.Properties.VariableNames,X.Properties.VariableNames,corrMatrix, ...
   'Colormap',redbluecmap,'CellLabelFormat','%.2f');
title("Pearson Correlation Heatmap");
```

```
%% 6. FEATURE DISTRIBUTIONS BY CLASS
figure;
tiledlayout(4,3);

for i= 1:width(X)
   nexttile;
   hold on;
   histogram(X{Y=='0',i},'Normalization','probability','FaceAlpha',0.5);
   histogram(X{Y=='1',i},'Normalization','probability','FaceAlpha',0.5);
   title(X.Properties.VariableNames{i},'Interpreter','none');
   hold off;
end
legend("Comfort = 0","Discomfort = 1");
sgtitle("Feature Distributions Separated by Class");

%% 7. Z-SCORE NORMALISATION
fprintf("\n   Applying Z-Score Normalisation   \n");

X_norm= (Xmat - mean(Xmat)) ./ std(Xmat);
save("normalized_features.mat","X_norm","Y");
fprintf("Normalised dataset saved as 'normalized_features.mat'.\n");
fprintf("EDA & Preprocessing complete.\n");
```

// **train_models_with_CV.m** //

```
clc;
clear;
close all;
rng('default');

%% 1. LOAD DATA
data= readtable("Medical_Env_Discomfort_Dataset.csv");
X= table2array(data(:,1:end-1));
Y= categorical(data.over_discomfort);

%% 2. TRAIN-TEST SPLIT
cvHold= cvpartition(Y,'HoldOut',0.3);
trainIdx= training(cvHold);
testIdx= test(cvHold);

Xtrain= X(trainIdx,:);
Ytrain= Y(trainIdx);

Xtest= X(testIdx,:);
Ytest= Y(testIdx);
fprintf("Train samples: %d | Test samples: %d\n",sum(trainIdx),sum(testIdx));

%% 3. Frequency-based CLASS WEIGHTS (IMBALANCE)
classCounts= countcats(Ytrain);
minorityClass= '0';
majorityClass= '1';
```

```
w0= classCounts(2)/classCounts(1);   % multiply minority
w1= 1;

freqWeights= zeros(size(Ytrain));
freqWeights(Ytrain==minorityClass) = w0;
freqWeights(Ytrain==majorityClass) = w1;

%% 4. LOGISTIC REGRESSION GRID SEARCH
fprintf("\n   Logistic Regression Grid Search (10-fold CV)   \n");

LambdaGrid= [1e-5 1e-4 1e-3 1e-2 1e-1 1 10];
k= 10;   % 10-fold CV
cv= cvpartition(Ytrain,"KFold",k);

LR_results= [];
for L= LambdaGrid
   f1_scores= zeros(k,1);
   for fold= 1:k
      tr= training(cv, fold);
      vl= test(cv, fold);
      mdl = fitclinear(Xtrain(tr,:), Ytrain(tr), ...
         'Learner','logistic', ...
         'Regularization','ridge', ...
         'Lambda',L, ...
         'Weights',freqWeights(tr));

      pred= categorical(predict(mdl,Xtrain(vl,:)));
      [~,~,f1]= prf1(Ytrain(vl),pred,'1');
      f1_scores(fold)= f1;
   end

   LR_results= [LR_results;
   L, mean(f1_scores)];
   fprintf("Lambda = %.5f | CV-F1 = %.3f\n",L,mean(f1_scores));
end

LR_results_table= array2table(LR_results, ...
   'VariableNames',{'Lambda','CV_F1'});
LR_results_table= sortrows(LR_results_table,'CV_F1','descend');
bestLambda= LR_results_table.Lambda(1);
fprintf("\nBest Lambda= %.5f\n",bestLambda);

%% Train Final Logistic Regression
LR_best= fitclinear(Xtrain, Ytrain, ...
   'Learner','logistic', ...
   'Regularization','ridge', ...
   'Lambda',bestLambda, ...
   'Weights',freqWeights);

save("LR_best.mat","LR_best");

%% 5. RANDOM FOREST GRID SEARCH
```

```
fprintf("\n   Random Forest Grid Search (10-fold CV)   \n");

NumTreesGrid= [10 20 50 100 150];
MinLeafGrid= [1 3 5 10];

RF_results= [];

for nt= NumTreesGrid
   for ml= MinLeafGrid
      f1_scores= zeros(k,1);
      for fold = 1:k
         tr= training(cv, fold);
         vl = test(cv, fold);

         RF= TreeBagger(nt, Xtrain(tr,:), Ytrain(tr), ...
            'Method','classification', ...
            'MinLeafSize',ml, ...
            'Weights',freqWeights(tr), ...
            'OOBPrediction','off');

         pred = categorical(predict(RF,Xtrain(vl,:)));
         [~,~,f1]= prf1(Ytrain(vl),pred,'1');
         f1_scores(fold)= f1;
      end

      RF_results= [RF_results;nt,ml,mean(f1_scores)];
      fprintf("Trees=%d | Leaf=%d | CV-F1=%.3f\n",nt,ml,mean(f1_scores));
   end
end

RF_results_table= array2table(RF_results, ...
   'VariableNames',{'NumTrees','MinLeaf','CV_F1'});
RF_results_table= sortrows(RF_results_table,'CV_F1','descend');
bestTrees= RF_results_table.NumTrees(1);
bestLeaf= RF_results_table.MinLeaf(1);

fprintf("\nBest RF Hyperparameters: Trees=%d | Leaf=%d\n",bestTrees,bestLeaf);

%% Train Final Random Forest
RF_best= TreeBagger(bestTrees,Xtrain,Ytrain, ...
   'Method','classification', ...
   'MinLeafSize',bestLeaf, ...
   'Weights',freqWeights, ...
   'OOBPrediction','off');

save("RF_best.mat","RF_best");

%% 6. SAVE DATASETS
save("train_data.mat","Xtrain","Ytrain");
save("test_data.mat","Xtest","Ytest");

disp("Training complete.");
```

```
%% 10-FOLD CV for FINAL LOGISTIC REGRESSION
fprintf("\n   Running 10-fold CV for Final Logistic Regression   \n");

cv= cvpartition(Ytrain,'KFold',10);
LR_prec_scores= zeros(10,1);
LR_rec_scores= zeros(10,1);
LR_f1_scores= zeros(10,1);

for i= 1:10
   tr= training(cv,i);
   vl= test(cv,i);
   mdl = fitclinear(Xtrain(tr,:),Ytrain(tr),...
      'Learner','logistic',...
      'Regularization','ridge',...
      'Lambda',bestLambda,...
      'Weights',freqWeights(tr));

   pred= categorical(predict(mdl,Xtrain(vl,:)));
   [p, r, f]= prf1(Ytrain(vl),pred, '1');
   LR_prec_scores(i) = p;
   LR_rec_scores(i)  = r;
   LR_f1_scores(i)   = f;
end

LR_CV_Prec= mean(LR_prec_scores);
LR_CV_Rec= mean(LR_rec_scores);
LR_CV_F1= mean(LR_f1_scores);

%  Logistic Regression Training Time
tic;
LR_best= fitclinear(Xtrain,Ytrain,...
   'Learner','logistic','Regularization','ridge',...
   'Lambda', bestLambda, 'Weights', freqWeights);
LR_Train_Time = toc;

%% 10-FOLD CV for FINAL RANDOM FOREST
fprintf("\n   Running 10-fold CV for Final Random Forest   \n");

RF_prec_scores= zeros(10,1);
RF_rec_scores = zeros(10,1);
RF_f1_scores= zeros(10,1);

for i= 1:10
   tr= training(cv,i);
   vl= test(cv,i);
   RF_fold = TreeBagger(bestTrees, Xtrain(tr,:), Ytrain(tr), ...
      'Method','classification','MinLeafSize', bestLeaf, ...
      'Weights', freqWeights(tr), 'OOBPrediction','off');

   pred= categorical(predict(RF_fold, Xtrain(vl,:)));
   [p,r,f]= prf1(Ytrain(vl),pred, '1');
```

```
    RF_prec_scores(i)= p;
    RF_rec_scores(i)= r;
    RF_f1_scores(i)= f;
end

RF_CV_Prec= mean(RF_prec_scores);
RF_CV_Rec= mean(RF_rec_scores);
RF_CV_F1= mean(RF_f1_scores);

%    Random Forest Training Time
tic;
RF_best= TreeBagger(bestTrees, Xtrain, Ytrain, ...
    'Method','classification', 'MinLeafSize',bestLeaf, ...
    'Weights',freqWeights,'OOBPrediction','off');
RF_Train_Time = toc;

%% AVG TRAIN AUC (10-FOLD CV)
cv= cvpartition(Ytrain,'KFold',10);
LR_auc_folds= zeros(cv.NumTestSets,1);
RF_auc_folds= zeros(cv.NumTestSets,1);

for i = 1:cv.NumTestSets
    tr = training(cv,i);
    vl = test(cv,i);

    % Logistic Regression
    LR_fold = fitclinear(Xtrain(tr,:), Ytrain(tr), ...
        'Learner','logistic','Regularization','ridge', ...
        'Lambda', bestLambda,'Weights', freqWeights(tr));
    [~, scoresLR]= predict(LR_fold,Xtrain(vl,:));
    LR_probs= scoresLR(:,2);
    [~,~,~,LR_auc_folds(i)]= perfcurve(Ytrain(vl), LR_probs, '1');

    % Random Forest
    RF_fold= TreeBagger(bestTrees, Xtrain(tr,:), Ytrain(tr), ...
        'Method','classification','MinLeafSize', bestLeaf, ...
        'Weights', freqWeights(tr),'OOBPrediction','off');
    [~,scoresRF]= predict(RF_fold,Xtrain(vl,:));
    RF_probs= scoresRF(:,2);
    [~,~,~,RF_auc_folds(i)]= perfcurve(Ytrain(vl), RF_probs,'1');
end

LR_Train_AUC = mean(LR_auc_folds);
RF_Train_AUC = mean(RF_auc_folds);
fprintf("Avg Train AUC = LR: %.3f | RF: %.3f\n", LR_Train_AUC, RF_Train_AUC);

%% AVG TRAIN ERROR (10-FOLD CV)
LR_err= zeros(cv.NumTestSets,1);
RF_err= zeros(cv.NumTestSets,1);

for i= 1:cv.NumTestSets
    tr= training(cv,i);
```

```
    vl= test(cv,i);

    % Logistic Regression
    LR_fold= fitclinear(Xtrain(tr,:), Ytrain(tr), ...
        'Learner','logistic','Lambda',bestLambda, ...
        'Weights',freqWeights(tr));
    predLR= categorical(predict(LR_fold,Xtrain(vl,:)));
    LR_err(i)= mean(predLR ~= Ytrain(vl));

    % Random Forest
    RF_fold = TreeBagger(bestTrees, Xtrain(tr,:), Ytrain(tr), ...
        'Method','classification','MinLeafSize',bestLeaf, ...
        'Weights', freqWeights(tr));
    predRF= categorical(predict(RF_fold,Xtrain(vl,:)));
    RF_err(i)= mean(predRF ~= Ytrain(vl));
end

LR_Train_Error= mean(LR_err);
RF_Train_Error= mean(RF_err);
fprintf("Avg Train Error = LR: %.3f | RF: %.3f\n", LR_Train_Error, RF_Train_Error);

%% FINAL RESULTS SUMMARY
ResultsTable= table( ...
    ["Logistic Regression"; "Random Forest"], ...
    [LR_Train_AUC; RF_Train_AUC], ...
    [LR_Train_Error; RF_Train_Error], ...
    [LR_Train_Time; RF_Train_Time], ...
    'VariableNames', ...
    {'Model','AvgTrainAUC','AvgTrainError','Training_Time_sec'} );

disp("      FINAL RESULTS SUMMARY      ");
disp(ResultsTable);

%% METRIC FUNCTION
function [precision, recall, f1] = prf1(trueLabels, predictedLabels, positiveClass)
    trueLabels = categorical(trueLabels);
    predictedLabels = categorical(predictedLabels);
    TP = sum(predictedLabels==positiveClass & trueLabels==positiveClass);
    FP = sum(predictedLabels==positiveClass & trueLabels~=positiveClass);
    FN = sum(predictedLabels~=positiveClass & trueLabels==positiveClass);
    precision = TP/(TP+FP+eps);
    recall = TP/(TP+FN+eps);
    f1 = 2*precision*recall/(precision+recall+eps);
end

save("CV_metrics.mat","LR_CV_Prec", "LR_CV_Rec", "LR_CV_F1", ...
    "RF_CV_Prec", "RF_CV_Rec", "RF_CV_F1");
```

// **test_LR_model.m** //

```
clc;
clear;
close all;

%% 1. LOAD MODEL & DATA
load("LR_best.mat");   % trained Logistic Regression model
load("test_data.mat");   % Xtest, Ytest

%% 2. PREDICTION
LR_pred= categorical(predict(LR_best,Xtest));
[LR_Test_Prec,LR_Test_Rec,LR_Test_F1]= prf1(Ytest,LR_pred,'1');

%% 3. TEST METRICS
LR_Test_Error= mean(LR_pred ~= Ytest);
fprintf("\nLogistic Regression — Test Results\n");
fprintf("Precision = %.3f\n",LR_Test_Prec);
fprintf("Recall    = %.3f\n",LR_Test_Rec);
fprintf("F1-score  = %.3f\n",LR_Test_F1);
fprintf("Test Error = %.3f\n",LR_Test_Error);

%% 4. CONFUSION MATRIX
figure;
confusionchart(Ytest,LR_pred);
title("Logistic Regression Confusion Matrix (Test Set)");

%% 5. ROC & AUC
[~, LR_scores]= predict( LR_best, Xtest);
LR_probs= LR_scores(:,2);   % probability of class '1'
[LR_X, LR_Y, ~,LR_AUC]= perfcurve(Ytest,LR_probs, '1');

figure;
plot(LR_X,LR_Y,'LineWidth',2);
xlabel('False Positive Rate');
ylabel('True Positive Rate');
title(sprintf('Logistic Regression ROC Curve (AUC = %.3f)', LR_AUC));
grid on;

%% 6. PREDICTION TIME
tic;
predict(LR_best, Xtest);
LR_Predict_Time =toc;

fprintf("Prediction Time(s): %.4f\n", LR_Predict_Time);

%% METRIC FUNCTION
function [precision, recall, f1] = prf1(trueLabels, predictedLabels, positiveClass)
    trueLabels = categorical(trueLabels);
    predictedLabels = categorical(predictedLabels);

    TP = sum(predictedLabels == positiveClass & trueLabels == positiveClass);
    FP = sum(predictedLabels == positiveClass & trueLabels ~= positiveClass);
    FN = sum(predictedLabels ~= positiveClass & trueLabels == positiveClass);
```

```
    precision = TP / (TP + FP + eps);
    recall = TP / (TP + FN + eps);
    f1 = 2 * precision * recall / (precision + recall + eps);
end

save("LR_test_metrics.mat","LR_Test_Prec","LR_Test_Rec","LR_Test_F1");
```

// **test_RF_model.m** //

```
clc;
clear;
close all;

%% 1. LOAD MODEL & DATA
load("RF_best.mat");   % trained Random Forest model
load("test_data.mat");    % Xtest, Ytest

%% 2. PREDICTION
RF_pred= categorical(predict(RF_best,Xtest));
[RF_Test_Prec, RF_Test_Rec, RF_Test_F1]= prf1(Ytest,RF_pred,'1');

%% 3. TEST METRICS
RF_Test_Error= mean(RF_pred ~= Ytest);
fprintf("\nRandom Forest — Test Results\n");
fprintf("Precision = %.3f\n",RF_Test_Prec);
fprintf("Recall    = %.3f\n",RF_Test_Rec);
fprintf("F1-score  = %.3f\n",RF_Test_F1);
fprintf("Test Error = %.3f\n",RF_Test_Error);

%% 4. CONFUSION MATRIX
figure;
confusionchart(Ytest, RF_pred);
title("Random Forest Confusion Matrix (Test Set)");

%% 5. ROC & AUC
[~,RF_scores]= predict(RF_best, Xtest);
RF_probs= RF_scores(:,2);   % probability of class '1'

[RF_X, RF_Y, ~, RF_AUC]= perfcurve(Ytest, RF_probs, '1');

figure;
plot(RF_X, RF_Y,'LineWidth',2);
xlabel('False Positive Rate');
ylabel('True Positive Rate');
title(sprintf('Random Forest ROC Curve (AUC = %.3f)',RF_AUC));
grid on;

%% 6. PREDICTION TIME
tic;
predict(RF_best, Xtest);
```

```
RF_Predict_Time= toc;
fprintf("Prediction Time (s): %.4f\n",RF_Predict_Time);

%%  METRIC FUNCTION
function [precision, recall, f1] = prf1(trueLabels, predictedLabels, positiveClass)
   trueLabels = categorical(trueLabels);
   predictedLabels = categorical(predictedLabels);

   TP = sum(predictedLabels == positiveClass & trueLabels == positiveClass);
   FP = sum(predictedLabels == positiveClass & trueLabels ~= positiveClass);
   FN = sum(predictedLabels ~= positiveClass & trueLabels == positiveClass);

   precision = TP / (TP + FP + eps);
   recall = TP / (TP + FN + eps);
   f1 = 2 * precision * recall / (precision + recall + eps);
end

save("RF_test_metrics.mat", "RF_Test_Prec", "RF_Test_Rec", "RF_Test_F1");
```

// **combined_ROC.m** //

```
clc;
clear;
close all;

%% 1. LOAD MODELS & TEST DATA
load("LR_best.mat");   % Trained Logistic Regression
load("RF_best.mat");   % Trained Random Forest
load("test_data.mat");
load("CV_metrics.mat");
load("LR_test_metrics.mat");
load("RF_test_metrics.mat");

%% 2. LOGISTIC REGRESSION ROC
[~, LR_scores]= predict(LR_best, Xtest);
LR_probs= LR_scores(:,2);   % Probability of class '1'

[LR_X, LR_Y, ~, LR_AUC]= perfcurve(Ytest, LR_probs,'1');

%% 3. RANDOM FOREST ROC
[~, RF_scores] = predict(RF_best, Xtest);
RF_probs = RF_scores(:,2);                    % Probability of class '1'
[RF_X, RF_Y, ~, RF_AUC]= perfcurve(Ytest, RF_probs, '1');

%% 4. COMBINED ROC PLOT
figure;
plot(LR_X, LR_Y,'LineWidth', 2); hold on;
plot(RF_X,RF_Y, 'LineWidth', 2);
plot([0 1],[0 1],'k--');          % Random classifier baseline

legend( ...
```

```
    sprintf('Logistic Regression (AUC = %.3f)', LR_AUC), ...
    sprintf('Random Forest (AUC = %.3f)', RF_AUC), ...
    'Location','southeast');
xlabel('False Positive Rate');
ylabel('True Positive Rate');
title('ROC Curve Comparison: Logistic Regression vs Random Forest');
grid on;

%%  5. DISPLAY AUC VALUES
fprintf("\n    TEST SET AUC COMPARISON    \n");
fprintf("Logistic Regression AUC = %.3f\n", LR_AUC);
fprintf("Random Forest AUC       = %.3f\n", RF_AUC);

%% OVERFITTING CHECK

figure;
metrics = {'Precision','Recall','F1-score'};
values = [ ...
    LR_CV_Prec  LR_Test_Prec;
    LR_CV_Rec   LR_Test_Rec;
    LR_CV_F1    LR_Test_F1 ];

b = bar(values, 'grouped');
b(1).BarWidth = 0.9;
b(2).BarWidth = 0.9;
set(gca,'XTickLabel',metrics)
legend({'10-CV','Test'},'Location','northeast')
title('LR Overfitting Check (CV vs Test)')
ylabel('Score')
grid on
% Zoom Y-axis to highlight differences
ylim([min(values(:))-0.03 1])

% Add value labels
for i = 1:size(values,1)
    for j = 1:2
        text(i + (j-1.5)*0.15, values(i,j), ...
            sprintf('%.3f',values(i,j)), ...
            'HorizontalAlignment','center', ...
            'VerticalAlignment','bottom', ...
            'FontSize',9);
    end
end

figure;
metrics= {'Precision','Recall','F1-score'};
values= [ ...
    RF_CV_Prec  RF_Test_Prec;
    RF_CV_Rec   RF_Test_Rec;
    RF_CV_F1    RF_Test_F1 ];

b = bar(values, 'grouped');
```

```
b(1).BarWidth = 0.9;
b(2).BarWidth = 0.9;
set(gca,'XTickLabel',metrics)
legend({'10-CV','Test'}, 'Location','northwest')
title('RF Overfitting Check (CV vs Test)')
ylabel('Score')
grid on

ylim([min(values(:))-0.03 1])
for i = 1:size(values,1)
   for j = 1:2
      text(i + (j-1.5)*0.15, values(i,j), ...
         sprintf('%.3f',values(i,j)), ...
         'HorizontalAlignment','center', ...
         'VerticalAlignment','bottom', ...
         'FontSize',9);
   end
end
```

AI assistance declaration: I declare that the project objective, experimental design, and data analysis presented in this code are my own work. I utilized AI tools to assist in structuring specific MATLAB functions, debugging logic, and refining technical terminology for poster and supplementary material. All final implementation, hyperparameter tuning, and validation of results were performed independently.