```
% Clear environment
clear; clc; close all;

% Dataset files
dataset_files = {'mental-state.csv', 'emotions.csv',
'merged_mental_emotions.csv'};
dataset_names = {'Mental State', 'Emotions', 'Merged Mental + Emotions'};
num_datasets = length(dataset_files);

% Initialize results storage
accuracies = zeros(num_datasets, 3); % Columns: SVM, kNN, Decision Tree

for i = 1:num_datasets
    fprintf('Processing dataset: %s\n', dataset_files{i});

    % Load dataset
    try
        data = readtable(dataset_files{i}, 'VariableNamingRule', 'preserve');
        fprintf('Dataset loaded: %s [%d x %d]\n', dataset_files{i},
size(data, 1), size(data, 2));
    catch
        error('Failed to load dataset: %s. Ensure it is in the working
directory.', dataset_files{i});
    end

    % Limit dataset size for faster testing
    max_rows = 1000;
    if size(data, 1) > max_rows
        data = data(1:max_rows, :);
        fprintf('Dataset size limited to %d rows for testing.\n', max_rows);
    end

    % Preprocess dataset
    % Handle missing values for different types of columns
    for varIdx = 1:width(data)
        if isnumeric(data{:, varIdx})  % For numeric columns
            data{:, varIdx} = fillmissing(data{:, varIdx}, 'movmean', 5);  %
Use moving average for numeric data
        elseif iscategorical(data{:, varIdx})  % For categorical columns
            data{:, varIdx} = fillmissing(data{:, varIdx}, 'constant',
mode(data{:, varIdx}));  % Mode for categorical data
        elseif ischar(data{:, varIdx})  % For string columns
            data{:, varIdx} = fillmissing(data{:, varIdx}, 'constant',
'Unknown');  % 'Unknown' for string columns
        end
    end

    % Extract labels (assume last column)
    labels = table2array(data(:, end));
    if iscellstr(labels)
```

```
        unique_labels = unique(labels);
        label_map = containers.Map(unique_labels, 1:length(unique_labels));
        labels = cellfun(@(x) label_map(x), labels);
    end

    % Ensure labels are valid
    if isempty(labels)
        error('Labels are empty. Check dataset.');
    end

    % Extract features
    features = table2array(data(:, 1:end-1));

    % Feature Scaling (Standardization)
    features = (features - mean(features)) ./ std(features);

    % Check if features are empty after scaling
    if isempty(features)
        error('Features are empty after scaling. Check dataset or
preprocessing steps.');
    end

    % Check for high correlations and remove highly correlated features
    corr_matrix = corr(features);
    high_corr = abs(corr_matrix) > 0.8;  % Reduce threshold to 0.8
    high_corr = triu(high_corr, 1);  % Only check upper triangle (to avoid
duplicate checks)

    % Remove correlated features
    features = features(:, ~any(high_corr, 1));  % Remove correlated features

    % Check if features are valid after removing correlated features
    if isempty(features)
        % If features are empty after correlation removal, apply PCA
        fprintf('Features are empty after removing highly correlated
features. Applying PCA...\n');

        % Apply PCA for dimensionality reduction (retain 95% variance)
        [coeff, score, ~, ~, explained] = pca(features);

        % Select the number of components that explain 95% of the variance
        cumulative_variance = cumsum(explained);
        num_components = find(cumulative_variance >= 95, 1);

        % Reduce features to the selected number of principal components
        features = score(:, 1:num_components);

        % Ensure that features are non-empty after PCA
        if isempty(features)
```

```
            error('Features are empty after PCA. Consider adjusting
preprocessing steps.');
        end
    end

    % Ensure data split is valid before proceeding
    if size(features, 1) < 2
        error('Not enough data remaining after preprocessing. Check your
dataset and preprocessing steps.');
    end

    % Split data into training and testing sets
    cv = cvpartition(size(features, 1), 'HoldOut', 0.3);
    train_features = features(training(cv), :);
    test_features = features(test(cv), :);
    train_labels = labels(training(cv), :);
    test_labels = labels(test(cv), :);

    % Ensure data split is valid
    if sum(training(cv)) == 0 || sum(test(cv)) == 0
        error('Data split resulted in empty training or test sets.');
    end

    % Ensure that the training data is non-empty
    if isempty(train_features) || isempty(train_labels)
        error('Training data is empty. Check data preprocessing or
splitting.');
    end

    % Train and evaluate classifiers
    % 1. Support Vector Machine (Multi-Class)
    fprintf('Training SVM classifier...\n');
    try
        model_SVM = fitcecoc(train_features, train_labels, 'Learners',
templateSVM('KernelFunction', 'linear'), 'Coding', 'onevsall');
        predicted_SVM = predict(model_SVM, test_features);
        accuracies(i, 1) = sum(predicted_SVM == test_labels) /
numel(test_labels) * 100;
    catch
        fprintf('SVM training failed for dataset: %s\n', dataset_files{i});
        accuracies(i, 1) = NaN;
    end

    % 2. k-Nearest Neighbors
    fprintf('Training k-NN classifier...\n');
    try
        model_kNN = fitcknn(train_features, train_labels, 'NumNeighbors', 5);
        predicted_kNN = predict(model_kNN, test_features);
        accuracies(i, 2) = sum(predicted_kNN == test_labels) /
numel(test_labels) * 100;
```

```
    catch
        fprintf('k-NN training failed for dataset: %s\n', dataset_files{i});
        accuracies(i, 2) = NaN;
    end

    % 3. Decision Tree
    fprintf('Training Decision Tree classifier...\n');
    try
        model_DT = fitctree(train_features, train_labels);
        predicted_DT = predict(model_DT, test_features);
        accuracies(i, 3) = sum(predicted_DT == test_labels) /
numel(test_labels) * 100;
    catch
        fprintf('Decision Tree training failed for dataset: %s\n',
dataset_files{i});
        accuracies(i, 3) = NaN;
    end
end
```

```
Processing dataset: mental-state.csv
Dataset loaded: mental-state.csv [2479 x 989]
Dataset size limited to 1000 rows for testing.
Training SVM classifier...
Training k-NN classifier...
Training Decision Tree classifier...
Processing dataset: emotions.csv
Dataset loaded: emotions.csv [2132 x 2549]
Dataset size limited to 1000 rows for testing.
Training SVM classifier...
Training k-NN classifier...
Training Decision Tree classifier...
Processing dataset: merged_mental_emotions.csv
Dataset loaded: merged_mental_emotions.csv [4611 x 3537]
Dataset size limited to 1000 rows for testing.
Training SVM classifier...
Warning: Unable to fit learner 1 (SVM) because: No data remaining after all rows with NaNs are
removed.
Warning: Unable to fit learner 2 (SVM) because: No data remaining after all rows with NaNs are
removed.
Warning: Unable to fit learner 3 (SVM) because: No data remaining after all rows with NaNs are
removed.
Warning: All binary learners failed to train. This ECOC model will predict into the majority class.
Training k-NN classifier...
Training Decision Tree classifier...
```

```
% Display results
fprintf('\nAccuracy Results (%%):\n');
```

```
Accuracy Results (%):
```

```
disp(array2table(accuracies, 'VariableNames', {'SVM', 'kNN',
'DecisionTree'}, 'RowNames', dataset_names));
```

	SVM	kNN	DecisionTree
Mental State	92.333	88.667	85.333

Emotions	91.333	86.667	96
Merged Mental + Emotions	34.333	34.333	83.333

```
% Visualization of Results
app = uifigure('Name', 'BCI Multi-Dataset Comparison', 'Position', [100,
100, 800, 600]);

% Title
uilabel(app, 'Text', 'Brain-Computer Interface: Multi-Dataset Comparison',
...
    'FontSize', 16, 'FontWeight', 'bold', ...
    'HorizontalAlignment', 'center', ...
    'Position', [100, 550, 600, 30]);

% Accuracy Table
uitable(app, 'Data', [dataset_names', num2cell(accuracies)], ...
    'ColumnName', {'Dataset', 'SVM (%)', 'kNN (%)', 'Decision Tree (%)'}, ...
    'Position', [50, 350, 700, 150]);

% Bar Chart
axes_graph = uiaxes(app, 'Position', [50, 50, 700, 250]);
bar(axes_graph, accuracies);
axes_graph.XTickLabel = dataset_names;
axes_graph.FontSize = 12;
ylabel(axes_graph, 'Accuracy (%)');
title(axes_graph, 'Comparison of Classifiers Across Datasets');
legend(axes_graph, {'SVM', 'kNN', 'Decision Tree'}, 'Location',
'northeastoutside');
```