```
%%Resetting the environment
%clear the command window and any variables, and close all figure windows
clc;
clear;
close all;


%%Loading the dataset
%link for the dataset: https://www.kaggle.com/datasets/uciml/red-wine-quality-cortez-et-al-2009?resource=download
%Using readtable function because our dataset contains not only numeric
%data but also numeric and text columns
redwine = readtable('winequality-red.csv');
% Display the first few rows of the dataset to understand its structure
head(redwine);
summary(redwine);

%%Check for any missing values using the ismissing function
%reference: https://uk.mathworks.com/help/matlab/ref/ismissing.html
missingValues = sum(ismissing(redwine));
missingValues;
%The output shows no missing values for all the variables so we are fine

%%Now we will remove any duplicate rows and then we will
% display the number of duplicate rows removed so we can tell how many rows
% have been removed
%reference: https://www.mathworks.com/matlabcentral/answers/275640-deleting-repeated-rows-in-a-matrix
Before = height(redwine);
redwine = unique(redwine, "rows");
After = height(redwine);
fprintf('Number of duplicate rows removed: %d\n', Before - After);

%%Now, since our problem has to do with the prediction whether the wine is
%of Low, Medium or High quality, the next step is to convert the quality column
%into a binary classification, where 3,4 is Low, 5,6 is Medium and 7,8
%is High quality and create a new column named QualityLabel
q = redwine.quality;

Quality = strings(size(q));
Quality(q <= 4) = "Low";
Quality(q >= 5 & q <= 6) = "Medium";
Quality(q >= 7) = "High";
redwine.QualityLabel = categorical(Quality, ["Low","Medium","High"]);

disp("Distribution of Class (Low, Medium, High):");
summary(redwine.QualityLabel);

%%Next step is to select only the predictor variables (the 11
%%physicochemical input features) and define the categorical target
%%variable
predictors = setdiff(redwine.Properties.VariableNames, {'quality','QualityLabel'});%Here we used the setdiff function
%in order to keep all the columns except quality and QualityLabel which are
%the target variables
Xtable = redwine(:, predictors);
Y = redwine.QualityLabel

%%Now we will standardise our data using the z-score
X = table2array(redwine(:, predictors));
Xz = zscore(X);
```

```
59
60  %%Now we will visualise the predictors with the standardised values using a
61  %%boxplot
62  %reference: https://uk.mathworks.com/help/stats/boxplot.html
63  boxplot(Xz, 'Labels', predictors);
64  title('Boxplot of Physicochemical Features');
65  xlabel('Features');
66  ylabel('Values(standardised)');
67  xtickangle(45);
68  grid on;
69
70  %%Now we will create a heatmap for the correlation between predictors
71  %reference: https://uk.mathworks.com/help/matlab/ref/heatmap.html
72  %First, we create a correlation matrix
73  correlationMatrix = corr(redwine{:, predictors});
74  figure;
75  heatmap = heatmap(predictors, predictors, correlationMatrix);
76  heatmap.Colormap = hot;
77  title('Correlation Matrix of predictors');
78
79  %%Now we will remove any outliers. We will do this for Feature-level and
80  %%Instance-level and compare which method has the strongest suspects of
81  %%outliers.
82
83  %For Feature-level we will use the IQR method
84  %reference: https://uk.mathworks.com/help/matlab/ref/iqr.html
85  %reference: https://uk.mathworks.com/help/matlab/ref/isoutlier.html
86  n = size(Xz, 1);
87  p = size(Xz, 2);
88
89  featureOutlierMask = false(n, p);
90
91  for j=1:p
92      col = Xz(:, j);
93      Q1 = quantile(col,0.25);
94      Q3 = quantile(col, 0.75);
95      IQR = Q3 - Q1;
96      lowerBound = Q1 - 1.5 * IQR;
97      upperBound = Q3 + 1.5 * IQR;
98      featureOutlierMask(:, j) = col < lowerBound | col > upperBound;
99  end
100
101 nFeatureOutliers = sum(featureOutlierMask, 2);
102 redwine.nFeatureOutliers = nFeatureOutliers;
103
104 %Now we set a strong rule for outlying at least 3 features
105 featureLevelFlag = (nFeatureOutliers >= 3);
106
107 fprintf("Feature-level flagged rows: %d\n", sum(featureLevelFlag));
108 %So we have 25 flagged rows in Feature-level
109
110 %Now we move on to Instance-level. We will use the Mahalanobis distance
111 %method. We will compute squared the distance of each row to the
112 %distribution
113 %reference: ChatGPT: Help me detect any outliers for Instance-level and the
114 %find the strongest suspects comparing with the Feature-level outlier
115 %detection.
116 D2 = mahal(Xz, Xz);
117 redwine.MahalanobisD2 = D2;
118
119 %Now, we flag the top 3% as instance-level outliers
```

```
contamination = 0.03;
cutoff = quantile(D2, 1-contamination);
instanceLevelFlag = (D2>=cutoff);

fprintf("Instance-level flagged rows: %d\n", sum(instanceLevelFlag));

%So we have 41 flagged rows in Instance-level

% Now we will combine the feature-level and instance-level flags to identify rows
flagged as outliers
outliers = featureLevelFlag & instanceLevelFlag;
suspects = find(outliers);

fprintf("Strongest outlier suspects: %d\n", numel(suspects));

%Now we will display the results in a table
%reference: ChatGPT: Create a table for the suspects results
TableOfSuspects = redwine(suspects, :);
disp("Outlier suspect row indices:");
disp(suspects);
disp("Preview of suspected outlier rows:");
disp(TableOfSuspects(:, [predictors, {'quality',
'QualityLabel','nFeatureOutliers','MahalanobisD2'}]));

%%Now, before and after removing the outliers, we will perform PCA 2D
%%visualisation in order to see the differrences
%reference: https://uk.mathworks.com/help/stats/pca.html
%reference: https://uk.mathworks.com/help/stats/gscatter.html
[coeff, score] = pca(Xz);

figure;
gscatter(score(:,1), score(:,2), redwine.QualityLabel);
hold on;
plot(score(outliers,1), score(outliers,2), 'ko', 'MarkerSize',8,'LineWidth',1.5);
hold off;
title("PCA visualisation for outliers suspects (before removing them)");
xlabel('PC1');
ylabel('PC2');
grid on;

% Now we will remove the outliers from the dataset and convert a new
% cleaned dataset, X table for the predictors, Xz score and Y table for the
% QualityLabel
redwine_clean = redwine(~outliers, :);
X_clean = table2array(redwine_clean(:, predictors));
Xz_clean = zscore(X_clean);
Y_clean = redwine_clean.QualityLabel;

fprintf("Rows before removing the outliers: %d\n", height(redwine));
fprintf("Rows after removing the outliers: %d\n", height(redwine_clean));

%And now we will perform PCA visualisation again, after removing the
%outliers
[~, score2] = pca(Xz_clean);

figure;
gscatter(score2(:,1), score2(:,2), Y_clean);
title("PCA visualisation (after removing the outliers)");
xlabel('PC1');
ylabel('PC2');
grid on;
```

```
179
180 %%This concludes the pre processing, cleaning and visualising the data. Now
181 %%we can move on to the machine learning algorithms
182
183 %We will train/test the two algorithms with the same techniques for fair
184 %comparison
185 %Firstly, we set a random number generator for reproducibility
186 rng(1)
187
188 %%Now we will split the dataset into training and test (70% train and 30%
189 %%test)
190 %reference: https://www.mathworks.com/matlabcentral/answers/377839-split-training-data-
    and-testing-data?s_tid=srchtitle
191 %First we create a partition
192 cv = cvpartition(Y_clean, 'Holdout', 0.3);
193 trainidx=cv.training;
194 testidx=cv.test;
195
196 %Now we seperate to training and test data
197 Xtrain = Xz_clean(trainidx,:);
198 Ytrain = Y_clean(trainidx,:);
199
200 Xtest = Xz_clean(testidx,:);
201 Ytest = Y_clean(testidx,:);
202
203 %Here we order the classes
204 classOrder = categories(Y_clean);
205
206 %%Next step is to train the Naive Bayes model
207 %reference: https://uk.mathworks.com/help/stats/fitcnb.html
208 %Also, we follow the lecture notes and the tutorial from week 4 of the
209 %module : https://moodle4.city.ac.uk/course/section.php?id=243698
210 NaiveBayes = fitcnb(Xtrain, Ytrain,'DistributionNames','normal');
211
212 %Prediction for our Naive Bayes trained model
213 [YpredictedNaiveBayes, posteriorNaiveBayes] = predict(NaiveBayes, Xtest);
214
215 %%Now we will evaluate the Naive Bayes classifier
216 %Starting with the confusion matrix
217 %reference: https://uk.mathworks.com/help/stats/confusionchart.html
218 confusionmatrix = confusionmat(Ytest, YpredictedNaiveBayes, 'Order', classOrder);
219 figure;
220 confusionchart(Ytest, YpredictedNaiveBayes);
221 title('Confusion Matrix of Naive Bayes');
222
223 %Now we use 5-fold cross-validation to give a more reliable estimate of
224 %performance of our data and to detect overfitting
225 %reference: https://uk.mathworks.com/help/stats/crossval.html
226 crossvalidationModel = crossval(NaiveBayes, 'KFold', 5);
227 crossvalidationLoss = kfoldLoss(crossvalidationModel);
228 fprintf('Cross-validation loss for the Naive Bayes classifier: %.3f\n',
    crossvalidationLoss);
229
230 %Now we will calculate the metrics for the Naive Bayes (f1 score,
231 %precision and recall)
232 %We will first find the true positive, true negative, false positive and
233 %false negative values
234 %reference: https://uk.mathworks.com/matlabcentral/answers/1703920-how-can-i-calculate-
    the-false-negatives-false-positives-and-the-correctly-classified-for-my-seizure
235 classOrder = categories(Y_clean);
236
```

```
TPnaivebayes = diag(confusionmatrix);
FPnaivebayes = sum(confusionmatrix,1)' - TPnaivebayes;
FNnaivebayes = sum(confusionmatrix,2) - TPnaivebayes;
TNnaivebayes = sum(confusionmatrix(:)) - TPnaivebayes  - FPnaivebayes - FNnaivebayes;

%Now we calculate the metrics
%reference: https://medium.com/analytics-vidhya/confusion-matrix-accuracy-precision-recall-f1-score-ade299cf63cd

%Now we calculate the precision
precisionNaiveBayes = TPnaivebayes ./ max(TPnaivebayes+FPnaivebayes,1);

%Now we calculate the recall
recallNaiveBayes = TPnaivebayes ./ max(TPnaivebayes + FNnaivebayes,1);

%Now we calculate the f1 score
f1scoreNaiveBayes = 2*(precisionNaiveBayes .* recallNaiveBayes) ./ max(precisionNaiveBayes+recallNaiveBayes, eps);

fprintf('Per class precision for Naive Bayes:'); disp(precisionNaiveBayes);
fprintf('Per class recall for Naive Bayes:'); disp(recallNaiveBayes);
fprintf('Per class f1 score for Naive Bayes:'); disp(f1scoreNaiveBayes);

%Now, because in multiclass problems we don't have single metrics we will
%compute the macro-metrics(average of the per class metrics for equal weight across classes) and the
%weighted-metrics(average of the per class metrics weighted by support)
%rference: chatGPT: help me compute the macro and weighted metrics
supportNaiveBayes = sum(confusionmatrix,2);

macroPrecisionNaiveBayes = mean(precisionNaiveBayes);
macroRecallNaiveBayes = mean(recallNaiveBayes);
macrof1scoreNaiveBayes = mean(f1scoreNaiveBayes);

weightedPrecisionNaiveBayes = sum(precisionNaiveBayes .* supportNaiveBayes) / sum(supportNaiveBayes);
weightedRecallNaiveBayes = sum(recallNaiveBayes .* supportNaiveBayes) / sum(supportNaiveBayes);
weightedf1scoreNaivebayes = sum(f1scoreNaiveBayes .* supportNaiveBayes) / sum(supportNaiveBayes);

%Now we creat a table to represent the metrics per class
perClassNaiveBayes = table(classOrder, TPnaivebayes, TNnaivebayes, FPnaivebayes, FNnaivebayes, precisionNaiveBayes, recallNaiveBayes, f1scoreNaiveBayes, supportNaiveBayes, 'VariableNames',{'Class','TP', 'TN','FP','FN','Precision','Recall','f1score', 'Support'});
disp('Naive Bayes: Metrics per class');
disp(perClassNaiveBayes);

%%And we are done with the Naive Bayes

%%Now we move on to the Decision Tree model. We will use same training data
%%for a fair comparison
%reference: https://uk.mathworks.com/help/stats/fitctree.html?s_tid=srchtitle_support_results_1_train+decision+tree
DecisionTree = fitctree(Xtrain, Ytrain, 'SplitCriterion','gdi','MinLeafSize',10);

%Prediction for our Decision Tree trained model
[YpredictedDecisionTree, DecisionTreeScores] = predict(DecisionTree, Xtest);

%Visualization of the decision tree
```

```
%reference: https://uk.mathworks.com/support/search.html/answers/1567223-visualization-of-decision-tree.html?fq%5B%5D=asset_type_name:answer&fq%5B%5D=category:stats/data-import-and-export-1&page=1
figure;
view(DecisionTree, 'Mode', 'graph');
title('Decision Tree');

%%Now we will evaluate the Decision Tree classifier
%Starting with the confusion matrix
%reference: https://uk.mathworks.com/help/stats/confusionchart.html
confusionmatrixTree = confusionmat(Ytest, YpredictedDecisionTree, 'Order', classOrder);
figure;
confusionchart(Ytest, YpredictedDecisionTree);
title('Confusion Matrix of Decision Tree');

% Now we will evaluate the performance of the Decision Tree classifier using cross-validation
crossvalidationModelTree = crossval(DecisionTree, 'KFold', 5);
crossvalidationLossTree = kfoldLoss(crossvalidationModelTree);
fprintf('Cross-validation loss for the Decision Tree classifier: %.3f\n', crossvalidationLossTree);

%Now we will calculate the metrics for the Decision Tree (f1 score,
%precision and recall)
%We will first find the true positive, true negative, false positive and
%false negative values
classOrder = categories(Y_clean);

TPTree = diag(confusionmatrixTree);
FPTree = sum(confusionmatrixTree,1)' - TPTree;
FNTree = sum(confusionmatrixTree,2) - TPTree;
TNTree= sum(confusionmatrixTree(:)) - TPTree  - FPTree - FNTree;

%Now we calculate the metrics
%reference: https://medium.com/analytics-vidhya/confusion-matrix-accuracy-precision-recall-f1-score-ade299cf63cd

%Now we calculate the precision
precisionTree = TPTree ./ max(TPTree+FPTree,1);

%Now we calculate the recall
recallTree = TPTree./ max(TPTree + FNTree,1);

%Now we calculate the f1 score
f1scoreTree= 2*(precisionTree .* recallTree) ./ max(precisionTree+recallTree, eps);

fprintf('precision for Decision Tree:'); disp(precisionTree);
fprintf('recall for Decision Tree:'); disp(recallTree);
fprintf('f1 score for Decision Tree:'); disp(f1scoreTree);

%Now, because in multiclass problems we don't have single metrics we will
%compute the macro-metrics(average of the per class metrics for equal weight across classes) and the
%weighted-metrics(average of the per class metrics weighted by support)
%rference: chatGPT: help me compute the macro and weighted metrics
supportTree = sum(confusionmatrixTree,2);

macroPrecisionTree = mean(precisionTree);
macroRecallTree = mean(recallTree);
macrof1scoreTree = mean(f1scoreTree);

```

```
weightedPrecisionTree = sum(precisionTree .* supportTree) / sum(supportTree);
weightedRecallTree = sum(recallTree .* supportTree) / sum(supportTree);
weightedf1scoreTree = sum(f1scoreTree .* supportTree) / sum(supportTree);

%Now we creat a table to represent the metrics per class
perClassTree = table(classOrder, TPTree, TNTree, FPTree, FNTree,
precisionTree, recallTree, f1scoreTree, supportTree, 'VariableNames',{'Class','TP',
'TN','FP','FN','Precision','Recall','f1score', 'Support'});
disp('Decision Tree: Metrics per class');
disp(perClassTree);

% Now we will visualize the performance metrics for both classifiers
% Now we will plot the performance metrics for both classifiers
figure;
bar([macroPrecisionTree, macroRecallTree, macrof1scoreTree; weightedPrecisionTree,
weightedRecallTree, weightedf1scoreTree]);
set(gca, 'XTickLabel', {'Macro', 'Weighted'});
ylabel('Scores');
title('Performance Metrics Comparison');
legend('Precision', 'Recall', 'F1 Score');

%And this concludes the code

```