CROP TYPE PREDICTION ANALYSIS

Name: Chai Shou Zheng
Student ID: 34035958
Last Modified: 5:26pm 17/5/2025

Q1)

Class = 0: "Other": **86.36%** (4318/5000)
Class = 1: "Oats": **13.64%** (682/5000)

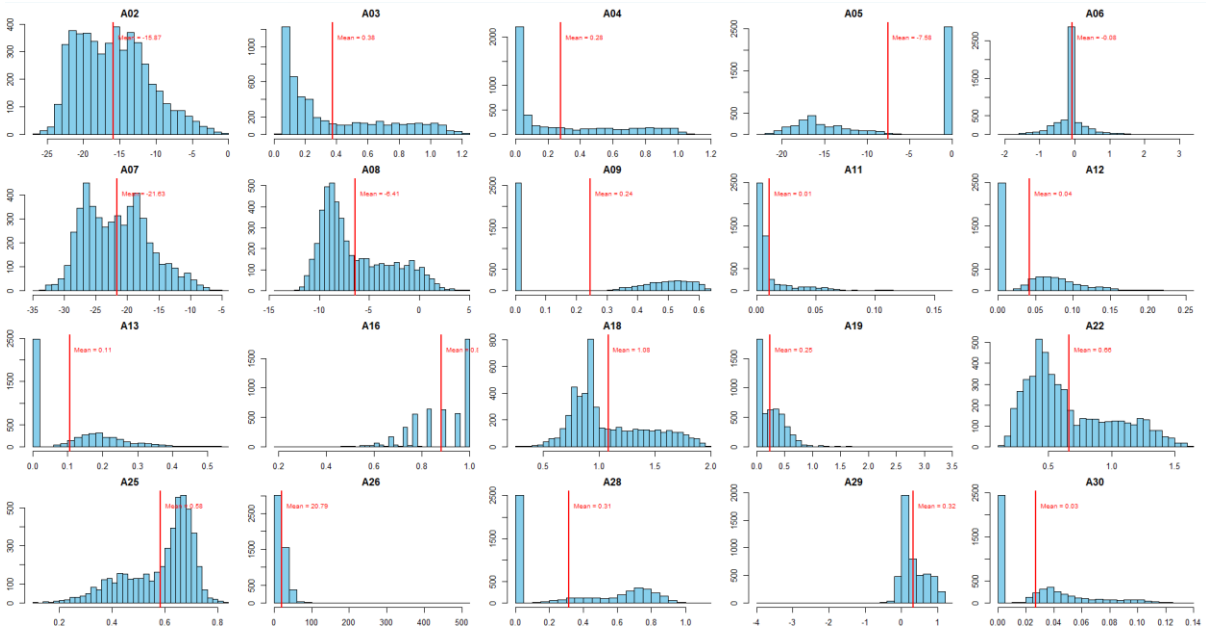
The dataset contains **5,000 observations** and **21 variables**

- **20 independent (predictor) variables**
- **1 dependent (target) variable**

Each independent variable column (A01–A30) is a **numerical continuous variables** representing Radar Sensing Data. Negative or zero values are expected and valid in Sensing Data, particularly for features involving statistical summaries such as correlations.

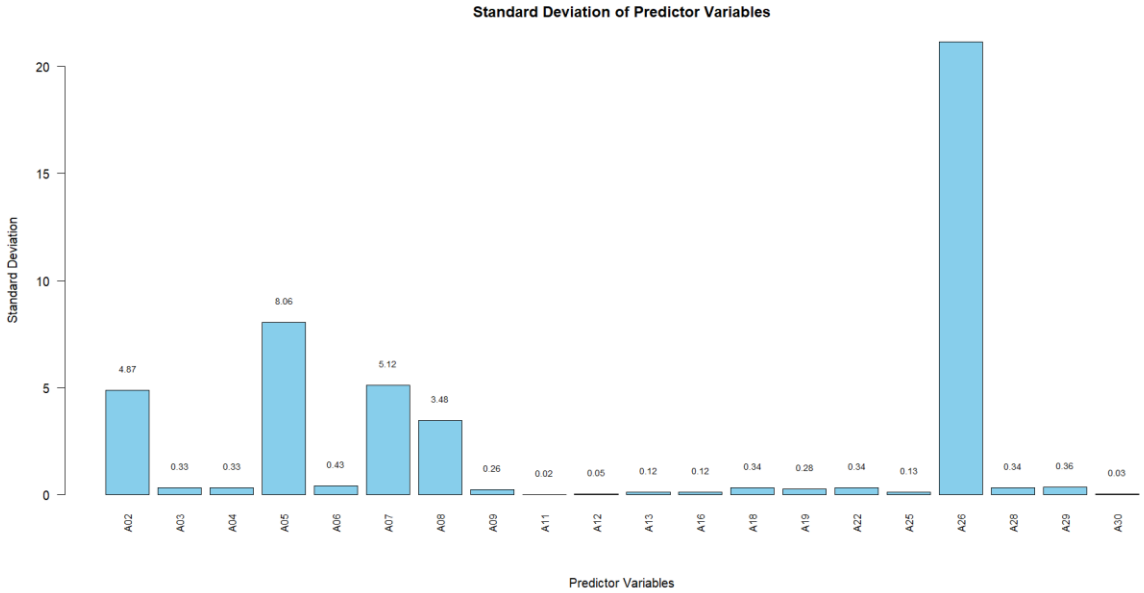
The target variable 'Class' is stored as an **integer** for a binary classification problem, where 'Oats' is represented as 1 and 'Other' as 0.

Mean of the Predictor Variables



The 20 predictor variables in the dataset exhibit a wide range of mean values. Several features such as **A02 (-15.87)**, **A05 (-7.58)**, **A06 (-0.07)**, **A07 (-21.63)**, and **A08 (-6.41)** have negative means. **A11 (0.01)** and **A30 (0.03)**, have means close to zero. On the other hand, **A26 (20.79)** has a notably high mean compared to the others. Most of the remaining predictors have moderate positive means, typically between **0.2 and 1.0**. Overall, these observations suggest that while the dataset is numerically diverse.

Standard Deviation of the Predictor Variables



The standard deviations of the 20 predictor variables reveal a mix of high and low variability across the dataset. Features such as **A02 (4.87)**, **A05 (8.06)**, **A07 (5.12)**, and particularly **A26 (21.15)** exhibit high standard deviations, indicating strong potential for differentiating between crop types and are likely to be informative for classification. On the other hand, some features display very low variability, such as **A11 (0.02)**, **A12 (0.05)**, and **A30 (0.03)**, may contribute minimally to model performance. Overall, most features show moderate variation.

Noteworthy Observations in the Data

- The dataset exhibits a highly imbalanced class distribution.
  Class = 0: "Other": 86.36%
  Class = 1: "Oats": 13.64%
  This imbalance may affect classifier performance as some models might not handle skewed distribution well, potentially leading to biased predictions.

- There are **no missing values** in the dataset
- Among all the predictors, **A26** has exceptionally high mean (≈20.79) and standard deviation (≈21.15)

Attributes to Consider Omitting

A11, **A12**, and **A30** have **the lowest three standard deviations, near zero** among all predictors, indicating that almost all the values are the same across the dataset (i.e., there's very little to no variation). These low-variance features are unlikely to contribute meaningfully to model performance and could be considered for removal as might help reduce noise and prevent overfitting.

Q2)

Data pre-processing

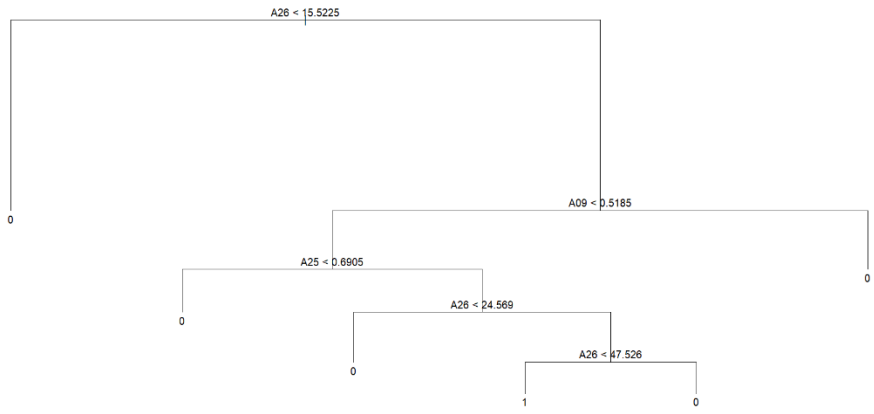
- The target variable 'Class' is stored as an integer. For compatibility with all classification model, we convert it into a **factor.**

	Feature	Proportion_Zero
1	A02	0.0000
2	A03	0.0000
3	A04	0.0000
4	A05	0.5100
5	A06	0.5048
6	A07	0.0000
7	A08	0.0000
8	A09	0.5106
9	A11	0.4958
10	A12	0.4972
11	A13	0.4956
12	A16	0.0000
13	A18	0.0000
14	A19	0.3654
15	A22	0.0000
16	A25	0.0000
17	A26	0.0000
18	A28	0.5014
19	A29	0.0018
20	A30	0.4894

- Several predictor variables contain a high proportion of zero values (close to or exceeding 50%), most notably **A05**, **A06**, **A08**, **A09**, **A11**, **A12**, **A13**, **A28**, and **A30**. These zeros are considered valid measurements in our analysis. For this analysis, **all original features were retained to allow each classifier to naturally learn and adapt from the full dataset. Predictors with a high proportion of zero values were not removed, as we aim to evaluate the performance of different classification models under consistent input conditions and compare their effectiveness accordingly.** However, these attributes have been flagged as potential candidates for removal or transformation in future modelling stages, particularly if they demonstrate low predictive value.

Q4)

Decision Tree



The decision tree is shallow and mostly predicts class 0 ("Other"), with only one narrow path leading to class 1 ("Oats"). This indicates that the model is conservative in predicting "Oats", due to class imbalance in the training data. While the tree captures a simple structure, its limited depth and bias toward class 0 suggest that it may not accurately identify “Oats” cases, resulting in lower recall for class 1.

Q5)

Decision Tree Confusion Matrix

	Actual Class	
Predicted Class	0	1
0	1235	150
1	77	38

Naïve Bayes Confusion Matrix

	Actual Class	
Predicted Class	0	1
0	1175	141
1	137	47

Bagging Confusion Matrix

	Observed Class	
Predicted Class	0	1
0	1292	170
1	20	18

Boosting Confusion Matrix

	Observed Class	
Predicted Class	0	1
0	1279	152
1	33	36

Random Forest Confusion Matrix

	Actual Class	
Predicted Class	0	1
0	1294	166
1	18	22

Performance of Each Classifier in Q4

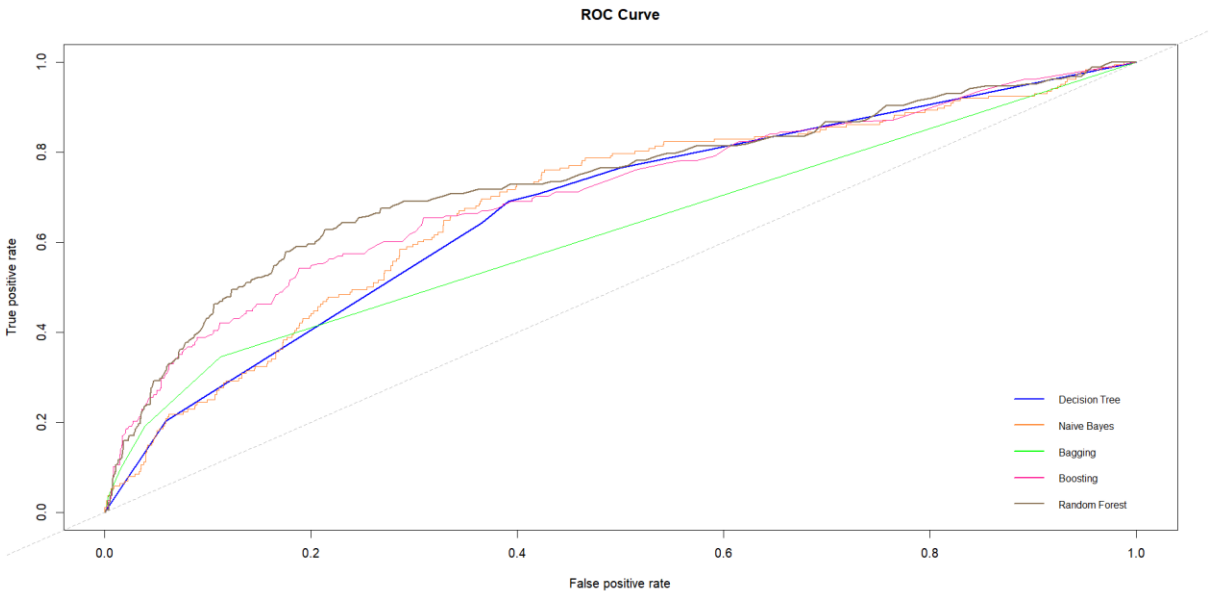
Metric	Decision Tree	Naive Bayes	Boosting	Bagging	Random Forest
Accuracy	0.8487	0.8147	0.8767	0.8733	0.8773
Precision	0.3304	0.2554	0.5217	0.4737	0.5500
Recall	0.2021	0.2500	0.1915	0.0957	0.1170
F1 Score	0.2508	0.2527	0.2802	0.1593	0.1930
AUC	0.6734	0.6841	0.7089	0.6213	0.7335

The performance metrics across all five classifiers shows varying strengths. Random Forest achieved the highest accuracy (0.8773) and precision (0.55), indicating that it correctly predicted a large portion of test cases and was highly reliable when predicting "Oats". However, its recall (0.1170) was the lowest, meaning it failed to identify most of the actual "Oats" cases (i.e., low true positive count). This is likely because the dataset has fewer "Oats" cases compared to "Other" due to the class imbalanced, which causes most models to predict "Other" more often and have difficulty correctly identifying the "Oats" class.

Boosting showed the most balanced performance, with a strong F1-score (0.2802), the highest among all classifiers. This suggests it provided a better trade-off between precision and recall. Naive Bayes had moderate and consistent performance, while Bagging offered high precision but low recall, similar to Random Forest.

Overall, while Random Forest performs strongly in making highly accurate predictions, Boosting may be more suitable in maximizing true positive, where identifying as many actual "Oats" instances as possible is important.

Q6)



Based on the ROC curve analysis, **Random Forest** demonstrates the best overall performance, achieving the highest true positive rate across most false positive rates. This indicates that it has the largest area under the ROC curve, suggesting strong discriminative power. Boosting also performs well, followed by Bagging, while Decision Tree and Naive Bayes exhibit comparatively lower performance. Overall, Random Forest and Boosting are more effective in this classification task than classifiers.

AUC for each classifier

```
# AUC Scores
> print_auc("Decision Tree", res.tree$pred)
AUC (Decision Tree): 0.6734
> print_auc("Naive Bayes", res.nb$pred)
AUC (Naive Bayes): 0.6841
> print_auc("Bagging", res.bag$pred)
AUC (Bagging): 0.6213
> print_auc("Boosting", res.boost$pred)
AUC (Boosting): 0.7089
> print_auc("Random Forest", res.rf$pred)
AUC (Random Forest): 0.7335
```

Based on the AUC values, Random Forest achieved the highest AUC score (0.7335), followed by Boosting (0.7089) and Naïve Bayes (0.6841). These classifiers demonstrate better overall performance in distinguishing between “Oats” (class = 1) and “Other” (class = 0) across different classification thresholds.

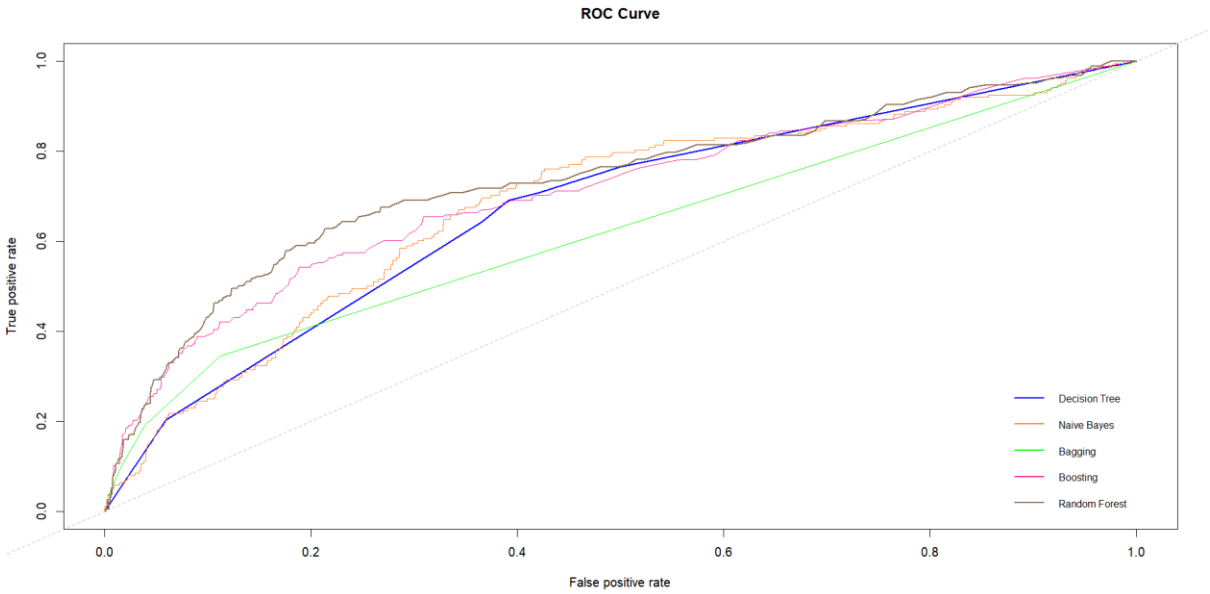
Decision Tree had a moderate AUC (0.6734), while Bagging recorded the lowest (0.6213), suggesting it has the weakest ability to separate the two classes with confidence.

A higher AUC indicates that a model is better at ranking positive instances (i.e., “Oats”) higher than negative ones, and thus Random Forest and Boosting show the strongest predictive power.

Q7)

Table Comparing the Results from Question 5 and Question 6 for All Classifiers

Metric	Decision Tree	Naive Bayes	Boosting	Bagging	Random Forest
Accuracy	0.8487	0.8147	0.8767	0.8733	0.8773
Precision	0.3304	0.2554	0.5217	0.4737	0.5500
Recall	0.2021	0.2500	0.1915	0.0957	0.1170
F1 Score	0.2508	0.2527	0.2802	0.1593	0.1930
AUC	0.6734	0.6841	0.7089	0.6213	0.7335



Based on the comparison of accuracy, precision, recall, F1-score, ROC and AUC across all classifiers, there is no single model that dominates all metrics. However, Boosting demonstrates the most balanced performance, achieving the highest F1 Score (0.2802), which reflects a good balance between precision and recall.

Although Random Forest has the highest accuracy (0.8773) and precision (0.5500), its recall (0.1170) is extremely low, which limits its usefulness in identifying class “Oats”.

In terms of confidence-based performance, Random Forest achieves the highest AUC (0.7335), followed closely by Boosting (0.7089), suggesting strong ability to distinguish between the classes across different thresholds.

This is further supported by the ROC curve in Q6, where the Random Forest curve consistently stays above the others, indicating superior performance in terms of true positive rate vs. false positive rate. However, this strength in AUC is weakened by its low recall, as it has the lowest recall, which is important for detecting “Oats”.

Therefore, when considering classification accuracy, (AUC), and the impact of class imbalance especially performance on the minority class “Oats” (reflected in F1-score, precision and recall), we think **Boosting is the best overall classifier** due to its balanced metrics and reliable detection of the “Oats” class, making it the most practical model in this imbalanced classification scenario.

Investigation Tasks

Q8)

Determine the most important attributes of each classifier as follows:

Decision Tree

```
Variables actually used in tree construction:
[1] "A26" "A09" "A25"
```

The decision tree model used only three attributes in its prediction: **A26, A09, and A25**. This suggests that these attributes were the most informative for distinguishing between “Oats” and “Other” in the dataset. Other attributes were not used at all, implying they could potentially be removed with minimal impact on performance.

Bagging

```
      A02       A03       A04       A05       A06
5.2544004 5.2180323 0.0000000 4.8007755 7.9626877
      A07       A08       A09       A11       A12
2.6719786 1.7861613 8.8244796 0.5108493 0.8034044
      A13       A16       A18       A19       A22
2.5411265 2.8008208 3.0595087 0.0000000 2.5199091
      A25       A26       A28       A29       A30
8.0244325 33.0496061 4.9635029 1.5824398 3.6258845
```

```
==============================
Model: Bagging
------------------------------
Top 3 Most Important:
      A26       A09       A25
33.049606  8.824480  8.024432
Bottom 3 Least Important:
      A04       A19       A11
0.0000000 0.0000000 0.5108493
==============================
```

In the Bagging model, all attributes contributed to the classification except for A04 and A19, which had an score of 0. The most influential attribute was **A26 (33.05), followed by A09 (8.82) and A25 (8.02)**, showing strong alignment with the decision tree model. Several attributes such as A11, A12, and A29 had relatively low importance scores, suggesting they have minimal impact on performance.

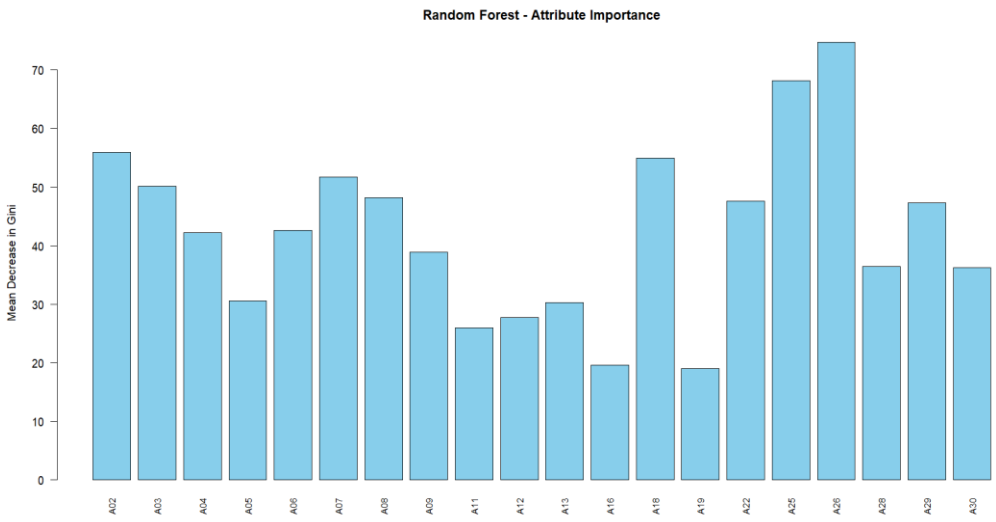
Boosting

```
      A02       A03       A04       A05       A06       A07
5.425434  5.736899  6.345730  3.748504  4.320038  6.482977
      A08       A09       A11       A12       A13       A16
4.511335 10.560772  1.717222  3.714678  1.608370  0.000000
      A18       A19       A22       A25       A26       A28
3.913252  1.370947  2.560762  7.527776 20.779032  1.915051
      A29       A30
3.544671  4.216549
```

```
==============================
Model: Boosting
------------------------------
Top 3 Most Important:
      A26       A09       A25
20.779032 10.560772  7.527776
Bottom 3 Least Important:
      A16       A19       A13
0.000000 1.370947 1.608370
==============================
```

In the Boosting model, all attributes contributed to the classification except for A16, which had an importance score of 0. The most important attribute was **A26 (20.78), followed by A09 (10.56) and A25 (7.53)**, consistent with patterns seen in the decision tree and bagging models. Attributes like A11, A13, A19, and A28 showed relatively low importance.

Random Forest



	MeanDecreaseGini
A02	55.92455
A03	50.16896
A04	42.24636
A05	30.60444
A06	42.59300
A07	51.72785
A08	48.23415
A09	38.90046
A11	26.01270
A12	27.73307
A13	30.28292
A16	19.66397
A18	54.98424
A19	19.03876
A22	47.62263
A25	68.19913
A26	74.72354
A28	36.44153
A29	47.29133
A30	36.28755

```
==============================
Model: Random Forest
------------------------------
Top 3 Most Important:
     A26      A25      A02
74.72354 68.19913 55.92455
Bottom 3 Least Important:
     A19      A16      A11
19.03876 19.66397 26.01270
==============================
```

In the Random Forest model, all attributes contributed to the classification, with no variable having a zero importance score. The most influential features were **A26 (74.72), A25 (68.20), and A02 (55.92)**, followed closely by A18 (54.98) and A07 (51.73). On the other hand, attributes such as A16 (19.66) and A19 (19.04) had the lowest importance scores.

Top three most important attributes and bottom three least important attributes

Model	Most important attributes	Least important attributes
Boosting	A26, A09, A25	A16, A19, A13
Random Forest	A26, A02, A25	A16, A19, A11
Bagging	A26, A09, A25	A04, A19, A11
Decision Tree	A26, A09, A25	

Most important attributes: **A26, A25, A09**
Least important attributes (can be omitted): **A16, A19, A11**

Q9)

Decision Tree

- Decision Tree is prone to overfitting, especially on small or noisy datasets. However, in our prediction, Decision Tree performed poorly because it built a shallow tree using only 3 attributes, likely due to the imbalanced nature of the extracted dataset, where "Oats" are much fewer than "Other." This resulted in underfitting, as the tree could not fully capture the complexity of the data or correctly identify many "Oats" instances, leading to low recall and F1-score.

Naive Bayes

- Naive Bayes assumes that all features are independent of one another given the class, which doesn't hold in our dataset. In our extracted dataset, many predictors may have interactions or correlations — especially since the attributes are likely derived from sensor or where values are not statistically independent. This mismatch between the model's assumptions and the actual data structure causes Naive Bayes to oversimplify the decision boundaries, leading to only moderate performance. While it maintained balanced precision and recall, it was unable to effectively capture deeper patterns.

Bagging

- Bagging reduces model variance by training multiple decision trees on random subsets of the data and averaging their predictions. This ensemble approach helps improve accuracy and stability, especially on noisy datasets. In our case, Bagging achieved high overall accuracy and precision, indicating it correctly classified the majority class ("Other") very well.

- However, due to the imbalance in our extracted dataset, Bagging struggled to consistently capture minority class instances. Since each bootstrap sample may contain very few or no "Oats" cases, the ensemble tends to be biased toward the dominant class, resulting in low recall. This means Bagging often failed to detect actual "Oats" cases, even though it was confident when it did predict them, leading to high precision but low recall.

Boosting

- Boosting works by focusing on hard-to-classify instances, adjusting the weights of training samples so that each new tree pays more attention to the errors made by the previous ones. This iterative correction process makes it especially effective for imbalanced datasets, where the minority class "Oats" is much less frequent than "Other."

- By continuously re-weighting misclassified "Oats" instances, Boosting is able to learn complex patterns and reduce bias toward the majority class. In our results, this is reflected in its highest F1-score and highest AUC, showing that Boosting not only made accurate predictions overall but also performed well in identifying the minority "Oats" class. This balance between precision and recall makes Boosting particularly suitable for situations where correctly identifying rare cases (like "Oats") is important.

Random Forest

- Random Forest builds on the Bagging method by introducing additional feature randomness — each tree in the forest is trained not only on a bootstrap sample of the data, but also on a random subset of features at each split. This helps reduce correlation between trees, improving generalization and reducing overfitting.

- In our dataset, Random Forest achieved the highest accuracy and precision, meaning it correctly predicted most cases and was particularly reliable when it predicted the "Oats" class. However, it still exhibited low recall, which indicates difficulty in identifying many of the actual "Oats" instances. This is largely due to the class imbalance in our data, resulting in fewer true positives for "Oats".

- Despite its overall strong performance, Random Forest's low sensitivity limits its usefulness when detecting minority class cases is a priority.

Conclusion

Boosting and Random Forest outperformed simpler models such as the Decision Tree and Naive Bayes, especially in F1-score and AUC because they can because they actively learn from errors and mitigate bias toward the majority class, whereas simpler or variance-focused models tend to favor the dominant class.

Q10)

Justification in Decision

- We selected the **decision tree** from Q4 because it clearly shows the decision path and the splitting conditions, making it simple enough for a person to manually classify instances as either class 0 ("Other") or class 1 ("Oats").

Selection of Attributes

- We used the top three predictors — A26, A09, and A25, which were consistently identified as the most important attributes across all classifiers and the decision tree in Question 4. These variables appeared in the earliest splits of the decision tree, indicating strong discriminatory power between the "Oats" and "Other" classes.

Steps to create Simplified Decision Tree

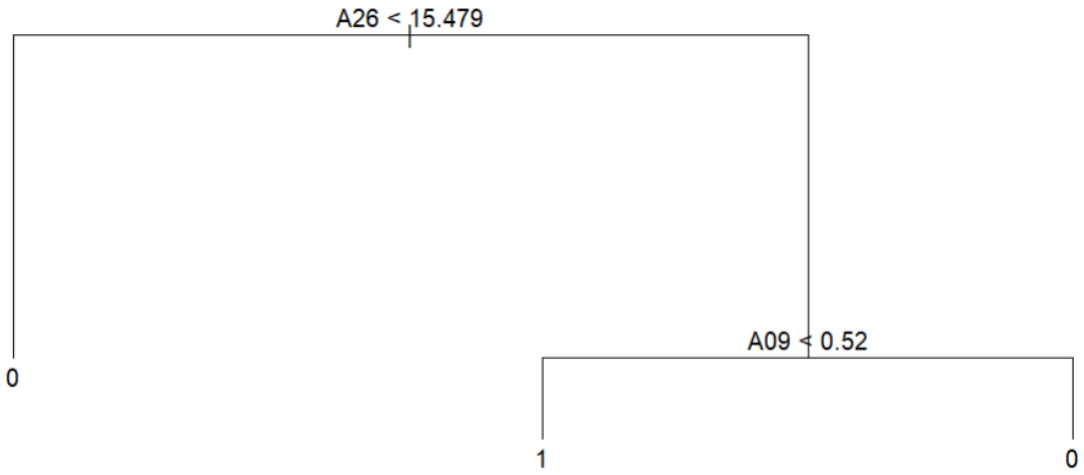
1. **Class Rebalancing**
   To ensure fairness in classification and avoid bias toward the "Other" class, I first rebalanced the dataset by sampling 440 instances from each class.

2. **Model Training and Pruning**
   I then trained a new decision tree using only the three selected predictors on this balanced dataset. I applied cross-validation using the to find the optimal number of nodes based on misclassification error. Based on this, I then pruned the tree to reduce complexity and prevent overfitting. This resulted in a simpler, more interpretable model that still maintained reasonable classification performance and was suitable for manual use.

Testing on the Original (Unbalanced) Test Set in Q3

- To ensure a fair comparison with previous models from Questions 3 and 4, I evaluated the simplified decision tree on the original unbalanced test set. This allowed me to directly compare performance with other classifiers under the same conditions.

Written Explanation of the Model



This decision tree uses two predictors A26 and A09 after pruning, and classifies samples as either "Oats" (1) or "Other" (0) based on the following logic:

1. If A26 < 15.479, then classify the instance as "Other" (class = 0).
2. Else, check A09:
    - If A09 < 0.5135, then classify as "Oats" (class = 1).
    - Otherwise, classify as "Other" (class = 0).

This model is simple and easy to apply manually. A person can classify a new instance by checking just two values — A26 and A09 — and following the conditions step-by-step to arrive at a decision.

Simplified Decision Tree Confusion Matrix and Performances

	Observed Class	
Predicted Class	0	1
0	757	54
1	555	134

```
-------------------------
Model: Simplified Decision Tree on Q3
Accuracy:  0.5940
Precision: 0.1945
Recall:    0.7128
F1 Score:  0.3056
-------------------------
```

Comparison of Simplify Decision Tree vs. Other Classifiers

Metric	Simplify Decision Tree	Decision Tree	Naive Bayes	Boosting	Bagging	Random Forest
Accuracy	0.5940	0.8487	0.8147	0.8767	0.8733	0.8773
Precision	0.1945	0.3304	0.2554	0.5217	0.4737	0.5500
Recall	0.7128	0.2021	0.2500	0.1915	0.0957	0.1170
F1 Score	0.3056	0.2508	0.2527	0.2802	0.1593	0.1930
AUC	0.6488	0.6734	0.6841	0.7089	0.6213	0.7335

Accuracy:

- The simplified decision tree has the lowest accuracy because it was trained on a balanced dataset and is less complex, focusing on fairness. The other models were trained on unbalanced data, which led to high accuracy but also heavy bias toward predicting the majority class (class 0, "Other"). This inflates accuracy but doesn't mean better performance on “Oats” class detection.

Precision:

- Precision measures how many predicted class 1 instances were actually correct. The Q10 model has lower precision because it predicts class 1 more often (higher recall), which increases false positives (predicting "Oats" when it is actually "Other"). In contrast, the other models predict class 1 rarely, but they miss many actual "Oats" cases, resulting in lower recall.

Recall:

- Recall measures how many actual class 1 (“Oats”) cases were correctly identified. The Q10 simplified decision tree achieves the highest recall among all models. One reason is that it was trained on a balanced dataset, giving equal attention to both “Oats” and “Other” classes. This allowed the model to learn distinguishing patterns for “Oats” more effectively.

- Another contributing factor is the simplicity of the model. With only a few clear splits (as shown in the diagram: A26 and A09), the tree prioritizes strong, high-impact features, making it more likely to capture class 1 cases without overfitting to noise or less relevant attributes.

- In contrast, more complex models trained on imbalanced data tend to focus on the majority class, causing them to overlook many “Oats” cases, resulting in lower recall.

F1 Score:

- The F1 score balances precision and recall. Despite having lower accuracy and precision, the Q10 model has the highest F1 score because of its strong recall. This indicates that it performs better at identifying class 1 cases overall, making it more useful when detecting the minority class is a priority.

Conclusion / Why It Performs Well

- **Balanced Training Data:** Prevented bias toward “Other”.
- **Simplicity:** Few splits (only A26 and A09) made the tree interpretable and less prone to overfitting.
- **Focus on Minority Class:** Prioritized identifying “Oats”, which is rare in the dataset.

Q11)

To build the best tree-based classifier, I chose to optimize the **Random Forest model**.

Key Factors in My Decision of Choosing Random Forest

1. **Overall good Model Performance**
   The Random Forest model achieved one of the highest AUC, ROC and accuracy scores among others tree-based classifiers, indicating strong ability to distinguish between classes. It also provided a better balance between precision and recall, which is especially important for detecting the minority "Oats" class. Also, Random Forest is well-suited for handling high-dimensional data, reducing overfitting.

2. **Interpretability Through Feature Importance**
   The model's built-in variable importance metrics (e.g., MeanDecreaseGini) helped identify and remove less impactful predictors

Steps to create Improved Random Forest Model

1. Class Rebalancing
I balanced the training data by randomly sampling 440 instances from each class. This step ensured that the model would not be biased toward the majority class ("Other") and could better learn to identify the minority class ("Oats").

2. Removing Least Impactful Predictors
To reduce noise and improve interpretability, I removed the least important predictors based on their MeanDecreaseGini values. These features contributed the least to reducing node impurity and were unlikely to improve predictive performance.

3. Model Tuning with Cross-Validation
I used the caret::train() function to perform 10-fold cross-validation, tuning the mtry parameter (i.e., the number of predictors considered at each split). The optimal value selected was mtry = 2. This approach helped ensure the model generalized well to unseen data by selecting the mtry value that maximized predictive power.

4. Testing on the Original (Unbalanced) Test Set in Q3

To ensure a fair comparison with previous models from Questions 3 and 4, I evaluated the improved Random Forest on the original unbalanced test set. This allowed me to directly compare accuracy, precision, recall, F1-score, and AUC with other classifiers under the same conditions.

Why I Chose the Attributes I Used

I selected the final set of attributes based on their variable importance as measured by MeanDecreaseGini. This metric reflects each attribute's contribution to reducing node impurity in the ensemble of decision trees.

To improve performance and reduce overfitting, I removed the least important predictors ("A16", "A19"), which had the lowest Gini scores and were likely contributing minimal value to the model's decisions, therefore reducing the noice, allowing the model to focus on stronger features like A25, A26, A02.

Improved RF Confusion Matrix and Performances

	Observed Class	
Predicted Class	0	1
0	979	25
1	333	163

```
-------------------------
Model: Improved RF on Q3 Test
Accuracy:  0.7613
Precision: 0.3286
Recall:    0.8670
F1 Score:  0.4766
-------------------------
```

Comparison of Improved Random Forest vs. Other Classifiers

Metric	Improved Random Forest	Original Random Forest	Decision Tree	Naive Bayes	Boosting	Bagging
Accuracy	0.7613	0.8773	0.8487	0.8147	0.8767	0.8733
Precision	0.3286	0.5500	0.3304	0.2554	0.5217	0.4737
Recall	0.8670	0.1170	0.2021	0.2500	0.1915	0.0957
F1 Score	0.4766	0.1930	0.2508	0.2527	0.2802	0.1593
AUC	0.9028	0.7335	0.6734	0.6841	0.7089	0.6213

Recall

- Recall is the most important metric in this model as it tells us how many actual "Oats" samples were correctly identified.

- The original model correctly identifies only 11.7% of actual "Oats" cases due to being trained on an imbalanced dataset that heavily favors the "Other" class.

- The Improved Random Forest correctly identified 86.7% of “Oats”, a massive improvement—higher than all other classifiers.

AUC

- AUC measures the model’s ability to rank a randomly chosen “Oats” higher than a randomly chosen “Other” across all thresholds.

- The Improved RF has an AUC of 0.9028, higher than the original RF (0.7335) and the highest among all classifiers, confirming stronger discrimination capability.

F1 Score

- F1 balances precision and recall. The original RF scored 0.1930, while the improved version increased to 0.4766, the highest F1 score across all classifiers, showing the most balanced performance.

Accuracy

- The original model have high accuracy because the dataset is imbalanced. So, by mostly predicting "Other", the model gets a lot of predictions correct, even though it's doing a poor job on "Oats".

- The improved model was trained on a balanced dataset, so it makes more "Oats" predictions. This increases the chance of false positives (predicting "Oats" when it's actually "Other"), reducing overall accuracy.

- However, in imbalanced classification tasks, high accuracy can be misleading. The drop in accuracy is acceptable here because other critical metrics improve significantly.

Precision

- In the original model, the classifier predicted “Oats” very rarely — only when it was very confident. (Result: High precision, but it missed most Oats, low recall.)

- In the improved model, we rebalanced the dataset, so the model is now more willing to predict “Oats”. That improves recall because it catches more actual “Oats”. But it also leads to more false positives, where the model mistakenly calls “Other” an “Oats” which causing the precision drops.

- As the model becomes better at finding “Oats” (increasing recall), it also risks including more incorrect ones, lowering precision. This is a precision–recall trade-off, and the improved model makes a reasonable compromise to better detect the minority class.

Summary
The improved Random Forest model demonstrates a significant enhancement in in recall, F1-score, and AUC, making it the most suitable model when the goal is to detect as many "Oats" instances as possible in an imbalanced dataset.

Q12)

Pre-processing

- To implement ANN classifier, I first preprocessed the dataset to ensure it was suitable for training. The "Class" variable was converted to a numeric format to make it compatible with the neural network model.

- To address the issue of class imbalance, I rebalanced the dataset by randomly sampling 680 observations from Class 0 and 680 from Class 1, resulting in a balanced dataset of 1360 observations. This ensures that the classifier learns equally from both classes.

- ANNs only works when input features are numeric and on similar scales. In this case, the features selected (A26, A09, and A25) represent Sensing Data, which are already on comparable numerical scales. As a result, further scaling or normalization was not necessary.

Testing on the Original (Unbalanced) Test Set in Q3
To further compare the model with others classifiers in Q4 we also tested the trained neural network on the original unbalanced test set obtained from a 70/30 split in Q3.

Attributes Used

- For predictors, I selected the three most common top-performing attributes identified across the five classifiers in Question 4 (A26, A09, and A25). These variables consistently contributed to better classification results and were therefore used as inputs to the neural network.

- The ANN was trained using the neuralnet package in R, with the number of hidden nodes set to 3, which provides a balance between model complexity and generalization.

ANN Confusion Matrix and Performances

	Observed Class	
Predicted Class	0	1
0	792	520
1	62	126

```
------------------------
Model: Neural Network
Accuracy:  0.6120
Precision: 0.6702
Recall:    0.1950
F1 Score:  0.3022
------------------------
```

Comparison of ANN vs. Other Classifiers

Metric	ANN	Random Forest	Decision Tree	Naive Bayes	Boosting	Bagging
Accuracy	0.6120	0.8773	0.8487	0.8147	0.8767	0.8733
Precision	0.6702	0.5500	0.3304	0.2554	0.5217	0.4737
Recall	0.1950	0.1170	0.2021	0.2500	0.1915	0.0957
F1 Score	0.3022	0.1930	0.2508	0.2527	0.2802	0.1593
AUC	0.6656	0.7335	0.6734	0.6841	0.7089	0.6213

The ANN classifier achieves an F1 Score of 0.3022, which is the highest among all models evaluated in Question 4. While its overall accuracy (0.6120) is lower than Random Forest (0.8773) and Bagging (0.8733). Also, ANN provides a much more meaningful balance between precision (0.6702) and recall (0.1950).

This trade-off indicates that the ANN is more effective at correctly identifying the minority class ("Oats"), as reflected by its relatively good recall. At the same time, it maintains a reasonably high precision, meaning a good proportion of the predicted "Oats" cases are actually correct. In contrast, Random Forest and Boosting achieved high overall accuracy by mostly predicting the majority class ("Other"), which led to very low recall for "Oats" (e.g., only 0.1170 for Random Forest), making them less reliable in detecting the minority class.

The superior F1 Score of the ANN are largely due to the balanced dataset and selecting the top features during training. The F1 Score reflects the balance between precision and recall, highlighting the model's ability to perform well when both false positives and false negatives matter. Unlike unbalanced models that were biased toward the dominant class, the ANN was trained on equal numbers of both classes (680 each), allowing it to learn meaningful distinctions between "Oats" and "Other." The attributes used (A26, A09, A25) were chosen based on previous feature importance analysis, ensuring that the input to the ANN was both relevant and discriminative.

Overall, the ANN outperformed all other classifiers in terms of F1 Score, making it the most effective model for identifying the minority class ("Oats"). It offers the best trade-off between precision and recall. This makes the ANN particularly suitable for applications where capturing less frequent but important cases (like "Oats") is critical.

Why ANN Perform Better

ANN can outperform models like decision trees, random forests, or Naive Bayes when the dataset contains complex, non-linear relationships. ANNs are capable of learning hidden patterns that simpler models may fail to capture. However, they typically require a large and well-balanced dataset to train effectively and avoid overfitting. To address this, I rebalanced the data by selecting an equal number of samples from both classes—"Other" (Class 0) and "Oats" (Class 1)—ensuring fair representation during training and testing.

To maintain simplicity while still allowing the network to learn meaningful patterns, I used a single hidden layer with three neurons. These design choices enabled the ANN to generalize well and achieve strong performance compared to other models, especially in terms of F1 Score, which are critical for accurately detecting the minority "Oats" class.

Q13)

Classifier: **Support Vector Machine (SVM)**
Package: **e1071**

Link: https://www.geeksforgeeks.org/classifying-data-using-support-vector-machinessvms-in-r/

How the SVM Model Works
SVM is a supervised learning method that tries to find the best boundary (called a hyperplane) that separates different classes — in this case, “Oats” (class = 1) and “Other” (class = 0).

If the data isn’t easily separated by a straight line (which often happens in real-world problems), SVM uses something called a kernel to transform the data into a new space where it becomes easier to draw a boundary. We used the RBF (Radial Basis Function) kernel, which is good at handling this kind of non-linear data.

Example:
Imagine we have sensor data like A26, A09, and A25. The SVM looks at these values and learns to separate “Oats” from “Other” by drawing the best possible curve through the data — so it correctly classifies as many Oats as possible while avoiding misclassifying Others.

Pre-processing Steps

- The original dataset was imbalanced. To address this, we randomly sampled 380 records from Class 1 and Class 0 to create a balanced training set (total 760 rows).

- The target variable Class was converted to a factor to indicate it is a classification problem (required by svm()).

Attribute Selection

- "A16" and "A19" were excluded from the dataset before model training, as they consistently showed low importance across all models.

Testing on the Original (Unbalanced) Test Set in Q3
To further compare the model with others classifiers in Q4 we also tested the trained neural network on the original unbalanced test set obtained from a 70/30 split in Q3.

SVM Confusion Matrix and Performances

	Observed Class	
Predicted Class	0	1
0	834	47
1	478	141

```
------------------------
Model: SVM
Accuracy:  0.6500
Precision: 0.2278
Recall:    0.7500
F1 Score:  0.3494
------------------------
```

Comparison of SVM vs. All Other Classifiers

Metric	SVM	ANN	Improved Random Forest	Simplify Decision Tree	Random Forest	Decision Tree	Naive Bayes	Boosting	Bagging
Accuracy	0.6500	0.6120	0.7613	0.5940	0.8773	0.8487	0.8147	0.8767	0.8733
Precision	0.2278	0.6702	0.3286	0.1945	0.5500	0.3304	0.2554	0.5217	0.4737
Recall	0.7500	0.1950	0.8670	0.7128	0.1170	0.2021	0.2500	0.1915	0.0957
F1 Score	0.3494	0.3022	0.4766	0.3056	0.1930	0.2508	0.2527	0.2802	0.1593
AUC	0.7416	0.6656	0.9028	0.6488	0.7335	0.6734	0.6841	0.7089	0.6213

All classifiers were evaluated using the same unbalanced test set from Question 3, ensuring a fair comparison under class imbalance.

Comparison of SVM vs Other Classifiers

Among all models, SVM and Improved Random Forest classifier shows strong performance compared to others models. Both achieves the highest recall among all classifiers, indicating they correctly identifies most of the actual "Oats" cases — a major improvement over the original Random Forest, which had a recall of only 0.1170.

Although SVM has the lowest accuracy (0.6500) due to more false positives, this drop is expected in imbalanced classification problems where detecting the minority class is more important than overall correctness. Its precision (0.2278) is lower because it predicts more "Oats", increasing the risk of misclassifying "Other" as "Oats". However, this trade-off results in a significantly higher F1 score (0.3494), which balances precision and recall better than any of the earlier models.

In terms of AUC (0.7416), SVM and Improved Random Forest also performs the best. Both SVM and Improved Random Forest outperformed traditional models like Naive Bayes, Bagging, and even the original Random Forest, particularly in recall and AUC. This highlights their effectiveness in identifying the minority "Oats" class while still maintaining good overall performance.

Conclusion

Considering both recall and AUC, SVM and Improved Random Forest are the most suitable models for this classification task. They demonstrate the best trade-off between detecting rare "Oats" cases and minimizing false positives, making them ideal for real-world deployment where correct identification of minority cases is essential.

Appendix:

```
# ------------------------------------------------
# R script: FIT3152_Ass2
# Project: FIT3152 Assignment2
#
# Date: 17/5/2025
# Time: 3:54AM
# Author: CHAI SHOU ZHENG
# Dataset: 'WinnData.csv'
# ------------------------------------------------
# clean up the environment before starting
rm(list = ls())

# Load libraries
library(tree)
library(e1071)
library(ROCR)
library(randomForest)
library(adabag)
library(rpart)
library(neuralnet)
library(car)
library(future)
library(caret)
library(adabag)

# Set the directory
setwd("FIT3152/Assignment2")

# Create individual data set
rm(list = ls())
set.seed(34035958)
WD = read.csv("WinnData.csv")
WD = WD[sample(nrow(WD), 5000, replace = FALSE),]
WD = WD[,c(sort(sample(1:30,20, replace = FALSE)), 31)]

# Save the extracted data set as a new CSV file
write.csv(WD, "Extracted_WinnData_34035958.csv", row.names = FALSE)
```

```
# ------------------------------------------------
# Q1. Data Exploration
# ------------------------------------------------

dim(WD) # View the dimensions of the dataset
str(WD) # View the structure of the dataset
colSums(is.na(WD)) # Check for missing values in each column

# Proportion of “Oats” (class = 1) vs “Other” (class = 2)
table(WD$Class) # View counts of each class

# View proportions of each class as percentages
round(prop.table(table(WD$Class))* 100, 2)

########################
# Desricptive Anaylsis
########################

# Calculate means of all 20 predictors
colMeans(WD[, -ncol(WD)])  # exclude the last column 'Class'(dependent variable)

# Plot histograms of mean for all predictors
par(mfrow = c(4, 5))  # 4x5 layout

for (col in colnames(WD[, -ncol(WD)])) {
  h <- hist(WD[[col]],
         main = paste(col),
         xlab = paste(col, "Value"),
         ylab = "Frequency",
         col = "skyblue",
         breaks = 30)

  # Draw mean line
  mean_val <- mean(WD[[col]])
  abline(v = mean_val, col = "red", lwd = 2)

  # Place label just below top bar height
  text(
    x = mean_val,
    y = max(h$counts) * 0.9,  # slightly below top
    labels = paste("Mean =", round(mean_val, 2)),
    pos = 4,
    col = "red",
    cex = 0.75
  )
}

par(mfrow = c(1, 1))  # Reset layout
```

```r
# Standard deviation of all 20 predictors (excluding 'Class')
sapply(WD[, -ncol(WD)], sd)

# Calculate standard deviations
std_devs <- sapply(WD[, -ncol(WD)], sd)

# Set Y-axis limit slightly above max
y_max <- max(std_devs) + 2  # Add padding for labels

# Create bar plot of standard deviation for all predictors
bp <- barplot(
  std_devs,
  main = "Standard Deviation of Predictor Variables",
  ylab = "Standard Deviation",
  xlab = "",
  col = "skyblue",
  las = 2,
  cex.names = 0.8
)

# Add numeric values on top of each bar
text(
  x = bp,
  y = std_devs + 0.5,
  labels = round(std_devs, 2),
  pos = 3,         # Position above the bar
  cex = 0.7,       # Font size
  col = "black"
)

# Add x-axis label
mtext("Predictor Variables", side = 1, line = 5)

# ------------------------------------------------
# Q2. Data Preprocessing
# ------------------------------------------------
# Convert the target variable 'Class' to a factor
WD$Class <- as.factor(WD$Class)

# Calculate proportion of zeros for each predictor
zero_props <- colMeans(WD[, -ncol(WD)] == 0)

# Convert the proportions into a tidy data frame for easier inspection or plotting
zero_df <- data.frame(
  Feature = names(zero_props),
  Proportion_Zero = as.numeric(zero_props)
)

print(zero_df)
```

```
# ------------------------------------------------
# Q3. Train-Test Split
# ------------------------------------------------
set.seed(34035958)  # set seed

# Create a 70-30 train-test split
train.row <- sample(1:nrow(WD), 0.7 * nrow(WD))

# Subset the dataset into training and testing sets
WD.train <- WD[train.row, ] # 70% training data
WD.test  <- WD[-train.row, ] # 30% testing data

# ------------------------------------------------
# Q4. Classification Model Implementation
# ------------------------------------------------

#################
#  Decision Tree
#################

# Train a Decision Tree model using the training dataset
WD.tree <- tree(Class ~ ., data = WD.train)
print(summary(WD.tree)) # Display summary

# Visualize the structure of the decision tree
plot(WD.tree)
text(WD.tree, pretty = 0)


#################
#  Naïve Bayes
#################

# Train a Naïve Bayes classifier using default settings
WD.bayes <- naiveBayes(Class ~ ., data = WD.train)


#################
#  Bagging
#################
set.seed(34035958)
# Train a Bagging ensemble model using 5 bootstrap samples
# Each sample is used to train a separate decision tree
WD.bag <- bagging(Class ~ ., data = WD.train, mfinal = 5)


#################
#  Boosting
#################
set.seed(34035958)
# Train a Boosting ensemble model with 10 trees
```

```
WD.boost <- boosting(Class ~ ., data = WD.train, mfinal = 10)


##################
#  Random Forest
##################
set.seed(34035958)
# Train a Random Forest classifier using default parameters
WD.rf <- randomForest(Class ~ ., data = WD.train, na.action = na.exclude)


# ------------------------------------------------
# Q5.Classification Performance Evaluation
# ------------------------------------------------

# function of evaluate confusion metrics
evaluate_confusion_metrics <- function(model_name, cm) {
 # Assumes cm = table(Predicted, Actual) or similar 2x2 confusion matrix
 TP <- cm["1", "1"]
 TN <- cm["0", "0"]
 FP <- cm["1", "0"]
 FN <- cm["0", "1"]

 accuracy  <- (TP + TN) / sum(cm)
 precision <- TP / (TP + FP)
 recall    <- TP / (TP + FN)
 f1_score  <- 2 * precision * recall / (precision + recall)

 # Print evaluation results
 cat("\n-------------------------")
 cat(sprintf("\nModel: %s", model_name))
 cat(sprintf("\nAccuracy:  %.4f", accuracy))
 cat(sprintf("\nPrecision: %.4f", precision))
 cat(sprintf("\nRecall:    %.4f", recall))
 cat(sprintf("\nF1 Score:  %.4f", f1_score))
 cat("\n-------------------------\n")
}


#################
# Decision Tree
#################

# Predict classes on test data
WD.predtree.class <- predict(WD.tree, WD.test, type = "class")

# Confusion matrix
cat("\n# Decision Tree Confusion Matrix\n")
conf.tree <- table(Predicted_Class = WD.predtree.class, Actual_Class = WD.test$Class)
print(conf.tree)
```

```
evaluate_confusion_metrics("Decision Tree", conf.tree)


#################
#  Naïve Bayes
#################

# Predict class labels
WD.predbayes.class <- predict(WD.bayes, WD.test)

# Confusion matrix
cat("\n# Naïve Bayes Confusion Matrix\n")
conf.nb <- table(Predicted_Class = WD.predbayes.class, Actual_Class = WD.test$Class)
print(conf.nb)

evaluate_confusion_metrics("Naive Bayes", conf.nb)


#################
#  Bagging
#################

# Predict using Bagging model
WD.predbag <- predict.bagging(WD.bag, WD.test)

# Confusion matrix
cat("\n# Bagging Confusion Matrix\n")
print(WD.predbag$confusion)

evaluate_confusion_metrics("Bagging", WD.predbag$confusion)


#################
#  Boosting
#################

# Predict using Boosting model
WD.predboost <- predict.boosting(WD.boost, newdata = WD.test)

# Confusion matrix
cat("\n# Boosting Confusion Matrix\n")
print(WD.predboost$confusion)

evaluate_confusion_metrics("Boosting", WD.predboost$confusion)


##################
#  Random Forest
##################

# Predict class labels
```

```
WD.predrf.class <- predict(WD.rf, WD.test)

# Confusion matrix
cat("\n# Random Forest Confusion Matrix\n")
conf.rf <- table(Predicted_Class = WD.predrf.class, Actual_Class = WD.test$Class)
print(conf.rf)

evaluate_confusion_metrics("Random Forest", conf.rf)


# ------------------------------------------------
# Q6. ROC Curve and AUC Comparison
# ------------------------------------------------

# Function which returns prediction and performance objects for ROC curve based on predicted
probabilities and true labels
get_roc_perf <- function(prob_vector, true_labels) {
  pred <- ROCR::prediction(prob_vector, true_labels)
  perf <- ROCR::performance(pred, "tpr", "fpr")
  return(list(pred = pred, perf = perf))
}

# Function which prints AUC given a model name and prediction object
print_auc <- function(model_name, pred_obj) {
  auc <- ROCR::performance(pred_obj, "auc")@y.values[[1]]
  cat(sprintf("AUC (%s): %.4f\n", model_name, auc))
}


###################
#  Decision Tree
###################

# Predict class probabilities on test data
WD.predtree.prob <- predict(WD.tree, WD.test, type = "vector")

# Get ROC prediction and performance
res.tree  <- get_roc_perf(WD.predtree.prob[, 2], WD.test$Class)
WD.tree.pred <- res.tree$pred
WD.tree.perf <- res.tree$perf

# Plot ROC curve
plot(WD.tree.perf, main = "ROC Curve", col = "blue", lwd = 2)
abline(0, 1, lty = 2, col = "gray")  # Random baseline

# AUC
print_auc("Decision Tree", WD.tree.pred)


#################
#  Naïve Bayes
```

```
#################

# Predict class probabilities for ROC
WD.predbayes.prob <- predict(WD.bayes, WD.test, type = "raw")

# Get ROC prediction and performance
res.nb  <- get_roc_perf(WD.predbayes.prob[, 2], WD.test$Class)
WD.nb.pred <- res.nb$pred
WD.nb.perf <- res.nb$perf

# Add Naïve Bayes ROC to the existing ROC plot
plot(WD.nb.perf, add = TRUE, col = "chocolate1")

# AUC
print_auc("Naive Bayes", WD.nb.pred)


##################
#  Bagging
##################

# Get ROC prediction and performance
res.bag  <- get_roc_perf(WD.predbag$prob[, 2], WD.test$Class)
WD.bag.pred <- res.bag$pred
WD.bag.perf <- res.bag$perf

# Add to existing ROC plot
plot(WD.bag.perf, add = TRUE, col = "green")

# AUC
print_auc("Bagging", WD.bag.pred)


#################
#  Boosting
#################

# Get ROC prediction and performance
res.boost <- get_roc_perf(WD.predboost$prob[, 2], WD.test$Class)
WD.boost.pred <- res.boost$pred
WD.boost.perf <- res.boost$perf

# Add to existing ROC plot
plot(WD.boost.perf, add = TRUE, col = "deeppink")

# AUC
print_auc("Boosting", WD.boost.pred)


###################
#  Random Forest
```

```
###################

# Predict class probabilities for ROC
WD.predrf.prob <- predict(WD.rf, WD.test, type = "prob")

# Get ROC prediction and performance
res.rf <- get_roc_perf(WD.predrf.prob[, 2], WD.test$Class)
WD.rf.pred <- res.rf$pred
WD.rf.perf <- res.rf$perf

# Add to existing ROC plot
plot(WD.rf.perf, add = TRUE, col = "burlywood4", lwd = 2)

# AUC
print_auc("Random Forest", WD.rf.pred)

# add a legend
legend("bottomright",
    legend = c("Decision Tree", "Naive Bayes", "Bagging", "Boosting", "Random Forest"),
    col = c("blue", "chocolate1", "green", "deeppink", "burlywood4"),  # match your current colors
    lty = 1,
    lwd = 2,
    cex = 0.8,
    box.lty = 0)


# ------------------------------------------------
# Q7. Summary of Classifier Performance
# ------------------------------------------------

# Create a dataframe with all the evaluation metrics
results_q7 <- data.frame(
  Model = c("Decision Tree", "Naive Bayes", "Bagging", "Boosting", "Random Forest"),
  Accuracy = c(0.8487, 0.8147, 0.8733, 0.8767, 0.8773),
  Precision = c(0.3304, 0.2554, 0.4737, 0.5217, 0.5500),
  Recall = c(0.2021, 0.2500, 0.0957, 0.1915, 0.1170),
  F1_Score = c(0.2508, 0.2527, 0.1593, 0.2802, 0.1930)
)

# Print the table
print(results_q7)


cat("\n# AUC Scores\n")
print_auc("Decision Tree", res.tree$pred)
print_auc("Naive Bayes", res.nb$pred)
print_auc("Bagging", res.bag$pred)
print_auc("Boosting", res.boost$pred)
print_auc("Random Forest", res.rf$pred)

# ------------------------------------------------
```

```
# Investigation Tasks
# ------------------------------------------------

# ------------------------------------------------
# Q8. Attribute Importance
# ------------------------------------------------

# Decision Tree model
cat("\n# Decision Tree Attribute Importance\n")
print(summary(WD.tree))  # Shows which attributes were used and how many times they appear in splits

# Bagging model
cat("\n# Bagging Attribute Importance\n")
print(WD.bag$importance)  # Higher values indicate greater contribution to classification in bagged trees

# Boosting model
cat("\n# Boosting Attribute Importance\n")
print(WD.boost$importance)  # Importance is based on frequency and quality of splits in boosting iterations

# Random Forest model
cat("\n# Random Forest Attribute Importance\n")
print(WD.rf$importance)  # Measures decrease in node impurity (e.g. Gini) or accuracy when attribute is permuted


# Create bar graph of attribute important for RF
# Extract importance values
importance_vals <- WD.rf$importance[, 1]  # Select the MeanDecreaseGini column
attribute_names <- rownames(WD.rf$importance)
importance_vals


# Create barplot to visualize Random Forest Attribute Importance
barplot(
  importance_vals,
  names.arg = attribute_names,
  las = 2,  # Rotate labels vertically
  col = "skyblue",
  main = "Random Forest - Attribute Importance",
  ylab = "Mean Decrease in Gini",
  cex.names = 0.8
)


#####################################
# Identify Top and Bottom Attributes
#####################################
```

```
# Define a helper function to find top and bottom 3 attributes based on importance
get_top_bottom_attributes <- function(importance_vector, model_name) {
  sorted <- sort(importance_vector, decreasing = TRUE)
  top3 <- head(sorted, 3)

  # Get the full sorted list
  full_sorted <- sort(importance_vector, decreasing = FALSE)
  bottom3 <- head(full_sorted, 3)

  cat("\n==============================\n")
  cat(sprintf("Model: %s\n", model_name))
  cat("------------------------------\n")
  cat("Top 3 Most Important:\n")
  print(top3)
  cat("Bottom 3 Least Important:\n")
  print(bottom3)
  cat("==============================\n")
}

# Bagging
get_top_bottom_attributes(WD.bag$importance, "Bagging")

# Boosting
get_top_bottom_attributes(WD.boost$importance, "Boosting")

# Random Forest (extract from MeanDecreaseGini column)
get_top_bottom_attributes(WD.rf$importance[, 1], "Random Forest")

# ------------------------------------------------
# Q10. Simplified Decision Tree
# ------------------------------------------------
# Re-balance the class
# Identify indices by class
class0_other_idx <- which(WD$Class == 0)  # "Other"
class1_oats_idx  <- which(WD$Class == 1)  # "Oats"

# Sample equal size from both classes
set.seed(34035958)
sample_other_q10 <- sample(class0_other_idx, 600)
sample_oats_q10  <- sample(class1_oats_idx, 600)

# Combine to form balanced training set
balanced_idx_q10 <- c(sample_other_q10, sample_oats_q10)

# Assign training and test sets for Q10
WD.train_q10 <- WD[balanced_idx_q10, ]
WD.test_q10  <- WD[-balanced_idx_q10, ]

# Check the class distribution
table(WD.train_q10$Class)
```

```
# Fit decision tree using only A26, A09, and A25
WD.tree_selected <- tree(Class ~ A26 + A09 + A25, data = WD.train_q10)

# 2. Cross-validation to find the optimal size using misclassification rate
set.seed(34035958)
cv_result_selected <- cv.tree(WD.tree_selected, FUN = prune.misclass)

# 3. Find the best tree size (lowest deviance)
best_size_selected <- cv_result_selected$size[which.min(cv_result_selected$dev)]

# 4. Prune the tree to that size
WD.tree_selected_pruned <- prune.tree(WD.tree_selected, best = best_size_selected)

# 5. Optional: view summary of the pruned tree
summary(WD.tree_selected_pruned)

# Plot the tree
plot(WD.tree_selected_pruned)
text(WD.tree_selected_pruned, pretty = 0)

# Predict using the pruned tree from Q10 on Q3 test data
WD.predtree_q3 <- predict(WD.tree_selected_pruned, WD.test, type = "class")

# Confusion matrix for Q3 test set
conf.tree_q3 <- table(Predicted = WD.predtree_q3, Actual = WD.test$Class)
cat("\n# Simplified Decision Tree on Q3 Test Set\n")
print(conf.tree_q3)

# Evaluate confusion matrix metrics
evaluate_confusion_metrics("Simplified Decision Tree on Q3 Test", conf.tree_q3)

# 4. Predict probabilities (for AUC)
WD.tree_q3.prob <- predict(WD.tree_selected_pruned, WD.test, type = "vector")

# 5. Create prediction object using probabilities for class 1
WD.tree_q3.pred <- ROCR::prediction(WD.tree_q3.prob[, 2], WD.test$Class)

# 6. Print AUC
print_auc("Simplified Decision Tree on Q3", WD.tree_q3.pred)

# ------------------------------------------------
# Q11. Improved Random Forest
# ------------------------------------------------
# Re-balance the class

# Identify indices by class
class0_other_idx <- which(WD$Class == 0)  # "Other"
class1_oats_idx  <- which(WD$Class == 1)  # "Oats"

# Sample equal size from both classes (e.g., 500 each)
set.seed(34035958)
```

```
sample_other <- sample(class0_other_idx, 440)
sample_oats  <- sample(class1_oats_idx, 440)

# Combine to form balanced training set
balanced_idx <- c(sample_other, sample_oats)

# Assign new training and test sets
WD.train_sampled <- WD[balanced_idx, ]
WD.test_sampled  <- WD[-balanced_idx, ]

# Remove Least Important Predictors
cols_to_remove <- c("A16", "A19")
WD.train_sampled <- WD.train_sampled[, !(names(WD.train_sampled) %in% cols_to_remove)]
WD.test_sampled  <- WD.test_sampled[, !(names(WD.test_sampled) %in% cols_to_remove)]

set.seed(34035958)
WD.rf.best <- randomForest(Class ~ .,
                           data = WD.train_sampled,
                           mtry = 2,
                           ntree = 300,
                           importance = TRUE)

# Remove the same least important predictors from the Q3 test set
WD.test_q3 <- WD.test[, !(names(WD.test) %in% c("A16", "A19"))]

# 2. Make predictions using the improved model
pred.rf.q3 <- predict(WD.rf.best, WD.test_q3, type = "class")

# Confusion matrix
conf.rf.q3 <- table(Predicted = pred.rf.q3, Actual = WD.test_q3$Class)
cat("\n# Confusion Matrix: Improved RF on Q3 Unbalanced Test Data\n")
print(conf.rf.q3)

# Evaluate confusion matrix metrics
evaluate_confusion_metrics("Improved RF on Q3 Test", conf.rf.q3)

# Predict class probabilities for AUC
prob.rf.q3 <- predict(WD.rf.best, WD.test_q3, type = "prob")[, 2]

# Get ROC/AUC performance
res.rf.q3 <- get_roc_perf(prob.rf.q3, WD.test_q3$Class)
print_auc("Improved RF on Q3 Test", res.rf.q3$pred)



# Cross-validation using AUC (ROC) to tune 'mtry'
set.seed(34035958)

ctrl <- trainControl(method = "cv", number = 10, classProbs = TRUE,
                     summaryFunction = twoClassSummary)
```

```
# Rename class for caret (caret expects factor levels to be character)
WD.train_sampled$Class <- factor(ifelse(WD.train_sampled$Class == 1, "Oats", "Other"))

# Train with cross-validation
rf_cv <- train(Class ~ ., data = WD.train_sampled,
               method = "rf",
               metric = "ROC",      # AUC is used for tuning
               trControl = ctrl,
               tuneLength = 5)      # Tries 5 different mtry values

# View best parameters of mtry
print(rf_cv$bestTune)

# ------------------------------------------------
# Q12. ANN
# ------------------------------------------------

# Convert Class from factor to numeric
WD$Class <- as.numeric(as.character(WD$Class))

# Sample 500 from each class to balance
set.seed(34035958)
class0_idx <- which(WD$Class == 0)
class1_idx <- which(WD$Class == 1)

sample0 <- sample(class0_idx, 380)
sample1 <- sample(class1_idx, 380)

balanced_idx <- c(sample0, sample1)
WD_balanced <- WD[balanced_idx, ]

# Shuffle the balanced dataset
WD_balanced <- WD_balanced[sample(1:nrow(WD_balanced)), ]

# 80/20 train-test split
set.seed(34035958)
train_idx <- sample(1:nrow(WD_balanced), 0.8 * nrow(WD_balanced))
WD.train_ann <- WD_balanced[train_idx, ]
WD.test_ann  <- WD_balanced[-train_idx, ]

# Choose top predictors from Q4
selected_vars <- c("A26", "A09", "A25")

# Ensure selected predictors are numeric
WD.train_ann[, selected_vars] <- lapply(WD.train_ann[, selected_vars], as.numeric)
WD.test_ann[, selected_vars]  <- lapply(WD.test_ann[, selected_vars], as.numeric)

# Train neural network
WD.nn <- neuralnet(Class == 1 ~ A26 + A09 + A25,
                   data = WD.train_ann,
                   hidden = 3,
```

```
            linear.output = FALSE)

# Ensure the same predictors are numeric in the Q3 test set
WD.test[, selected_vars] <- lapply(WD.test[, selected_vars], as.numeric)

# Predict using the neural network on original test set from Q3
WD.nn.pred_q3 <- compute(WD.nn, WD.test[, selected_vars])
prob_q3 <- WD.nn.pred_q3$net.result
pred_q3 <- ifelse(prob_q3 > 0.5, 1, 0)

# Confusion matrix using original test set (unbalanced)
conf.mat_q3 <- table(observed = WD.test$Class, predicted = pred_q3)
print(conf.mat_q3)

# Evaluate performance on original test set
evaluate_confusion_metrics("Neural Network", conf.mat_q3)

# AUC
pred_nn <- ROCR::prediction(prob_q3, WD.test$Class)
print_auc("Neural Network", pred_nn)


# ------------------------------------------------
# Q13. SVM
# ------------------------------------------------

# Rebalance the data (specific to SVM)
class0_idx_svm <- which(WD$Class == 0)
class1_idx_svm <- which(WD$Class == 1)

set.seed(34035958)
sample0_svm <- sample(class0_idx_svm, 380)
sample1_svm <- sample(class1_idx_svm, 380)
balanced_idx_svm <- c(sample0_svm, sample1_svm)
WD_svm_balanced <- WD[balanced_idx_svm, ]

# Remove least important predictors for SVM
cols_to_remove <- c("A16", "A19")
WD_svm_balanced <- WD_svm_balanced[, !(names(WD_svm_balanced) %in%
cols_to_remove)]
WD_test_q3_svm <- WD.test[, !(names(WD.test) %in% cols_to_remove)]

# Convert target to factor for classification
WD_svm_balanced$Class <- as.factor(WD_svm_balanced$Class)

# Train SVM model
svm_model <- svm(Class ~ ., data = WD_svm_balanced, kernel = "radial", probability = TRUE)

# Predict on Q3 test set
svm_pred_class <- predict(svm_model, WD_test_q3_svm)
```

```
svm_pred_prob <- attr(predict(svm_model, WD_test_q3_svm, probability = TRUE),
"probabilities")[, "1"]

# Confusion matrix
conf.svm <- table(Predicted = svm_pred_class, Actual = WD_test_q3_svm$Class)
cat("\n# SVM Confusion Matrix\n")
print(conf.svm)

# Evaluate performance
evaluate_confusion_metrics("SVM", conf.svm)

# AUC
svm_roc <- ROCR::prediction(svm_pred_prob, WD_test_q3_svm$Class)
print_auc("SVM", svm_roc)
```