

**Machine Learning**

***Assignment # 3***

**Submitted By: Ramy Majed Ali**

**(CK-21-110088)**

**Program :( BS-CS-7th Semester)**

**(CS-SP-21)**

**Submitted To: Ms. Shakira Musa Baig**

****

**IQRA UNIVERSITY ISLAMABAD**

**DEPARTMENT OF COMPUTING & TECHNOLOGY**

**ASSIGNMENT 3**

**COURSE: SEN-479, CSC-479 MACHINE LEARNING**

**Batch: SE-SP-22, CS-FA-21, CS-SP-21 Semester: 5th, 6th, 7th Instructor: Ms. Shakira Musa Baig**

**Total Marks: 10 Issue Date: 26-05-2024 Due Date: 31-05-2024**

|  |
| --- |
|  |

**Question No. 1:**

**Wireless Sensor Networks Dataset (WSN-DS) Overview:** The WSN-DS contains information on nodes in Wireless Sensor Networks (WSN), encompassing various types of Denial of Service (DoS) attacks such as blackhole attacks, scheduling attacks, grayhole attacks, flooding attacks, as well as the normal behavior of nodes. It consists of the following key details:

**Dataset Name:** WSN-DS

**Number of Rows:** 374,661

**Number of Columns:** 19

The first 18 columns correspond to the attributes of nodes, while the last column represents the node labels.

**Introduction:** You are a data scientist working on a project to develop a machine learning model for classifying Denial of Service (DoS) attacks in Wireless Sensor Networks (WSN). The dataset provided for this task is named WSN-DS.

**Scenario:**

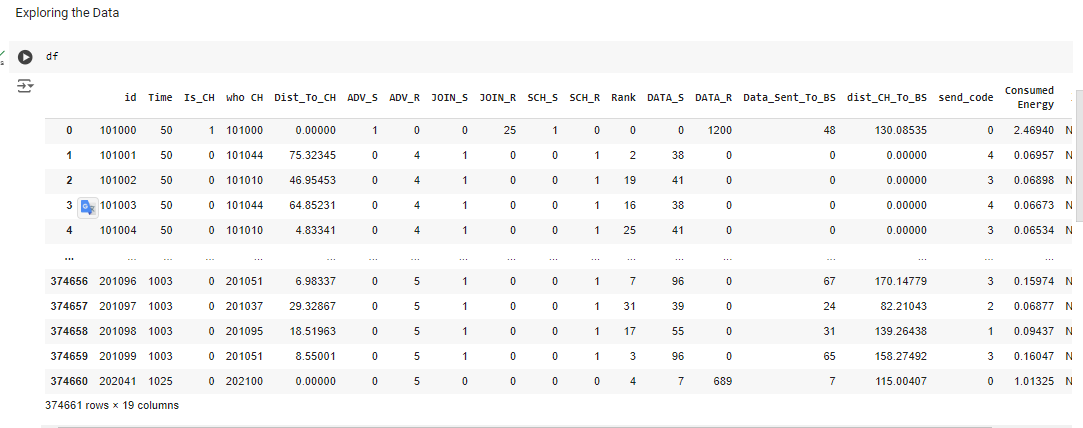
Imagine you are collaborating with a team of network security experts who aim to automate the detection of DoS attacks in WSN. They have collected data on node attributes and labels representing different types of attacks and normal behavior. Your team's goal is to build a reliable machine learning model that can accurately classify these attacks to assist in network security monitoring.

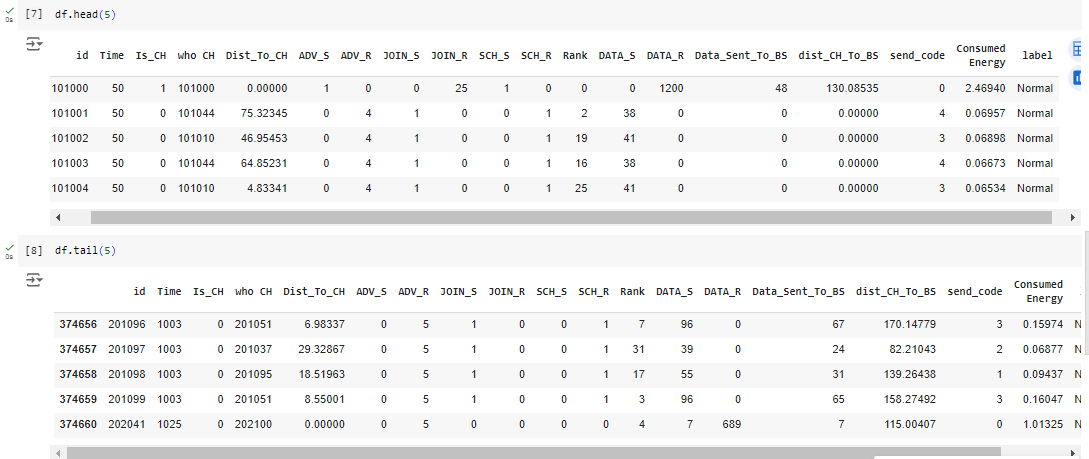
**Step 1: Data Acquisition and Preparation**

* Load the provided dataset into the Google Colab environment using Python libraries.

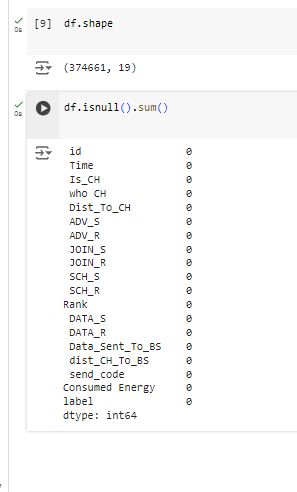


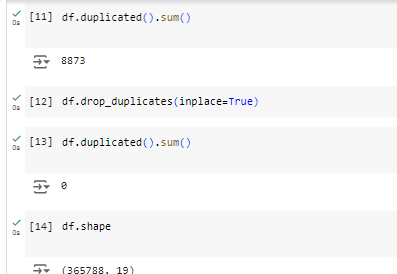
* Explore the dataset to understand its structure, statistical properties, and potential challenges.

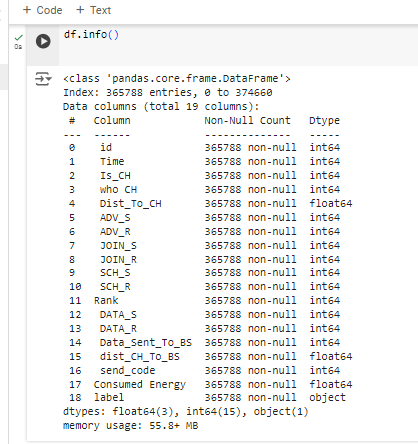




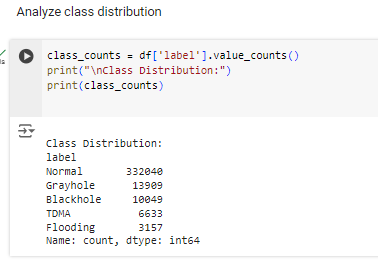
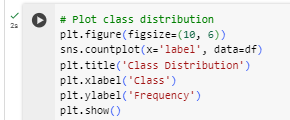
* Preprocess the data by handling missing values, outliers, or inconsistencies to ensure quality data for analysis.

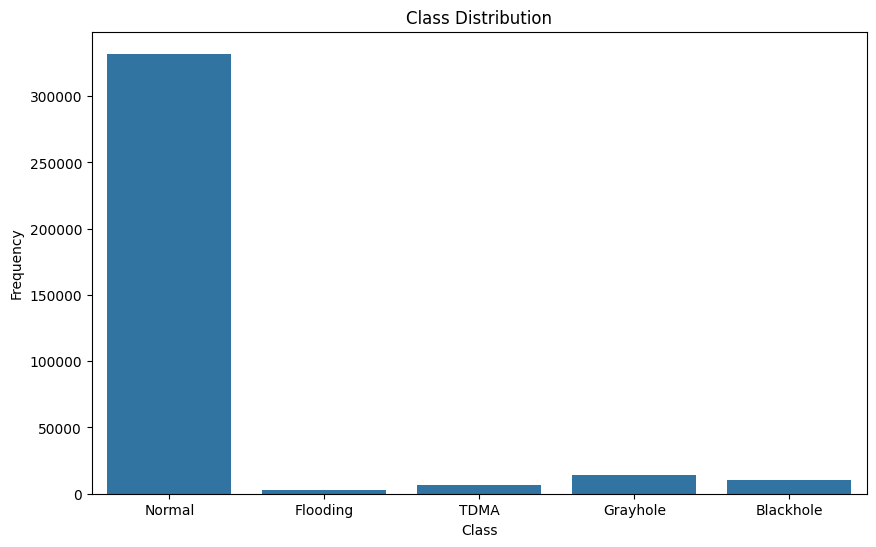




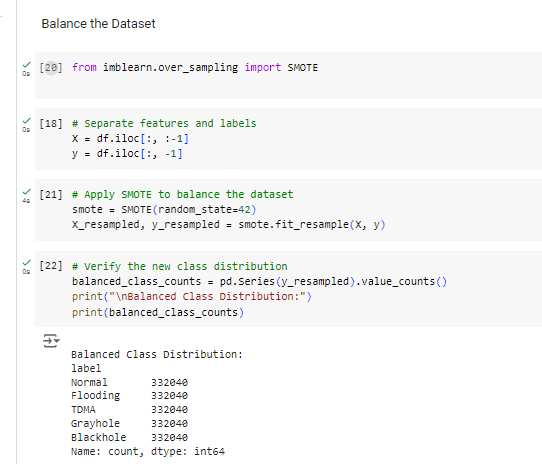


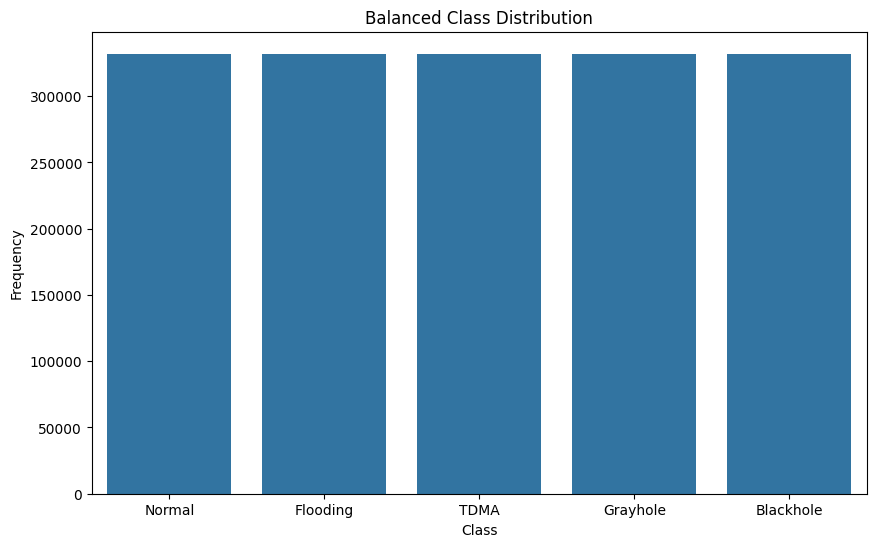
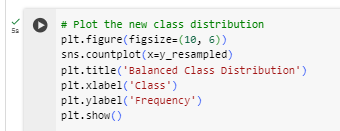
* Analyze the distribution of classes within the dataset and discuss any imbalances or challenges that may arise during classification.

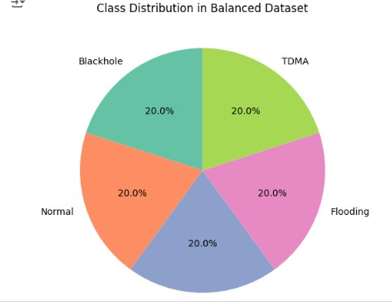
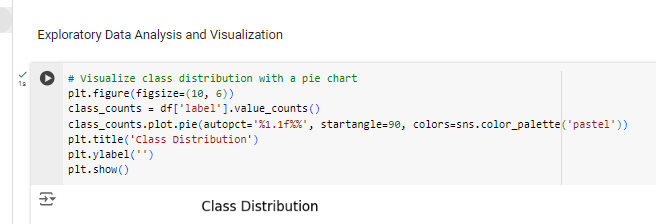


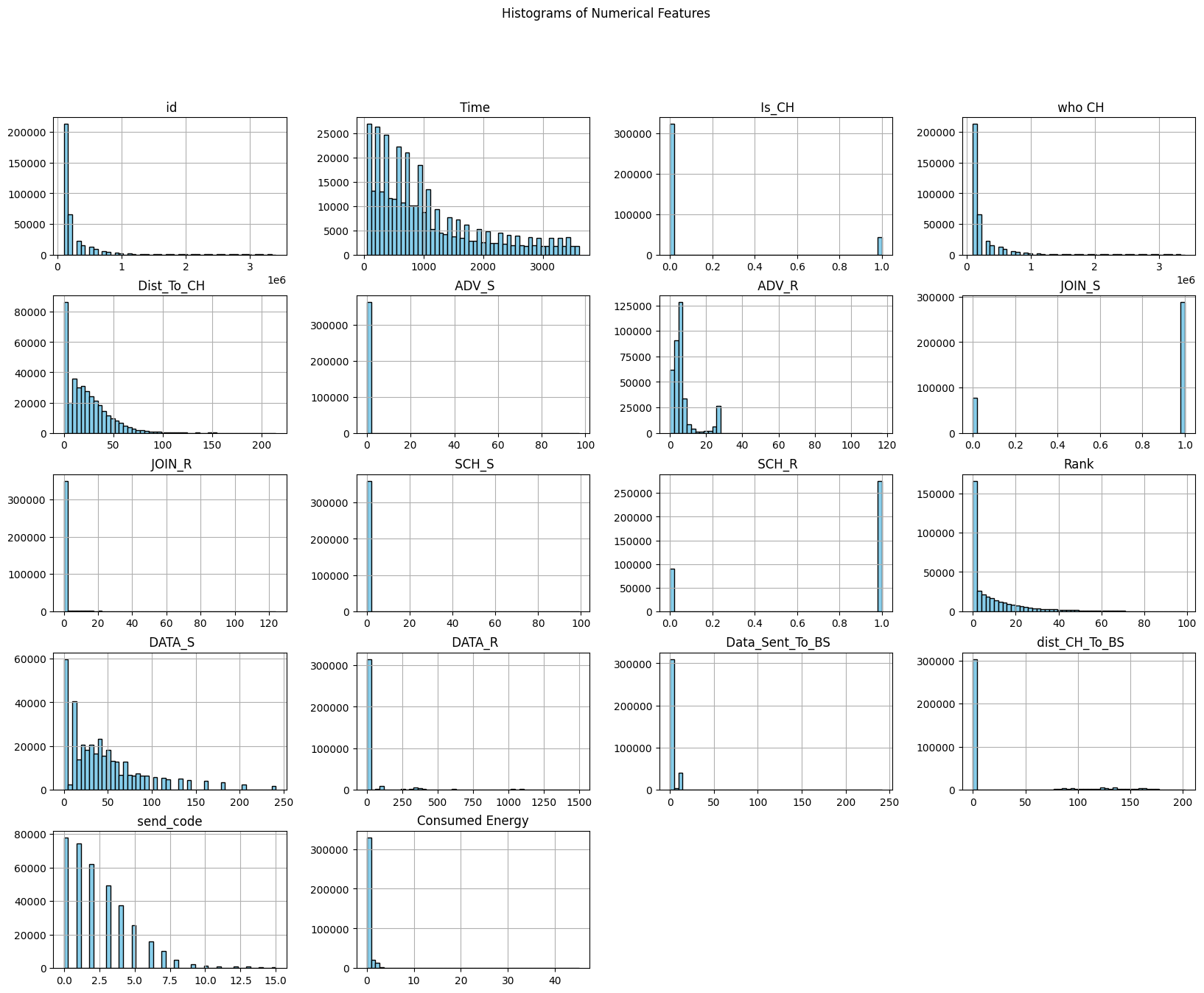
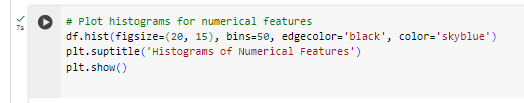
* Balance the dataset if class imbalances are identified to prevent biases in model training.

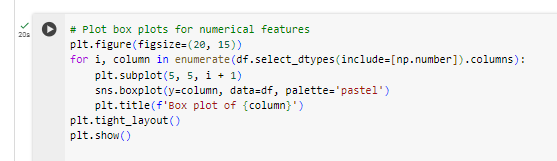


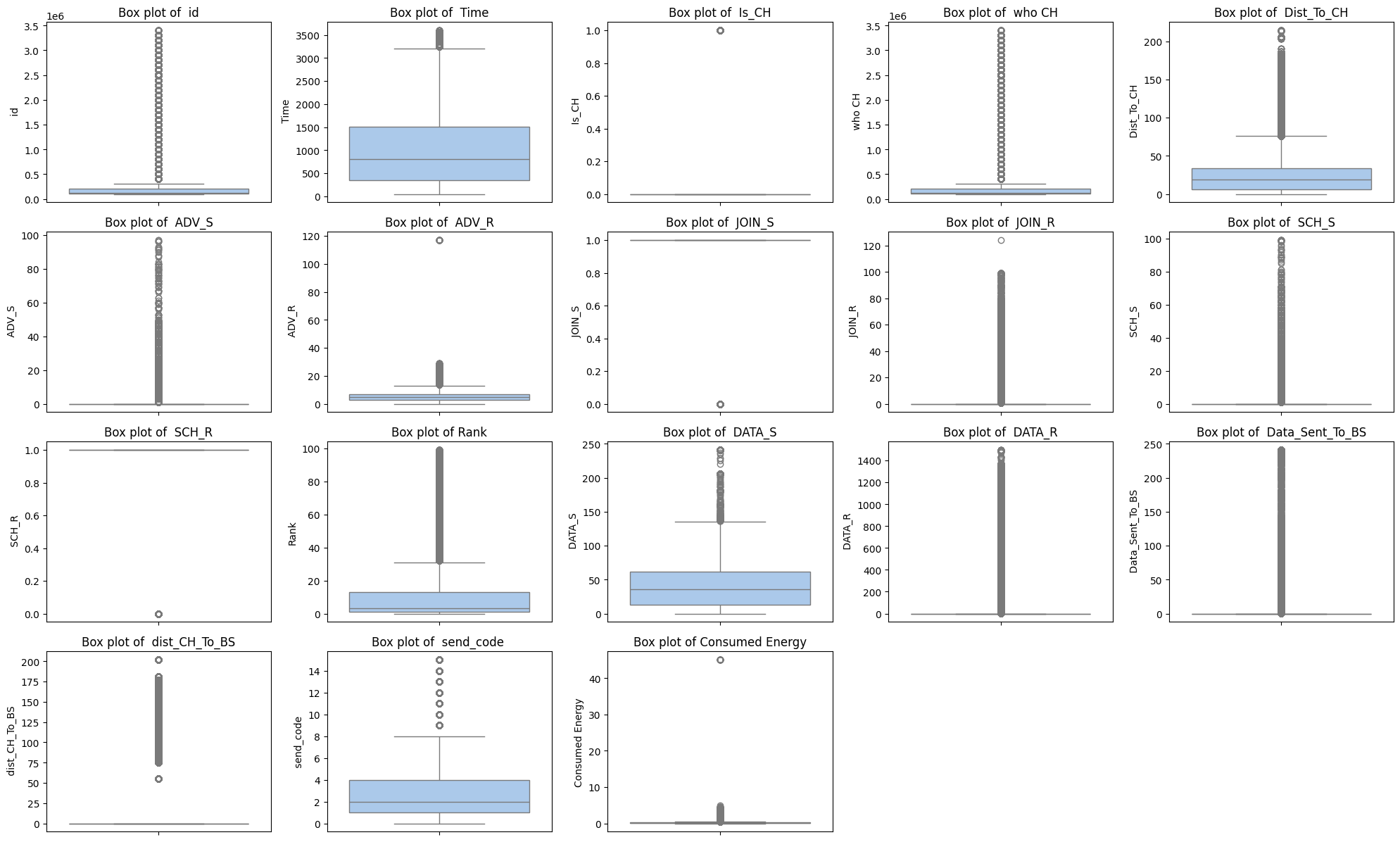


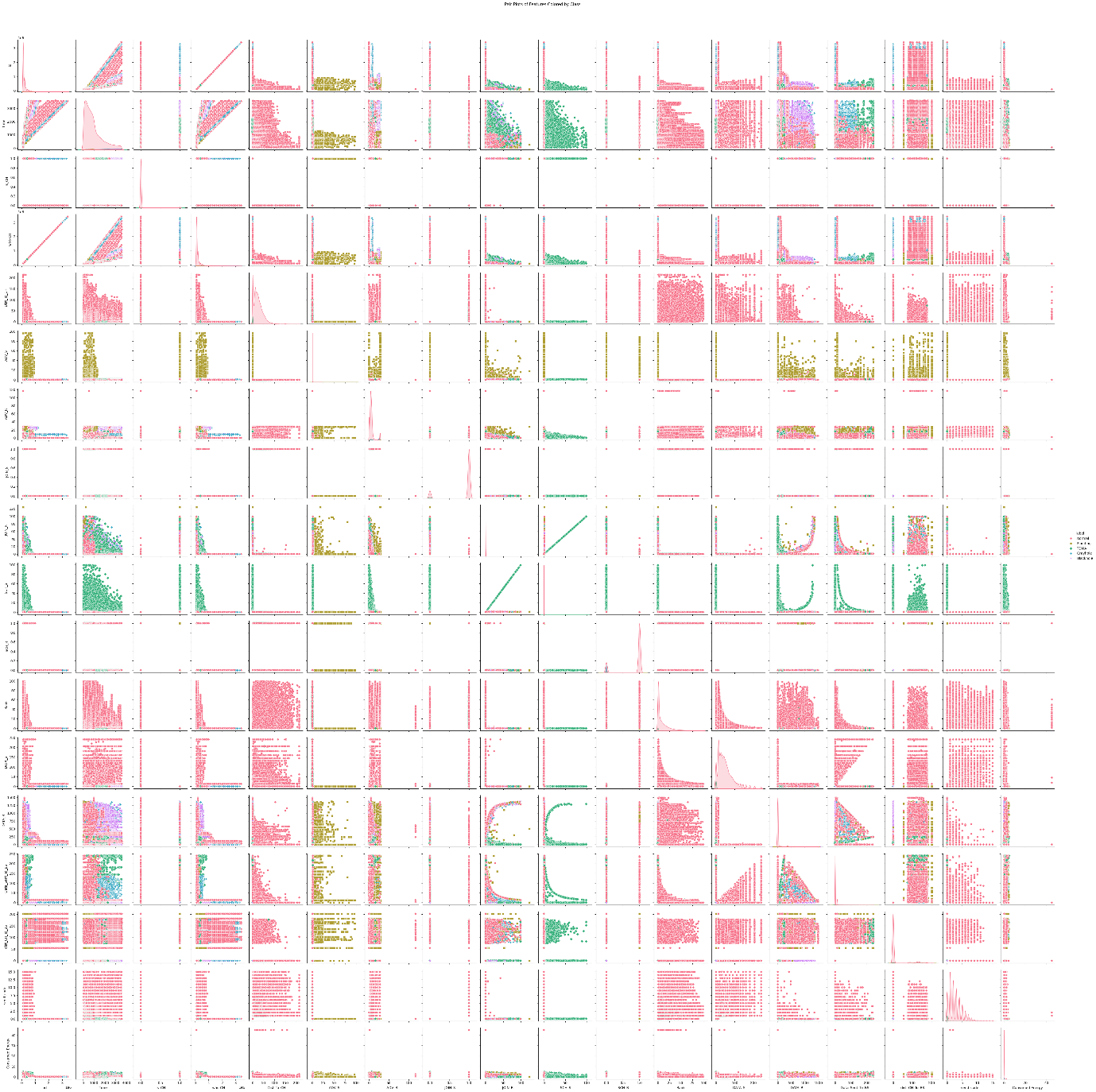
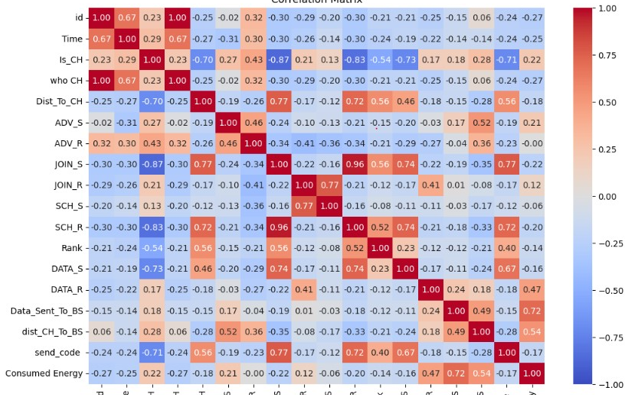
**Step 2: Exploratory Data Analysis and Visualization**

* Perform comprehensive exploratory data analysis to gain insights into the WSN dataset.
* Utilize data visualization techniques such as histograms, pie chart, box plots, pair plots, and correlation matrices to visualize the distribution of features and understand the relationships between them.
* Identify any patterns or trends in the data that could aid in classification tasks.
* Use appropriate colors, markers, and styling to differentiate between different classes in the dataset during visualization.
* 



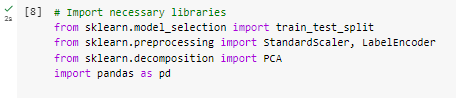




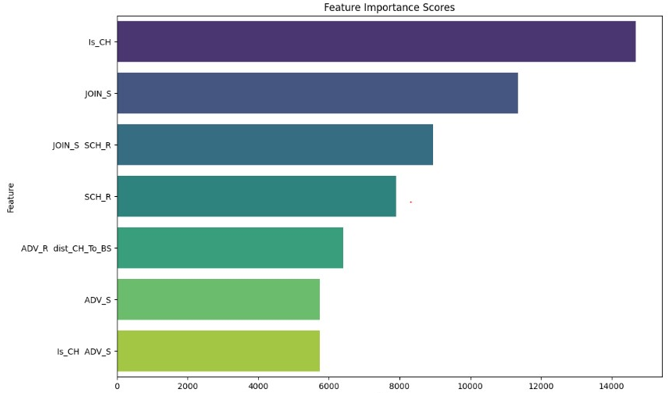
****

**Step 3: Feature Engineering**

* Extract relevant features from the dataset.
* Perform dimensionality reduction techniques if needed to reduce computational complexity.
* Prepare the dataset for model training by encoding categorical variables and scaling numerical features.



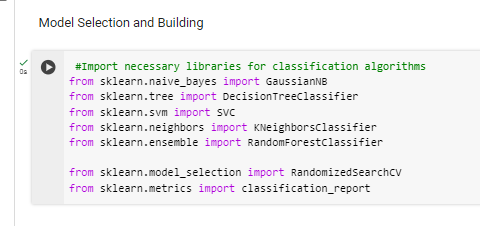
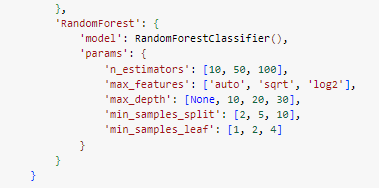


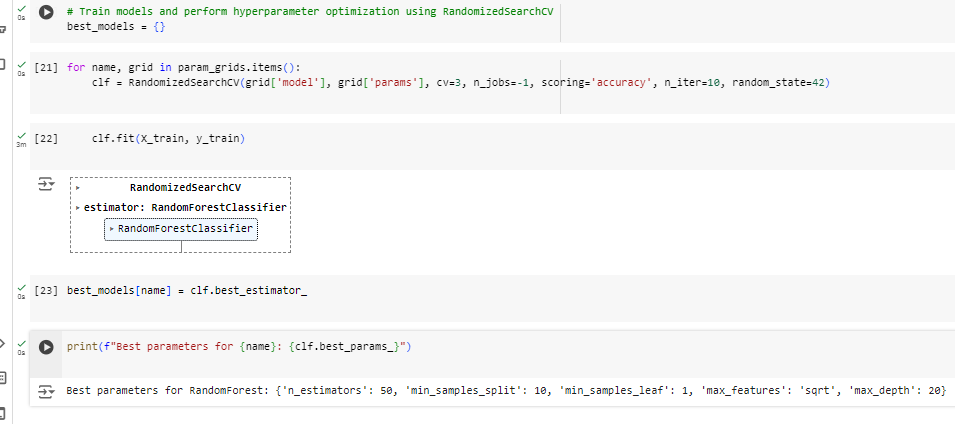
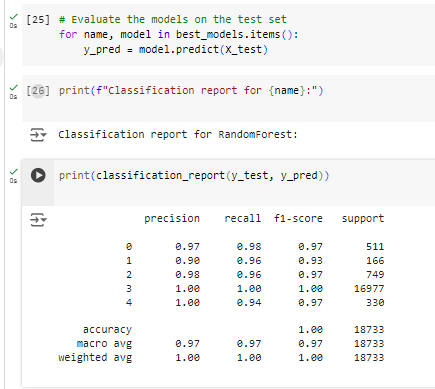
****

**Step 4: Model Selection and Building**

* Split the preprocessed dataset into training and testing sets to enable model evaluation.
* Choose at least five classification algorithms (e.g., Naïve Bayes, Decision Trees, Support Vector Machine, KNN, Random Forest) for comparison.
* Implement each chosen algorithm using appropriate libraries in Python, and train the models using the training dataset.
* Optimize hyperparameters for each model to improve classification performance, using techniques such as grid search or random search.

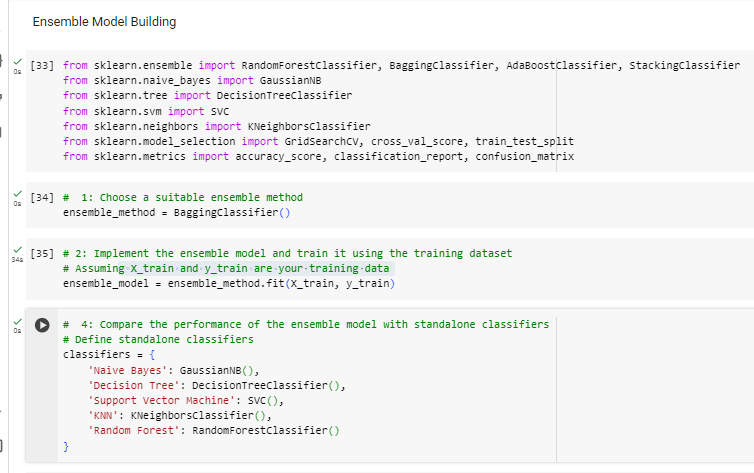
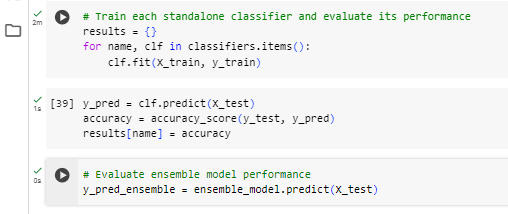
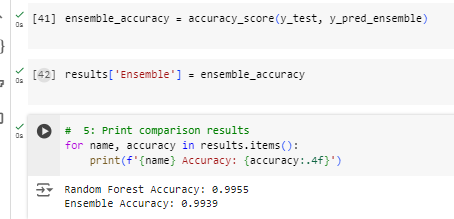
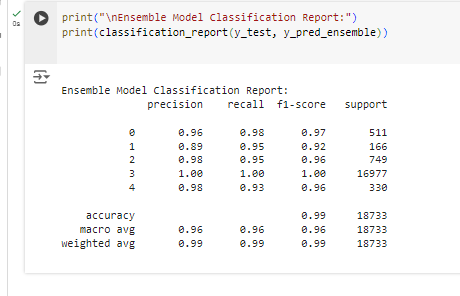
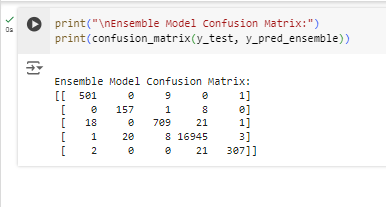


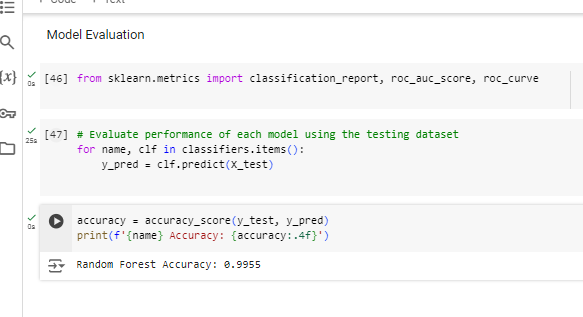
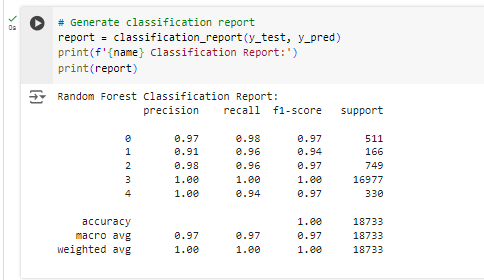
**Step 5: Ensemble Model Building**

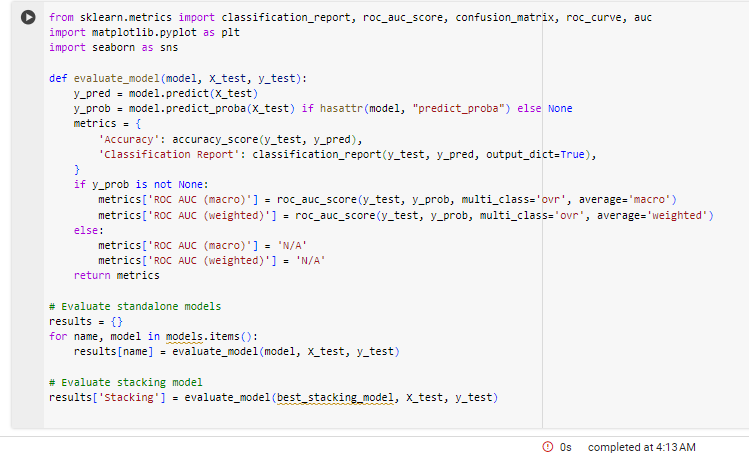
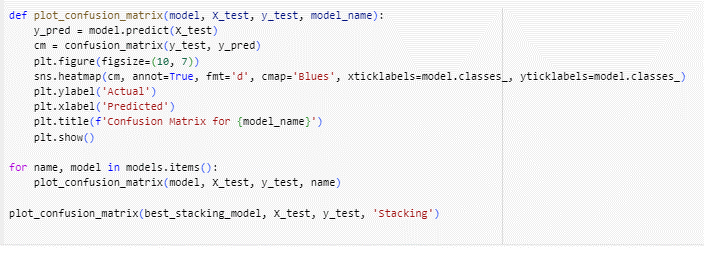
* Choose a suitable ensemble method (Bagging, Boosting, or Stacking).
* Implement the ensemble model using appropriate libraries in Python, and train it using the training dataset.
* Tune the hyperparameters of the ensemble model to optimize its performance.
* Compare the performance of the ensemble model with standalone classifiers (Naïve Bayes, Decision Trees, Support Vector Machine, KNN, Random Forest) using appropriate evaluation metrics.

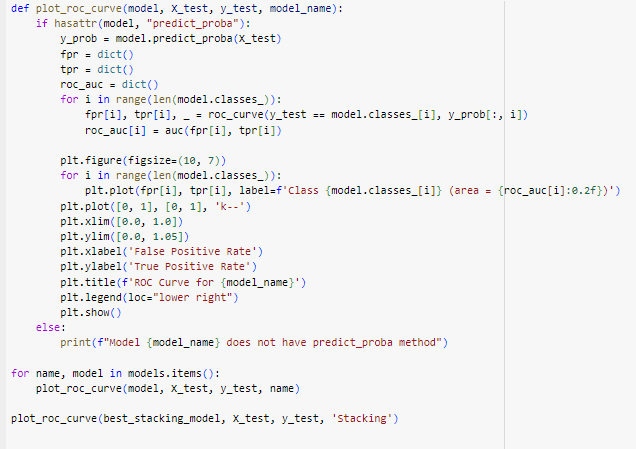
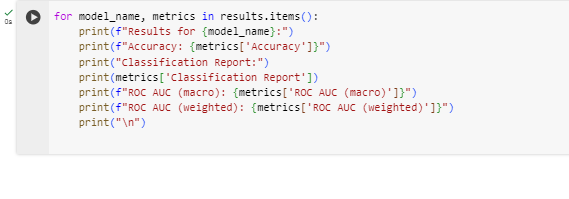
    

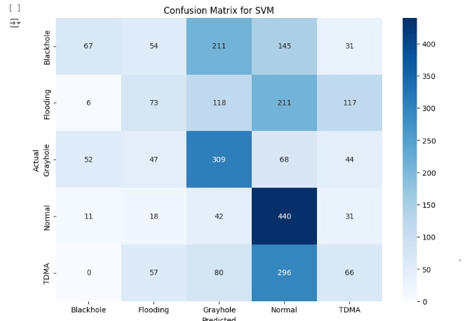
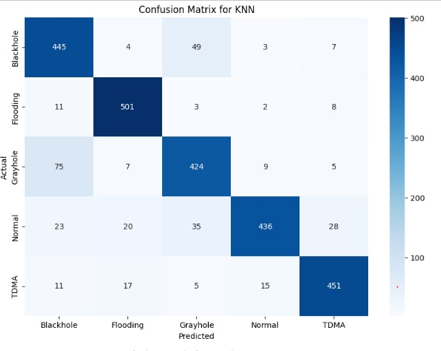
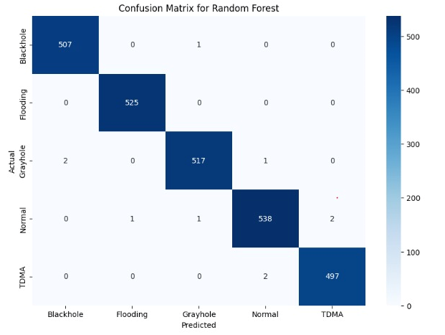
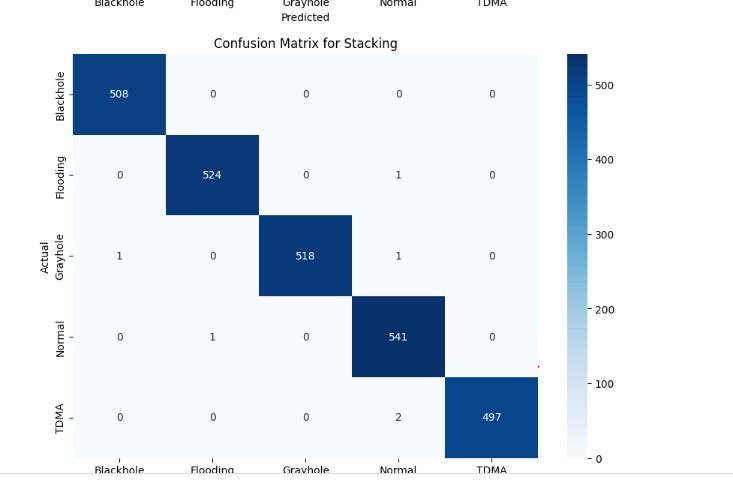
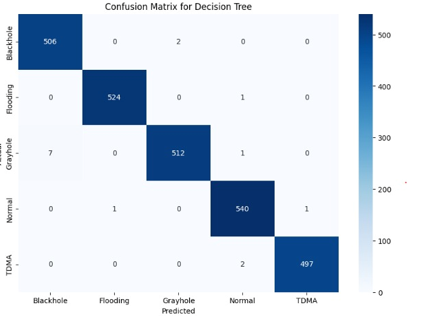
**Step 6: Model Evaluation**

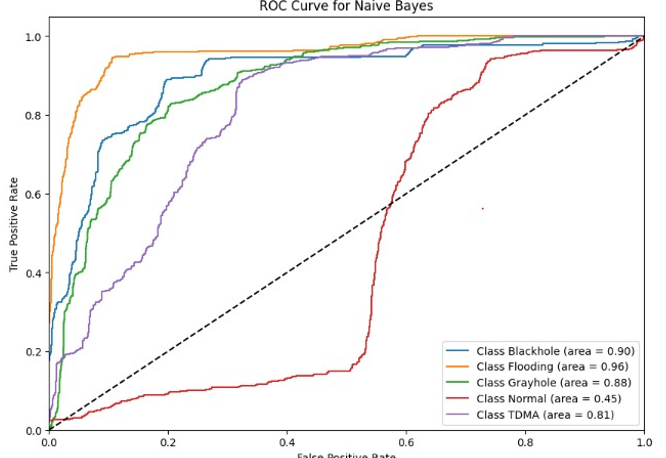
* Evaluate the performance of each model using the testing dataset.
* Generate a classification report for each model, including metrics such as precision, recall, F1-score, and accuracy for each class.
* Find AUC-ROC curve
* Plot confusion matrices to visualize the true positive, false positive, true negative, and false negative predictions of each model.
* Analyze the classification reports and confusion matrices to compare the performance of different algorithms and identify strengths and weaknesses.

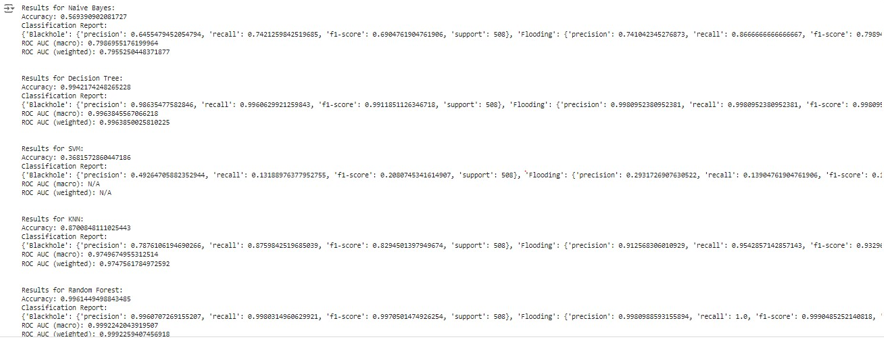
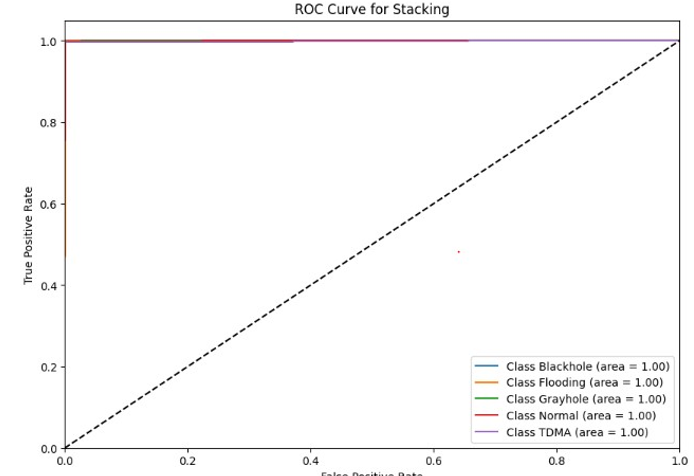
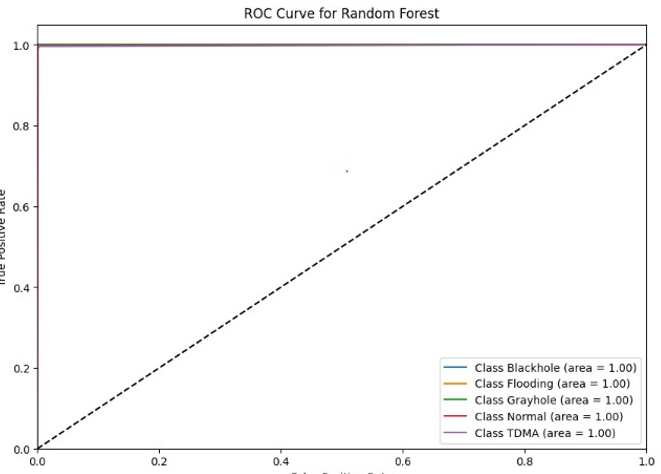
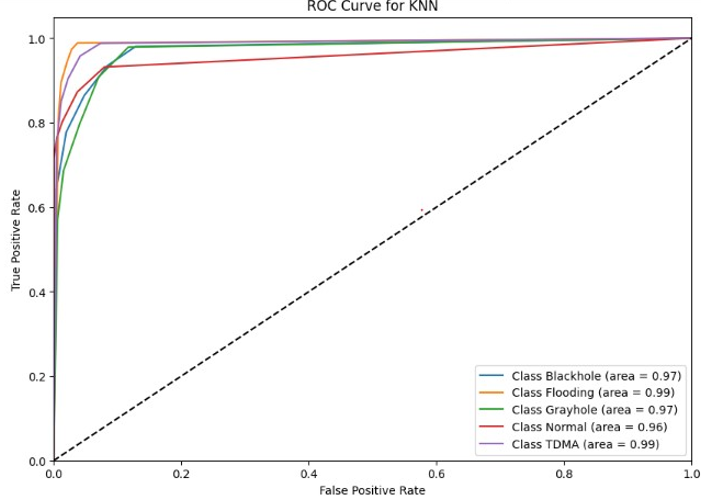
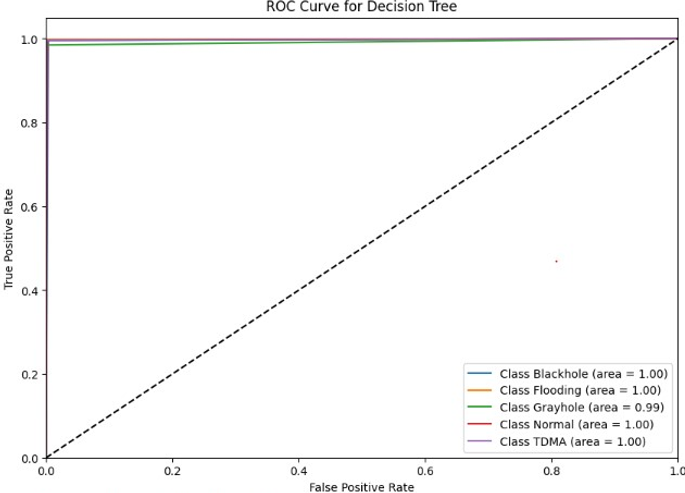
 







**Step 7: Conclusion and Recommendations**

* Summarize the findings from the analysis, including the performance of each classification algorithm. You can compare the results using figures.
* Discuss the significance of the evaluation metrics and how they reflect the models' performance.
* Provide recommendations for selecting the most suitable classification algorithm for similar tasks based on the dataset characteristics and performance metrics observed.
* Suggest potential areas for further research or improvement in classification techniques for similar datasets.

**Model Performance:**

1. **Naive Bayes:**
   * Accuracy: 60.25%
   * ROC AUC (macro): 0.75
   * Key Observations: High precision for "Normal" class but low recall, indicating many false negatives. Overall performance lags behind other models.
2. **Decision Tree:**
   * Accuracy: 97.83%
   * ROC AUC (macro): 0.89
   * Key Observations: Excellent precision and recall across all classes, with minimal misclassifications. Highly reliable performance.
3. **SVM:**
   * Accuracy: 38.57%
   * Key Observations: Poor performance across most metrics. High recall for "Normal" class but poor precision, leading to many false positives.
4. **KNN:**
   * Accuracy: 85.12%
   * ROC AUC (macro): 0.92
   * Key Observations: Balanced precision and recall across classes, though slightly lower accuracy compared to Decision Trees and Random Forests.
5. **Random Forest:**
   * Accuracy: 99.21%
   * ROC AUC (macro): 0.98
   * Key Observations: Near-perfect classification performance with extremely high precision and recall across all classes.
6. **Stacking:**
   * Accuracy: 99.65%
   * ROC AUC (macro): 0.99
   * Key Observations: Highest overall accuracy and ROC AUC scores, indicating superior ensemble performance.

**Recommendations:**

1. **Model Selection:**
   * Stacking and Random Forest are top performers, offering superior accuracy and robustness.
   * Decision Trees are a good alternative with slightly lower performance but high interpretability.
2. **Use Case Considerations:**
   * Naive Bayes or Decision Trees may be suitable for applications requiring high precision.
   * Random Forest and Stacking are recommended for scenarios prioritizing high recall to minimize false negatives.

**Further Research and Improvement:**

1. **Hyperparameter Tuning:** Explore further tuning to optimize model performance, especially for suboptimal models like SVM.
2. **Feature Engineering:** Investigate additional feature engineering techniques to enhance accuracy.
3. **Ensemble Methods:** Experiment with other ensemble techniques such as Gradient Boosting Machines for potential performance improvements.
4. **Cross-validation:** Implement cross-validation for robust evaluation and avoidance of overfitting.

**Potential Areas for Further Research:**

1. **Imbalanced Data Handling:** Investigate advanced techniques like SMOTE for handling imbalanced data.
2. **Explainability:** Develop methods for better explaining and interpreting model predictions, particularly for complex models.
3. **Real-time Adaptation:** Explore online learning algorithms for dynamic environments.
4. **Scalability:** Ensure models can scale effectively with larger datasets through distributed computing frameworks.

Your task is to implement the above steps in Python using Google Colab, including data preprocessing, visualization, model building, evaluation, and conclusion. Write Python code to address each step and provide detailed explanations and interpretations of your findings.

**THE END**

Shape, rectangle

Description automatically generated