**LAB 04**

**Machine learning source code understanding**

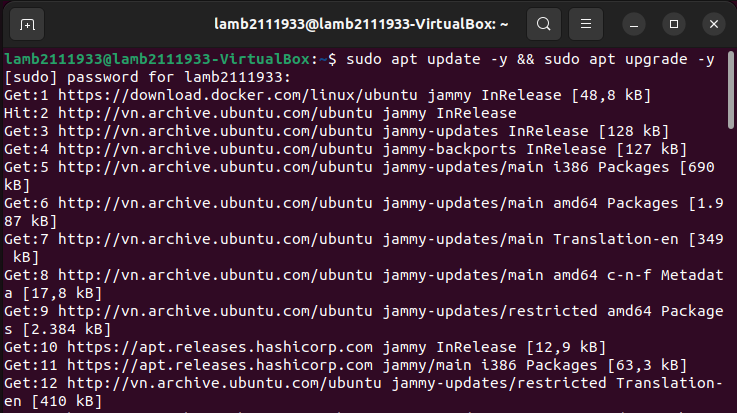
| Class | CT201H [M01-M04] |
| --- | --- |
| Student ID | B2111933 |
| Name | Truong Dang Truc Lam |
| Email address | lamb2111933@student.ctu.edu.vn |
| Class | CT201H [M01-M04] |
| Browser | Chrome |

1. **Test environment setting**

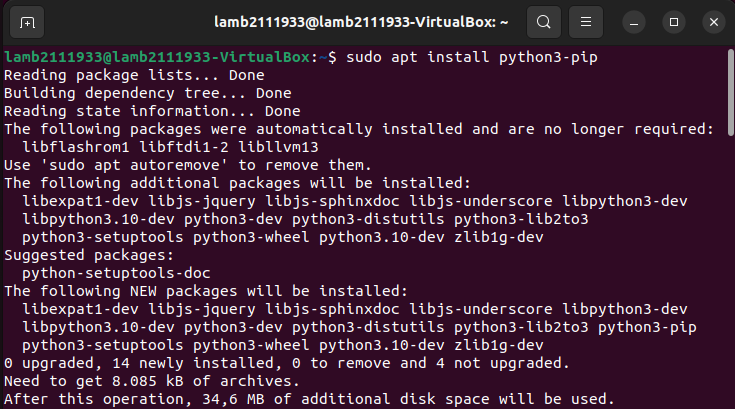
|  | Attacker | Target |
| --- | --- | --- |
| OS | Ubuntu |  |
| IP address | 192.168.1.11 |  |
| Attacking type |  |  |
| Attacking program |  |  |
| Detecting program |  |  |
| Blocking program |  |  |
| Analyzing program |  |  |

1. **Exercise as far as you do**

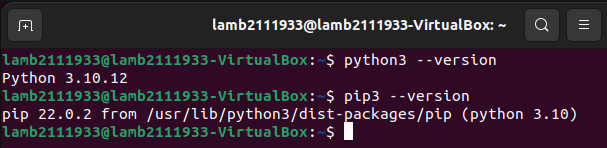
[Step1] Install Machine learning libraries Scikit-learn in Python



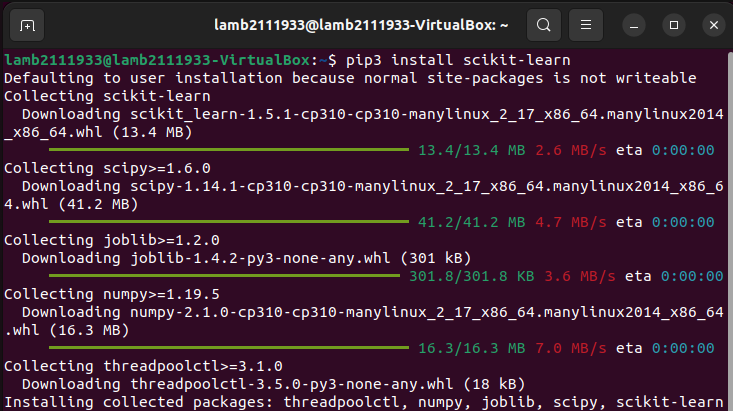
Update packages

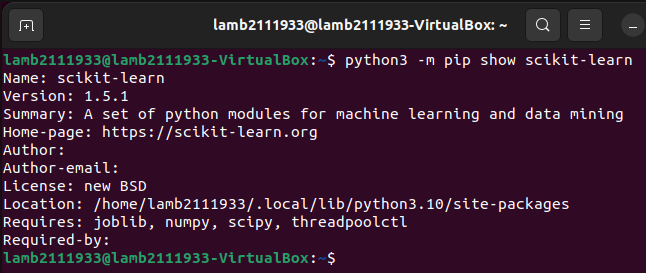


Install pip packages



Check if python has installed

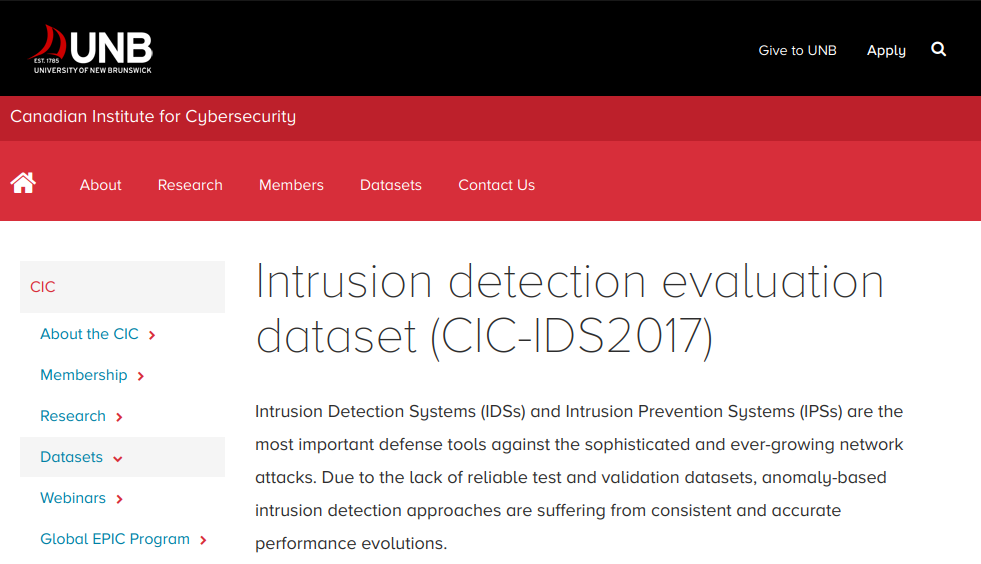




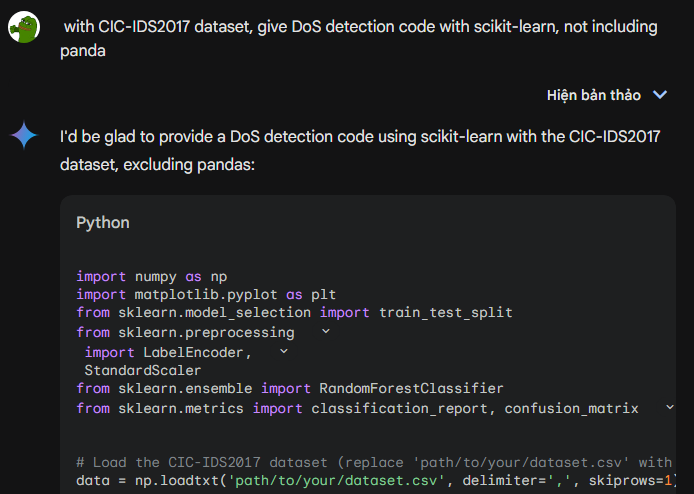
Install Scikit-learn successfully

[Step2] Search Machine learning source code example and clone (write reference site)

Test 1 : Code example of DoS detection using Scikit-learn (Tuesday)

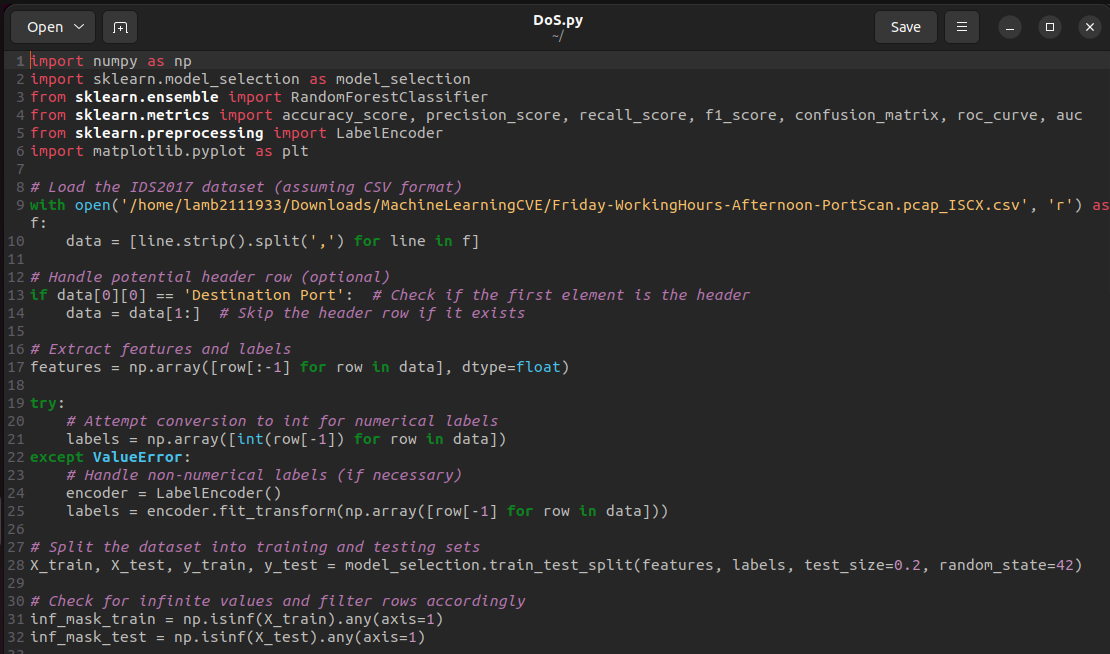


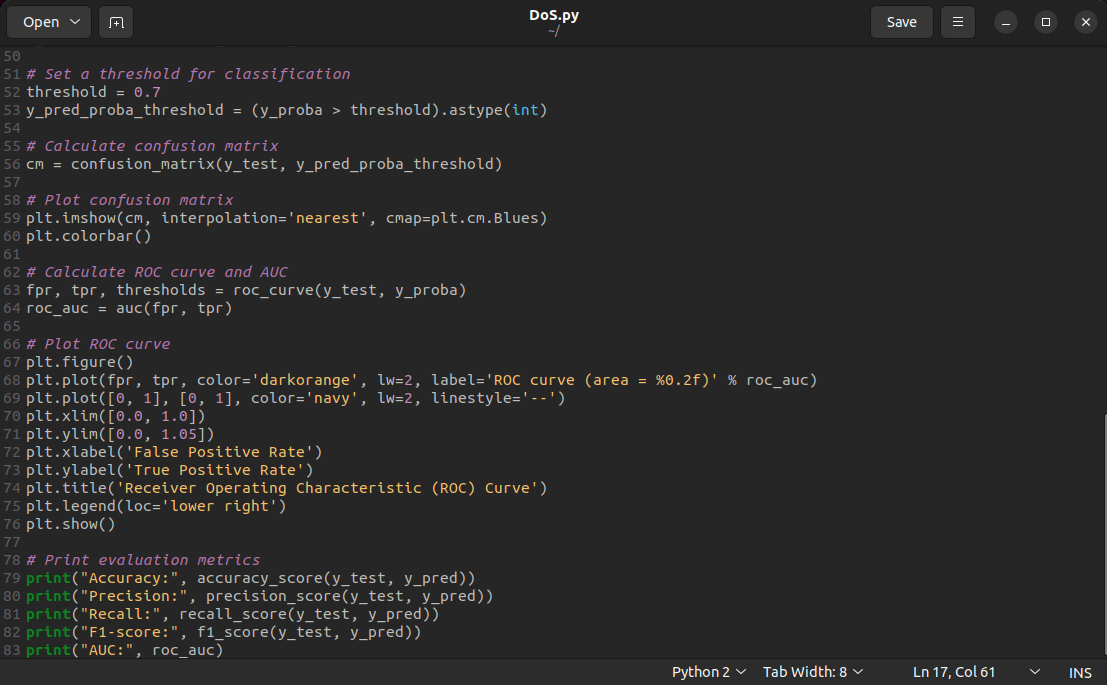
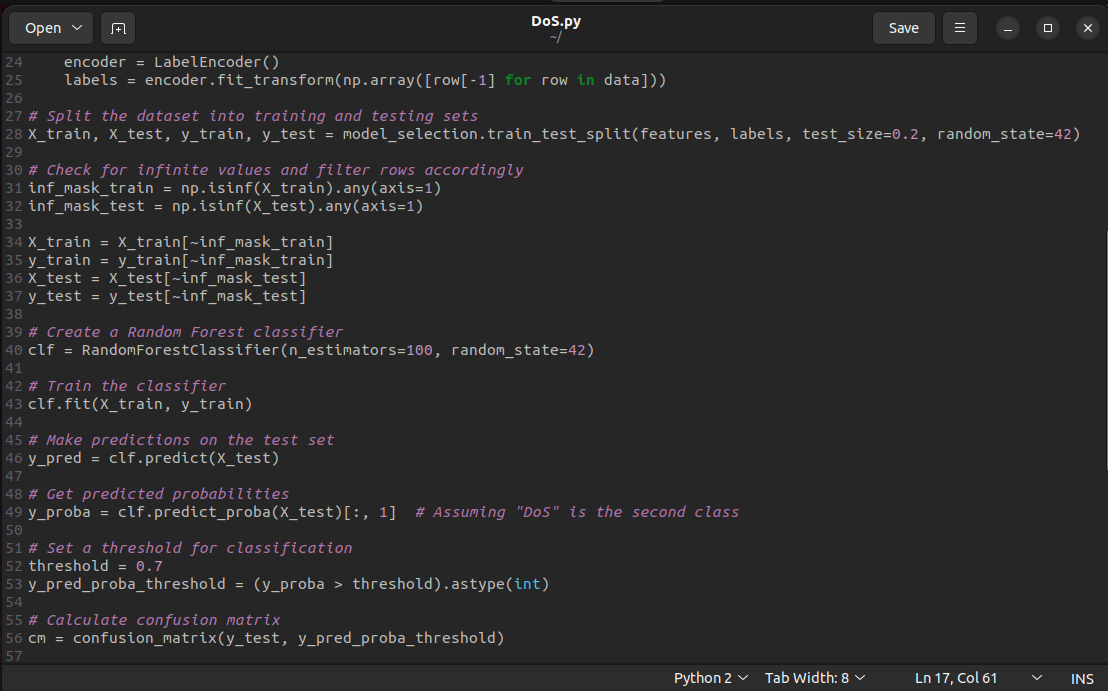
Download CIC-IDS2017 dataset



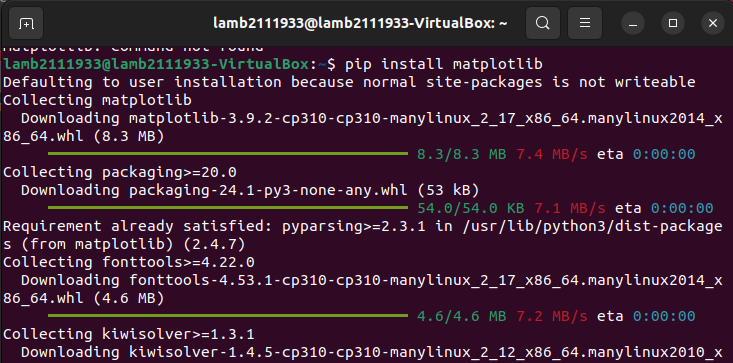
Ask Gemini for code

However, the initial code will not fit the dataset, it must be modified several times to working properly. This is the final code:





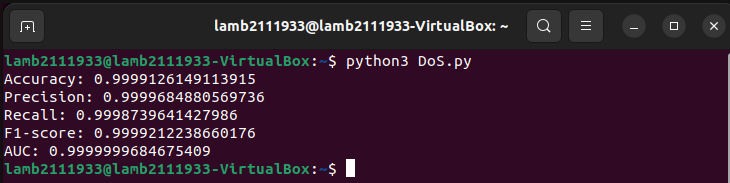
Ask Gemini for code



Additionally install matplotlib for visualization

Test 2 : Code example of SYN Flooding detecting using Scikit-learn (Wednesday)

[Step3] Execute the Scikit-learn code



Compile and run, it’s working properly

[Step4] Survey what kind of Scikit-learn error we can encounter (from reference site)

From Gemini:

Scikit-learn, a powerful machine learning library in Python, is prone to various errors. Understanding these common pitfalls can help you troubleshoot and improve your machine learning models. Here are some of the most frequently encountered errors:

Data-Related Errors

* Shape Mismatch: Ensuring that your data has the correct dimensions (number of rows and columns) is crucial. Mismatched shapes often lead to errors like ValueError: Incompatible shapes.
* Missing Values: Missing data can cause issues in many algorithms. Handle missing values appropriately using techniques like imputation or deletion.
* Data Type Mismatch: Check that your data is in the expected format (e.g., numerical for regression, categorical for classification). Incorrect data types can lead to errors.
* Feature Scaling: If your features have significantly different scales, consider scaling them using techniques like normalization or standardization. This can improve model performance.

Model-Related Errors

* Overfitting: When a model performs well on the training data but poorly on unseen data, it's likely overfitting. Regularization techniques or gathering more data can help address this.
* Underfitting: A model that performs poorly on both training and test data is underfitting. Increasing the complexity of the model or providing more relevant features can help.
* Incorrect Model Choice: Selecting an inappropriate model for your problem can lead to suboptimal results. Consider the nature of your data and task when choosing a model.
* Hyperparameter Tuning: Tuning hyperparameters can significantly impact model performance. Explore different combinations of hyperparameters to find the best configuration.

Code-Related Errors

* Syntax Errors: These are basic coding mistakes that can prevent your code from running. Pay attention to syntax rules, indentation, and keyword usage.
* Logic Errors: While your code may run without errors, it might produce incorrect results due to logical flaws. Carefully review your code's logic to ensure it's performing as intended.
* Library Version Compatibility: Ensure that the versions of Scikit-learn and other libraries you're using are compatible. Outdated or incompatible versions can lead to errors.

Other Errors

* Memory Issues: Large datasets can consume a significant amount of memory. Consider using techniques like out-of-core learning or memory-efficient algorithms to address memory constraints.
* Random State: When dealing with random processes, setting a random state can help ensure reproducibility. This can be especially useful for debugging and comparing results.
* Installation Issues: If you encounter installation problems, double-check your installation steps and dependencies. Consult the official documentation for guidance.

By understanding these common errors and their causes, you can effectively troubleshoot and improve your machine learning projects using Scikit-learn.

[Step5] Explain your code

My provided code implements a Random Forest classifier for intrusion detection using the IDS2017 dataset (assuming it's in CSV format). Here's its breakdown.

1. Imports:

* numpy: Used for numerical operations on data.
* sklearn.model\_selection: Provides functions for splitting data into training and testing sets.
* sklearn.ensemble: Contains the RandomForestClassifier class.
* sklearn.metrics: Offers functions for evaluating model performance.
* sklearn.preprocessing: Provides tools for data preprocessing (optional here).
* matplotlib.pyplot: Used for plotting confusion matrix and ROC curve.

2. Load Dataset:

* Opens the CSV file /home/lamb2111933/Downloads/MachineLearningCVE/Friday-WorkingHours-Afternoon-PortScan.pcap\_ISCX.csv.
* Reads each line, splits it by commas, and stores it in a list called data.
* Checks for a header row by looking at the first element. If it's "Destination Port", removes the header (optional).
* Extracts features (all columns except the last) and labels (last column) from the data.
* Attempts to convert labels to integers (assuming numerical labels).
* If conversion fails (non-numerical labels), uses LabelEncoder to encode them.

3. Data Preprocessing:

* Splits the data into training and testing sets using train\_test\_split with a 20% test size and a random state of 42 (for reproducibility).
* Checks for rows containing infinite values (np.isinf) and filters them out for both training and testing sets.

4. Train Random Forest Classifier:

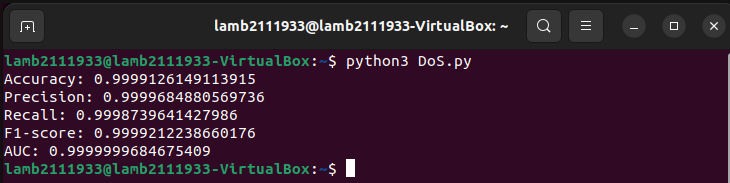
* Creates a RandomForestClassifier object with 100 estimators (decision trees) and a random state of 42.
* Trains the classifier on the training data (X\_train, y\_train).

5. Make Predictions and Evaluate Performance:

* Predicts labels on the testing set using clf.predict(X\_test).
* Gets predicted probabilities for the "DoS" class (assuming the second class) using clf.predict\_proba(X\_test)[:, 1].
* Sets a threshold (0.7) to classify based on probabilities (higher than 0.7 is predicted as DoS).
* Calculates the confusion matrix (cm) to visualize true positives, false positives, etc.
* Plots the confusion matrix using plt.imshow.
* Calculates ROC curve and AUC (Area Under the Curve) using roc\_curve and auc.
* Plots the ROC curve using plt.plot.
* Prints various evaluation metrics: accuracy, precision, recall, F1-score, and AUC.

Overall, this code trains a Random Forest classifier on the IDS2017 dataset, evaluates its performance on unseen data, and visualizes the results using confusion matrix and ROC curve.

Evaluation



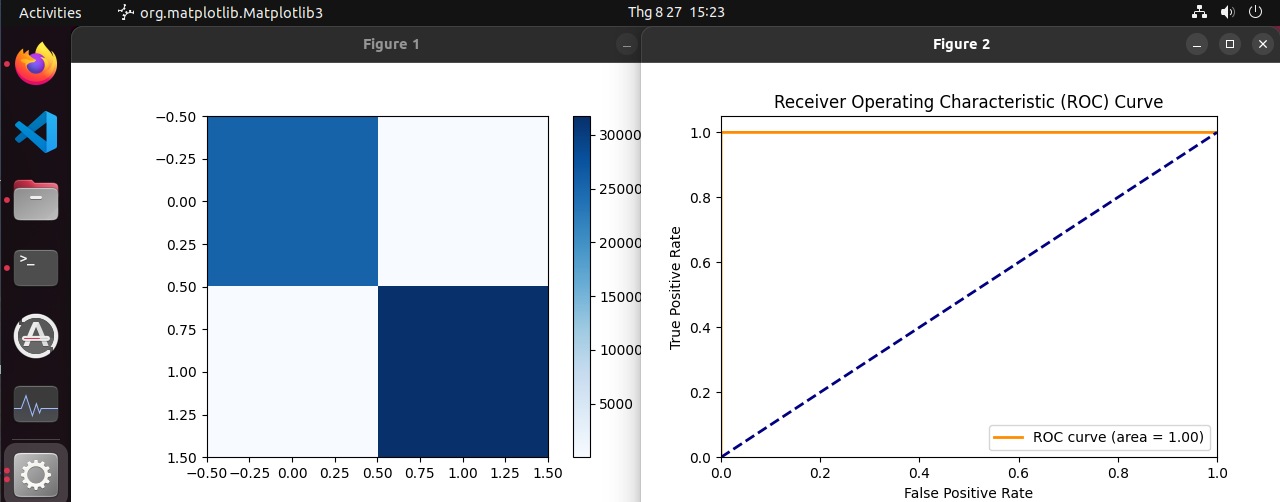
The provided image displays the evaluation metrics for a machine learning model, specifically a binary classifier. Let's break down each metric:

* Accuracy: This measures the overall correctness of the model's predictions. A higher accuracy indicates a better overall performance.
* Precision: This measures the proportion of positive predictions that were actually correct. A high precision indicates that the model is good at identifying positive instances.
* Recall: This measures the proportion of actual positive instances that were correctly identified by the model. A high recall indicates that the model is good at capturing all positive instances.
* F1-score: This is the harmonic mean of precision and recall. It provides a balanced measure of both precision and recall.
* AUC: This stands for Area Under the Curve, often referring to the ROC curve (Receiver Operating Characteristic curve). It measures the model's ability to distinguish between positive and negative classes.

Evaluating the Model's Performance:

* Accuracy: 0.9999126149113915 - This is extremely high, suggesting that the model is making very few incorrect predictions.
* Precision: 0.9999684880569736 - This is also very high, indicating that when the model predicts a positive instance, it's almost always correct.
* Recall: 0.9998739641427986 - This is slightly lower than precision but still very high, suggesting that the model is capturing most of the positive instances.
* F1-score: 0.9999212238660176 - This is a balanced measure of precision and recall, and it's also very high, indicating excellent overall performance.
* AUC: 0.9999999684675409 - This is exceptionally close to 1, which is ideal. It suggests that the model can effectively distinguish between positive and negative instances.

Overall Evaluation: Given these metrics, it's clear that the model is performing exceptionally well. The high accuracy, precision, recall, F1-score, and AUC all indicate that the model is highly accurate, precise, and capable of distinguishing between positive and negative instances.



Visualize in figures