Une image contenant Police, texte, logo, Graphique

Description générée automatiquement

|  |
| --- |
| Année académique 2025-2026 |

**Rapport de projet**



**Détection d’intrusions réseau**

**Mise en œuvre d’un système de détection d’intrusions réseau à l’aide d’Azure Machine Learning**

**Réalisé par :**

**IKEN Zakaria**

**JAOUANI Mohammed**

**Formation :**

Master 2 EEA – Mesure et traitement de l’information

Table des matières

[I. Introduction 3](#_Toc222869843)

[1) Présentation générale de la détection d’intrusions réseau 3](#_Toc222869844)

[a) Définition théorique et académique 3](#_Toc222869845)

[b) Représentation et signification des données réseau en détection d’intrusions 4](#_Toc222869846)

[c) Exemples d’applications de la détection d’intrusions réseau 6](#_Toc222869847)

[d) Lien théorique et complexité en détection d’intrusions réseau 6](#_Toc222869848)

[2) Synergie entre Microsoft Azure et le Machine Learning pour une détection d’intrusion réseau efficace 16](#_Toc222869849)

[a) Azure comme infrastructure cloud sécurisée 16](#_Toc222869850)

[b) Azure Machine Learning comme plateforme d’entraînement 16](#_Toc222869851)

[c) Complémentarité avec les services de sécurité Azure 16](#_Toc222869852)

[d) Apport concret de cette synergie 16](#_Toc222869853)

[II. État de l'Art 17](#_Toc222869854)

[1) Modèles existants de détection d’intrusions réseau 17](#_Toc222869855)

[a) Base et architecture 17](#_Toc222869856)

[III. Pré-Réalisation 18](#_Toc222869857)

[1) Analyse du Modèle de Départ : 18](#_Toc222869858)

[a) Règles de Base de la Détection d’intrusions réseau 18](#_Toc222869859)

[b) Modèle pris en exemple 22](#_Toc222869860)

[c) Structure du programme 23](#_Toc222869861)

[d) Fonctionnalités principales 25](#_Toc222869862)

[IV. Réalisation 27](#_Toc222869863)

[1) Évolution des versions 27](#_Toc222869864)

[a) Mise en place de l’environnement Azure Machine Learning 27](#_Toc222869865)

[b) Création de la machine de calcul (Compute Instance) 28](#_Toc222869866)

[c) Importation et préparation de la base de données 29](#_Toc222869867)

[d) Prétraitement des données 30](#_Toc222869868)

[e) Entraînement du modèle 32](#_Toc222869869)

[f) Matrice de confusion 32](#_Toc222869870)

[g) Simulation de détection en temps réel 33](#_Toc222869871)

[V. Tests et Validation 34](#_Toc222869872)

[1) Procédures de Test : 34](#_Toc222869873)

[a) Description des Scénarios de Test : 34](#_Toc222869874)

[b) Résultat des tests 41](#_Toc222869875)

[2) Conclusion des observations : 41](#_Toc222869876)

[a) Synthèse 41](#_Toc222869877)

[VI. Conclusion 42](#_Toc222869878)

[1) Bilan 42](#_Toc222869879)

[2) Perspectives 43](#_Toc222869880)

[3) Extensions du projet 43](#_Toc222869881)

[VII. Références 44](#_Toc222869882)

[VIII. Annexe 46](#_Toc222869883)

# Introduction

## Présentation générale de la détection d’intrusions réseau

### Définition théorique et académique

La détection d’intrusions réseau (*Network Intrusion Detection*) est un domaine fondamental de la cybersécurité qui vise à identifier toute activité malveillante, suspecte ou non autorisée au sein d’un réseau informatique. Une intrusion correspond à une tentative de compromission de la confidentialité, de l’intégrité ou de la disponibilité des systèmes informatiques, pouvant prendre la forme d’attaques par déni de service (DoS/DDoS), de balayage de ports (*port scanning*), d’attaques par force brute, ou encore d’exploitations de vulnérabilités applicatives.

Les systèmes de détection d’intrusions (*Intrusion Detection Systems – IDS*) sont des mécanismes de surveillance capables d’analyser le trafic réseau afin de détecter ces comportements anormaux. On distingue principalement deux types d’IDS :

* Les IDS basés sur le réseau (NIDS), qui analysent le trafic circulant sur le réseau.
* Les IDS basés sur l’hôte (HIDS), qui surveillent l’activité interne d’un système spécifique.

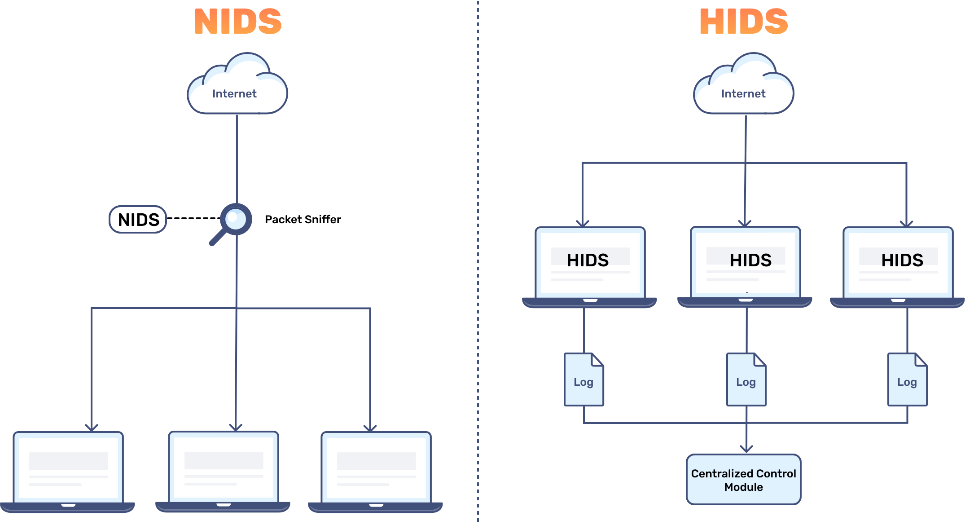


Figure 1 : Architecture des systèmes de détection d’intrusion NIDS et HIDS

Les approches traditionnelles de détection reposent sur des signatures d’attaques connues ou des règles statiques définies par des experts en sécurité. Bien que efficaces face à des menaces connues, ces méthodes montrent des limites importantes face à des attaques nouvelles, évolutives ou inconnues (*zero-day attacks*). C’est dans ce contexte que l’apprentissage automatique (Machine Learning) s’impose comme une solution prometteuse, permettant d’identifier automatiquement des comportements suspects à partir de données historiques.

### Représentation et signification des données réseau en détection d’intrusions

La détection d’intrusions réseau repose sur l’analyse de **données réseau structurées**, issues de la capture et de l’observation du trafic.

Les données réseau peuvent être représentées selon plusieurs niveaux :

#### Paquets réseau

Un **paquet réseau** est l’**unité élémentaire de communication** dans les réseaux informatiques. Lorsqu’un ordinateur envoie des données à un autre, l’information est découpée en petits segments appelés paquets. Chaque paquet contient plusieurs éléments essentiels :

* **En-tête (header)** : informations de contrôle nécessaires pour le transport.

**Adresse IP source et destination** : identifie les machines émettrices et réceptrices.

**Protocole (TCP/UDP/ICMP, etc.)** : indique comment transmettre et traiter le paquet.

**Ports source et destination** : indiquent quelle application ou service est visé.

**Autres champs de contrôle** : tailles, flags de contrôles (ex : SYN, ACK en TCP), numéros de séquence, checksum, etc.

* **Données (payload)** : la partie utile transportée par le paquet, qui peut être un fragment d’un fichier, un message, une requête, etc.

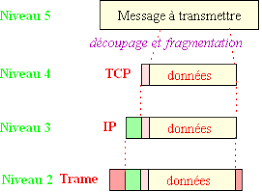


Figure 2 : Encapsulation des flux réseau dans les couches TCP/IP

Le rôle du paquet est de permettre la communication fiable entre hôtes sur un réseau, même lorsque les conditions de transport sont variables. En termes de détection d’intrusions, l’analyse des paquets permet de repérer des anomalies au niveau même du trafic brut — par exemple, un grand nombre de paquets SYN (Synchronize) non suivis d’un ACK (Acknowledgment) peut indiquer une attaque par déni de service de type SYN flood.

****

Figure 3 : Établissement d’une connexion TCP

L’étude des paquets réseau permet de comprendre **le comportement élémentaire des communications**, ce qui constitue la base de la construction de caractéristiques (features) exploitables par un modèle de Machine Learning.

#### Flux réseau (network flows)

Regroupements de paquets partageant des caractéristiques communes (adresse source/destination, ports, protocole).

#### Sessions de communication

Interactions complètes entre deux entités sur une période donnée, chaque flux ou session réseau est décrit par un ensemble de **caractéristiques (features)**, telles que :

* La durée de la connexion,
* Le nombre de paquets échangés,
* Le volume de données transférées,
* Les drapeaux TCP (SYN, ACK...)
* Les taux de paquets entrants et sortants.

Ces caractéristiques constituent une **représentation numérique du comportement réseau**. Un trafic dit normal présente des motifs réguliers et prévisibles, tandis qu’un trafic malveillant se distingue par des anomalies statistiques ou comportementales. L’objectif du Machine Learning est d’apprendre ces motifs afin d’identifier automatiquement les déviations significatives indiquant une intrusion.

### Exemples d’applications de la détection d’intrusions réseau

La détection d’intrusions réseau est aujourd’hui utilisée dans de nombreux contextes critiques, notamment :

* **Sécurisation des réseaux d’entreprise**, afin de protéger les systèmes internes contre les attaques externes ou internes.
* **Protection des infrastructures cloud**, où la surveillance du trafic est essentielle pour prévenir les attaques à grande échelle.
* **Centres de supervision de la sécurité (SOC)**, qui utilisent des IDS pour analyser les alertes de sécurité en temps réel.
* **Réseaux IoT**, particulièrement vulnérables en raison du grand nombre d’appareils connectés et de leurs capacités limitées en sécurité.
* **Détection des attaques DoS**, permettant d’anticiper ou d’atténuer les surcharges réseau.

L’intégration du Machine Learning dans ces systèmes améliore considérablement leur capacité à détecter des attaques complexes ou inédites, tout en réduisant le taux de faux positifs.

### Lien théorique et complexité en détection d’intrusions réseau

La détection d’intrusions réseau constitue un problème complexe en raison de plusieurs facteurs. Tout d’abord, le volume et la vitesse du trafic réseau rendent l’analyse en temps réel particulièrement difficile. Ensuite, les jeux de données utilisés sont souvent déséquilibrés, avec une majorité de trafic normal et une minorité d’attaques, ce qui complique l’apprentissage des modèles.

De plus, les comportements réseau évoluent constamment, rendant obsolètes les règles statiques utilisées par les systèmes traditionnels. Les méthodes basées sur le Machine Learning permettent de répondre à cette complexité en apprenant automatiquement à partir des données et en s’adaptant à de nouveaux schémas d’attaque.

Cependant, ces approches introduisent également des défis théoriques, tels que le choix des caractéristiques pertinentes, la gestion du surapprentissage (overfitting), ainsi que l’interprétabilité des modèles. **L’utilisation de plateformes cloud comme Azure Machine** **Learning** permet de répondre à ces enjeux en offrant des outils avancés pour l’entraînement, l’évaluation et le déploiement de modèles robustes et évolutifs.

#### Modèle utilisé

Dans ce projet, nous avons développé un système de détection d’intrusion (IDS – Intrusion Detection System) basé sur des techniques d’apprentissage automatique. L’objectif est de permettre à un modèle intelligent d’analyser automatiquement des flux réseau et de déterminer s’ils sont normaux ou malveillants.

Le modèle principal utilisé est Random Forest, un algorithme d’ensemble (ensemble learning) composé de plusieurs arbres de décision.

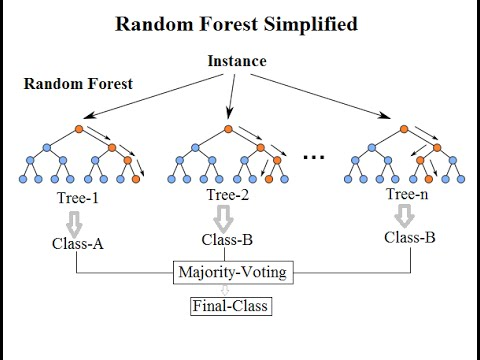
****

Figure 4 : Architecture de l’algorithme Random Forest

Le système développé fonctionne en deux configurations :

* **Classification binaire** : distinguer le trafic BENIGN (normal) du trafic ATTACK (malveillant)
* **Classification multi-classes** : identifier le type exact d’attaque (DoS Hulk, GoldenEye, etc.)

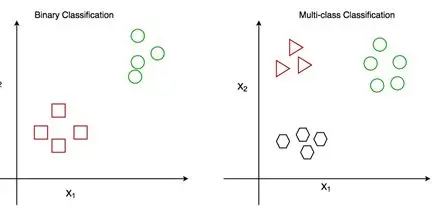


Figure 5 : Classification binaire et multi-classes

Cette double approche permet de couvrir à la fois la détection générale et l’identification précise des menaces.

###### Objectif du modèle

L’objectif principal du modèle est de détecter automatiquement des comportements anormaux dans le trafic réseau à partir de données historiques.

Plus précisément, le système doit :

* Analyser les caractéristiques statistiques d’un flux réseau (durée, nombre de paquets, bytes envoyés, taux d’erreurs, etc.)
* Apprendre à reconnaître les schémas typiques d’une attaque
* Classifier chaque nouveau flux comme normal ou malveillant
* Déclencher une alerte si la probabilité d’attaque dépasse un certain seuil

Dans un contexte réel, ce type de modèle peut être intégré dans un système de surveillance réseau afin d’assister les équipes de sécurité (SOC) dans la détection proactive des menaces.

Notre projet vise à reproduire un IDS intelligent capable d’apprendre à partir de données passées pour anticiper des comportements futurs.

###### Type de Machine Learning utilisé

Le projet repose sur un apprentissage supervisé (Supervised Learning).

Dans ce type d’apprentissage, le modèle est entraîné sur un dataset contenant des données déjà étiquetées. Chaque flux réseau dans le dataset **CICIDS2017** est associé à un label indiquant s’il est :

* BENIGN (trafic normal)
* Un type spécifique d’attaque

Le modèle apprend la relation entre les caractéristiques du flux (features) et son label. Une fois entraîné, il peut prédire la classe de nouveaux flux inconnus.

Il s’agit d’un problème de classification, car le modèle doit attribuer une classe parmi plusieurs catégories possibles.

###### Algorithme choisi : Random Forest

Random Forest est un algorithme d’apprentissage automatique basé sur un ensemble d’arbres de décision.

Son fonctionnement repose sur trois principes fondamentaux :

* **Bootstrap sampling** : Le dataset est divisé en plusieurs sous-ensembles aléatoires.
* **Construction de plusieurs arbres** : Chaque arbre est entraîné sur un sous-ensemble différent des données et sur un sous-ensemble aléatoire des variables.
* **Vote majoritaire** : Lors de la prédiction, chaque arbre donne une décision et la classe finale est déterminée par vote.

Cette méthode permet :

* De réduire le sur-apprentissage (overfitting)
* D’améliorer la stabilité des prédictions
* De mieux généraliser sur des données nouvelles

Dans notre cas, le dataset contient un grand nombre de variables numériques décrivant le trafic réseau. Random Forest est particulièrement adapté à ce type de données structurées.

###### Fonctionnement du système

Le fonctionnement du système suit un pipeline structuré en plusieurs étapes :

1. Chargement du dataset CICIDS2017 dans l’environnement Azure.
2. Nettoyage des données :

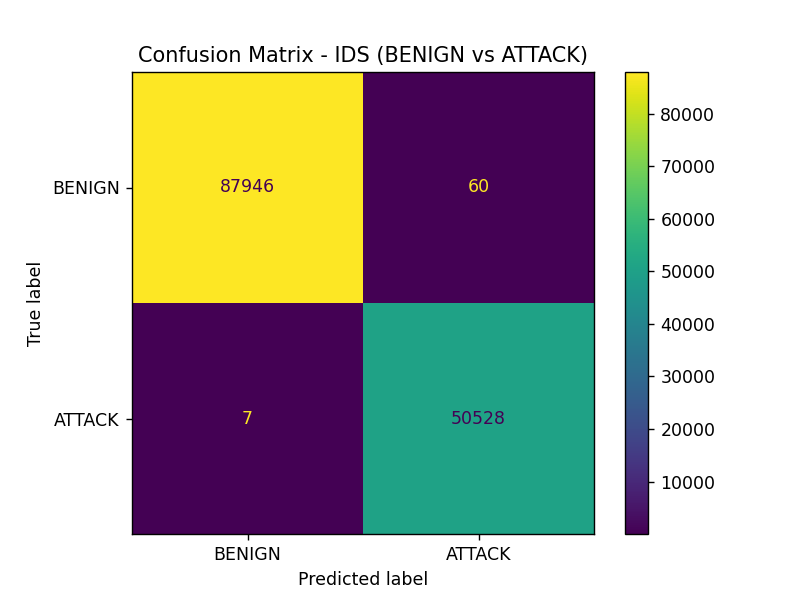
* Suppression des valeurs infinies
* Remplacement des valeurs manquantes par la médiane
* Sélection des colonnes numériques uniquement

1. Encodage des labels (BENIGN = 0, ATTACK = 1).
2. Séparation des données en deux parties :

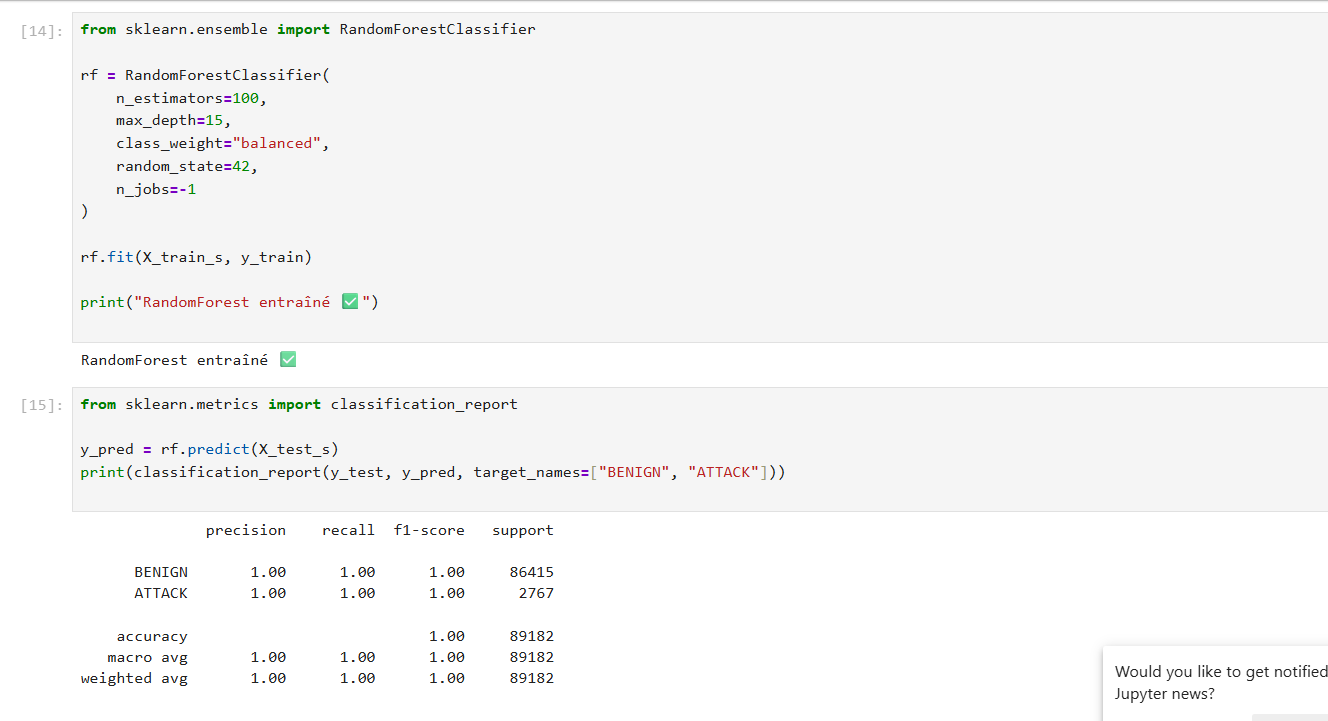
* 80 % pour l’entraînement
* 20 % pour le test

1. Normalisation des données avec StandardScaler.
2. Entraînement du modèle Random Forest.
3. Évaluation des performances à l’aide de :

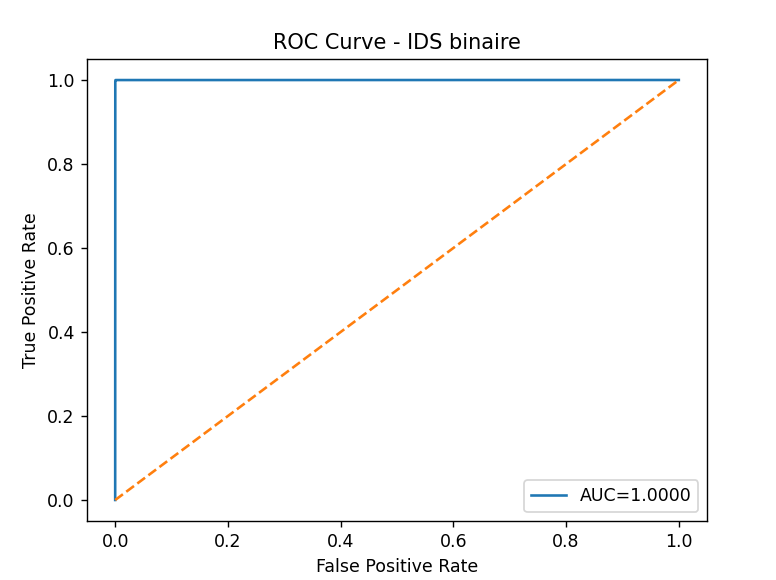
* Matrice de confusion



* Precision, Recall, F1-score



* Courbe ROC et score AUC



1. Simulation d’un environnement temps réel :

* Les flux sont analysés un par un
* Une probabilité d’attaque est calculée
* Une alerte est déclenchée si le seuil (0,7) est dépassé

Ce pipeline reproduit le fonctionnement d’un IDS opérationnel.

###### Pourquoi ce modèle est-il adapté à la détection d’intrusions ?

La détection d’intrusion présente plusieurs défis :

* Données déséquilibrées (beaucoup de trafic normal, peu d’attaques)
* Comportements variés selon le type d’attaque
* Présence de bruit et de valeurs extrêmes

Random Forest est adapté car :

* Il est robuste face aux données bruitées
* Il fonctionne bien avec des datasets déséquilibrés
* Il gère efficacement un grand nombre de variables
* Il offre de bonnes performances en Recall

En cybersécurité, le **Recall** est essentiel : il est préférable de détecter un maximum d’attaques, même si cela génère quelques faux positifs.

De plus, le modèle fournit une probabilité d’attaque, ce qui permet d’ajuster dynamiquement le seuil d’alerte.

###### Intégration du modèle dans Azure Machine Learning

Le projet a été développé et exécuté dans l’environnement Azure Machine Learning.

*Les étapes réalisées sont :*

1. Création d’un Workspace Azure ML.
2. Création d’une Compute Instance pour exécuter le code Python.
3. Importation du dataset CICIDS2017 dans Azure.
4. Exécution du notebook d’entraînement dans le cloud.
5. Entraînement et évaluation des modèles.
6. Sauvegarde du modèle et du scaler.
7. Génération d’une API Flask pour un déploiement futur.

*L’utilisation d’Azure permet :*

* Une exécution dans un environnement cloud professionnel
* Une meilleure gestion des ressources
* Une possibilité de déploiement en production

#### Base utilisée

###### Qu'est-ce que la base utilisée ?

*Présentation générale*

Dans le cadre de ce projet, nous avons utilisé la base de données **CICIDS2017 (Canadian Institute for Cybersecurity Intrusion Detection System 2017)**.  
Cette base a été développée par le Canadian Institute for Cybersecurity dans le but de fournir un dataset réaliste et complet pour l’évaluation des systèmes de détection d’intrusion (IDS).

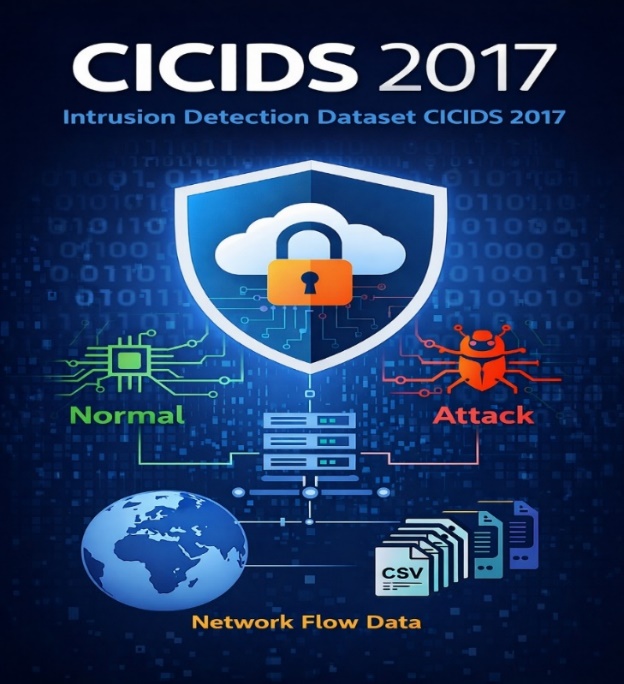


Figure 6 : Dataset CICIDS2017 pour la détection d’intrusions

CICIDS2017 est aujourd’hui l’un des datasets les plus utilisés dans la recherche en cybersécurité et en Machine Learning appliqué aux réseaux.

Son objectif est de reproduire un environnement réseau réel d’entreprise comprenant :

* Du trafic normal
* Différents scénarios d’attaques informatiques

Elle constitue donc une base adaptée à l’entraînement et à l’évaluation de modèles d’intelligence artificielle pour la détection d’intrusions.

*Nature des données*

La base CICIDS2017 est composée de **flux réseau (network flows)**.

Un flux réseau représente une communication entre deux machines sur un réseau, caractérisée par plusieurs paramètres tels que :

* Adresse IP source
* Adresse IP destination
* Port source et port destination
* Protocole utilisé (TCP, UDP…)
* Durée de la communication
* Nombre total de paquets échangés
* Nombre total d’octets transférés
* Taux de paquets par seconde
* Statistiques sur les délais inter-paquets
* Indicateurs d’erreurs ou de comportements anormaux

Chaque flux est décrit par plus de **70 caractéristiques numériques**, extraites à partir de captures réseau (PCAP) à l’aide d’outils d’analyse spécialisés.

Ces caractéristiques permettent de décrire précisément le comportement du trafic réseau, ce qui rend possible l’apprentissage automatique.

*Fichier utilisé dans le projet*



Figure 7 : Ensemble des fichiers composant le dataset CICIDS2017 utilisés dans ce projet

La base de données est composée de plusieurs fichiers correspondant à différentes périodes d’enregistrement. Après une étude approfondie, nous avons identifié que le fichier « Wednesday » contient à la fois du trafic normal et du trafic malveillant (attaques). Par conséquent, ce fichier constitue le choix le plus approprié pour notre projet, car il permet d’entraîner et d’évaluer efficacement notre modèle de détection.

**Wednesday-workingHours.pcap\_ISCX.csv**

Ce fichier correspond aux flux capturés pendant une journée spécifique (mercredi) dans un environnement simulé.

Il contient :

* Du trafic normal (BENIGN)
* 4 attaques de type DoS (Denial of Service)

Chaque ligne du fichier correspond à un flux réseau. Chaque flux possède un label (colonne "Label") indiquant sa nature.

*Types d’attaques présentes*

Dans le fichier utilisé, on retrouve principalement les attaques suivantes :

* *DoS Hulk*
* *DoS GoldenEye*
* *DoS Slowloris*
* *DoS Slowhttptest*

Ces attaques appartiennent à la catégorie **Denial of Service (DoS)**.

Leur objectif est de rendre un serveur indisponible en :

* Envoyant un volume massif de requêtes
* Saturant les ressources système
* Bloquant les connexions légitimes

La présence de plusieurs variantes de DoS permet de tester la capacité du modèle à :

* Détecter une attaque de manière générale
* Différencier plusieurs types d’attaques

*Structure et taille des données*

Le dataset contient :

* Des dizaines de milliers de flux réseau
* Une majorité de trafic BENIGN
* Une minorité de flux correspondant à des attaques

Cela introduit un **déséquilibre des classes**, ce qui reflète la réalité : dans un réseau réel, la majorité du trafic est normal, et seules certaines portions correspondent à des attaques.

Ce déséquilibre constitue un défi important pour le Machine Learning, car le modèle doit apprendre à détecter des événements rares.

*Prétraitement effectué sur la base*

Avant d’entraîner le modèle, plusieurs étapes de préparation ont été réalisées :

* Suppression des valeurs infinies
* Remplacement des valeurs manquantes par la médiane
* Sélection des colonnes numériques uniquement
* Normalisation des données avec StandardScaler
* Encodage des labels (BENIGN = 0, ATTACK = 1 pour le modèle binaire)

Ces étapes sont essentielles pour garantir la stabilité et la performance du modèle.

*Les raisons pour lesquelles cette base est adaptée à notre projet*

La base CICIDS2017 est particulièrement adaptée pour plusieurs raisons :

1. **Réalisme :** Les données ont été générées dans un environnement proche d’un réseau d’entreprise réel.
2. **Variété des attaques :** Plusieurs types d’attaques sont inclus, ce qui permet une analyse multi-classes.
3. **Richesse des caractéristiques** : Le grand nombre de variables permet au modèle d’identifier des patterns complexes.
4. **Référence académique :** Elle est largement utilisée dans la littérature scientifique, ce qui renforce la validité du projet.
5. **Compatibilité avec le Machine Learning :** Les données sont structurées sous forme tabulaire, ce qui facilite leur utilisation avec des algorithmes comme Random Forest.

*Rôle de la base dans notre système IDS*

Dans notre projet, la base CICIDS2017 a permis :

* D’entraîner le modèle à reconnaître les comportements malveillants
* D’évaluer ses performances sur des données non vues
* De simuler un système IDS en temps réel
* De tester une classification binaire et multi-classes

Le modèle apprend à partir des exemples passés afin de généraliser sur de nouveaux flux.

Ainsi, cette base constitue le cœur de notre système de détection.

## Synergie entre Microsoft Azure et le Machine Learning pour une détection d’intrusion réseau efficace

### Azure comme infrastructure cloud sécurisée

Dans ce projet, **Microsoft Azure** a servi d’infrastructure principale pour héberger et exécuter l’ensemble de la solution de détection d’intrusion. Azure offre un environnement cloud sécurisé, fiable et scalable, permettant de stocker les données réseau, d’exécuter les traitements et de gérer les ressources de calcul (CPU, mémoire, stockage). L’utilisation du cloud permet également une grande flexibilité : en cas d’augmentation du volume de trafic réseau, les ressources peuvent être ajustées dynamiquement. Cette approche garantit une haute disponibilité du système et une meilleure gestion des performances, ce qui est essentiel pour un projet de cybersécurité où la rapidité de traitement est cruciale.

### Azure Machine Learning comme plateforme d’entraînement

La plateforme **Azure Machine Learning** a permis de structurer et professionnaliser l’entraînement du modèle de Machine Learning. Elle offre un environnement dédié à la création d’expériences, à l’entraînement des algorithmes, au suivi des performances et au stockage des modèles entraînés. Grâce à cette plateforme, nous avons pu exécuter l’algorithme Random Forest, évaluer ses métriques (accuracy, précision, rappel) et sauvegarder le modèle final. Azure Machine Learning facilite également la gestion des versions et prépare le déploiement futur du modèle sous forme d’API, rendant ainsi le projet exploitable dans un contexte réel d’entreprise.

### Complémentarité avec les services de sécurité Azure

L’intégration du modèle avec les outils de sécurité Azure renforce considérablement son efficacité. Des solutions comme **Microsoft Defender for Cloud** ou **Microsoft Sentinel** collectent et centralisent les événements de sécurité, les logs réseau et les alertes système. Ces outils fonctionnent généralement avec des règles prédéfinies pour détecter des menaces connues. L’ajout d’un modèle de Machine Learning permet d’aller plus loin en détectant des comportements anormaux ou des attaques nouvelles, basées sur des schémas appris à partir des données. Ainsi, la combinaison entre règles de sécurité classiques et intelligence artificielle améliore la précision et réduit les faux positifs.

### Apport concret de cette synergie

La synergie entre Azure et le Machine Learning transforme un simple modèle académique en une solution proche des standards professionnels. Elle permet une détection automatisée des intrusions, une analyse rapide de grands volumes de données et une adaptation continue face à l’évolution des menaces. Cette architecture cloud intelligente réduit le temps de réaction face aux attaques et améliore la capacité d’anticipation des risques. En combinant puissance de calcul, centralisation des données et intelligence algorithmique, cette approche constitue une base solide pour la mise en place d’un système de détection d’intrusion moderne, évolutif et adapté aux environnements cloud actuels.

# État de l'Art

## Modèles existants de détection d’intrusions réseau

La détection d’intrusions réseau (IDS) a fait l’objet de nombreuses recherches au cours des dernières années. Plusieurs modèles de Machine Learning et de Deep Learning ont été proposés afin d’identifier automatiquement les comportements malveillants dans le trafic réseau. Ces modèles sont généralement entraînés sur des bases de données publiques spécialisées en cybersécurité, telles que KDD99, NSL-KDD ou CICIDS2017. Chaque approche repose sur une architecture différente et présente des avantages spécifiques en termes de précision, complexité et capacité de généralisation.

### Base et architecture

|  |  |  |
| --- | --- | --- |
| ***Modèle*** | ***Base/Entra*î*nement*** | ***Architecture*** |
| **Random Forest** | CICIDS2017 | L’architecture Random Forest repose sur un ensemble d’arbres de décision. Chaque arbre prend une décision indépendante (attaque ou trafic normal) en analysant différentes caractéristiques du flux réseau. Ensuite, tous les arbres votent, et la décision finale correspond à la majorité des votes.  Cette architecture permet de réduire le surapprentissage (overfitting) et d’améliorer la précision globale du modèle, car plusieurs arbres travaillent ensemble au lieu d’un seul. |
| **SVM** | NSL-KDD | |  | | --- | | L’architecture SVM consiste à tracer une frontière de séparation optimale entre deux classes (normal / attaque). Le modèle cherche à maximiser la distance entre cette frontière et les points les plus proches (appelés vecteurs de support). Cette approche fonctionne très bien lorsque les données sont bien séparables, mais elle peut devenir plus complexe lorsque le nombre de caractéristiques est élevé. | |
| **ANN** | |  | | --- | | CICIDS2017 | | |  | | --- | |  | | Un ANN est composé de plusieurs couches de neurones artificiels :   1. Une couche d’entrée 2. Une ou plusieurs couches cachées 3. Une couche de sortie   Chaque neurone applique une transformation mathématique aux données reçues. Les couches cachées permettent d’extraire des relations complexes entre les caractéristiques du trafic réseau.  Ce type d’architecture est capable d’apprendre des patterns non linéaires, mais nécessite plus de données et de puissance de calcul. | |
| **CNN** | CICIDS2017 | |  | | --- | |  | | Un CNN utilise des couches de convolution qui analysent les données en appliquant des filtres automatiques. Ces filtres permettent d’extraire des motifs importants dans les données. Même si les CNN sont surtout connus pour les images, ils peuvent être adaptés aux données réseau en transformant les caractéristiques en matrices. Cette architecture est plus complexe mais peut capturer des relations avancées entre les variables | |
| **LSTM** | CICIDS2017 | |  | | --- | | Un CNN utilise des couches de convolution qui analysent les données en appliquant des filtres automatiques. Ces filtres permettent d’extraire des motifs importants dans les données. Même si les CNN sont surtout connus pour les images, ils peuvent être adaptés aux données réseau en transformant les caractéristiques en matrices. Cette architecture est plus complexe mais peut capturer des relations avancées entre les variables. | |  | |

# Pré-Réalisation

## Analyse du Modèle de Départ :

### Règles de Base de la Détection d’intrusions réseau

Avant de concevoir et d’entraîner un modèle de détection d’intrusion, il est essentiel de comprendre les principes fondamentaux qui régissent la surveillance et l’analyse du trafic réseau. La détection d’intrusions repose sur un ensemble de règles et de mécanismes permettant d’identifier des comportements anormaux ou malveillants dans un système informatique.

Cette étape d’analyse constitue la base théorique du modèle développé dans ce projet. Elle permet de définir les critères de distinction entre trafic normal et trafic suspect, ainsi que les mécanismes de décision utilisés pour classer les flux réseau. Comprendre ces règles fondamentales est indispensable afin de construire un système de détection cohérent, fiable et adapté aux menaces actuelles.

*Voici les règles de base pour la détection d’intrusions réseau :*

#### Définition

Un système de détection d’intrusion réseau (IDS – Intrusion Detection System) est un mécanisme capable d’analyser automatiquement le trafic réseau afin d’identifier les comportements anormaux ou suspects.

Dans notre projet, l’IDS est basé sur un **modèle d’apprentissage supervisé**. Cela signifie que :

* Le modèle apprend à partir de données étiquetées (normal / attaque).
* Il identifie des patterns dans les caractéristiques du trafic réseau.
* Il utilise ces patterns pour prédire la classe d’un nouveau flux réseau.

Le système fonctionne donc comme un classificateur binaire (ou multi-classes selon ton code) capable de produire une décision automatique.

#### Configuration Initiale

La configuration initiale correspond à toutes les étapes réalisées avant l’utilisation opérationnelle du modèle.  
Dans notre projet, cette phase comprend plusieurs sous-étapes :

###### Chargement et préparation des données

Les données réseau sont importées depuis le dataset utilisé dans le projet.  
Chaque ligne représente une connexion réseau décrite par plusieurs caractéristiques (features) comme :

* Durée de connexion
* Type de protocole
* Nombre de paquets
* Taux d’erreurs
* Nombre de connexions vers la même destination

Ces caractéristiques constituent la base de l’apprentissage du modèle.

###### Prétraitement des données

Avant l’entraînement du modèle, plusieurs opérations sont effectuées :

* Suppression ou traitement des valeurs manquantes
* Encodage des variables catégorielles (ex : protocole TCP/UDP/ICMP)
* Normalisation ou standardisation des variables numériques
* Séparation des variables explicatives (X) et de la variable cible (y)

Cette étape est essentielle car elle garantit que les données sont exploitables par le modèle.

###### Séparation des données

*Les données sont divisées en :*

* Ensemble d’entraînement (training set)
* Ensemble de test (test set)

Cette séparation permet d’évaluer objectivement les performances du modèle sur des données qu’il n’a jamais vues.

###### Entraînement du modèle

Le modèle de Machine Learning (celui que tu as utilisé dans ton code) est entraîné sur les données d’apprentissage.

Pendant cette phase, le modèle apprend les relations entre les caractéristiques du trafic réseau et les classes (normal / attaque).

C’est ici que l’architecture du système prend forme :

**Données → Prétraitement → Modèle → Prédiction**

#### Règles de Transition

Étape du processus :

Les règles de transition décrivent le fonctionnement dynamique du système une fois entraîné.

###### Réception d’un flux réseau

Un nouveau flux réseau est capturé ou fourni au système.

###### Transformation des données

Le flux subit le même traitement que les données d’entraînement :

* Encodage
* Normalisation
* Mise au bon format

Cela garantit la cohérence avec le modèle entraîné.

###### Passage dans le modèle

Le modèle analyse les caractéristiques du flux à l’aide de ses paramètres appris.

Selon l’architecture utilisée dans ton code (par exemple Random Forest, SVM, etc.), la décision est prise en fonction :

* Des règles internes (arbres de décision)
* De la distance par rapport aux classes
* Ou d’une fonction de séparation

###### Classification

Le flux est classé comme :

* Trafic normal
* Attaque (avec éventuellement un type spécifique (Dos))

###### Résultat et performance

Les résultats sont évalués à l’aide de métriques calculées dans ton code, comme :

* Accuracy (précision globale)
* Matrice de confusion
* Précision
* Recall
* F1-score

Ces résultats permettent de mesurer l’efficacité du système et de vérifier sa capacité à détecter correctement les intrusions.

#### Boucle de Simulation

La boucle de simulation représente l’exécution dynamique du système de détection d’intrusions.  
Elle permet de tester le modèle entraîné sur de nouvelles données afin d’observer son comportement et ses performances.

###### Objectif de la simulation

* Reproduire le fonctionnement réel du système de détection d’intrusion.
* Tester le modèle entraîné sur des données nouvelles (ensemble de test).
* Observer le comportement du système face à différents types de trafic réseau.

###### Injection des données

* Les flux réseau de l’ensemble de test sont envoyés au système un par un (ou par lot).
* Chaque flux représente une connexion réseau avec ses caractéristiques (features).
* La simulation imite l’arrivée progressive de données comme dans un environnement réel.

###### Application du prétraitement

* Les mêmes transformations appliquées lors de l’entraînement sont réutilisées :
  + Encodage des variables catégorielles
  + Normalisation / standardisation
  + Sélection des caractéristiques
* Cela garantit la cohérence entre phase d’entraînement et phase de test.

###### Prédiction du modèle

* Chaque flux est transmis au modèle entraîné.
* Le modèle applique les paramètres appris pendant l’entraînement.
* Il attribue une classe :
  + Trafic normal
  + Intrusion du type Dos.

###### Comparaison avec les vraies étiquettes

* Les prédictions sont comparées aux valeurs réelles.
* Cela permet d’identifier :
  + Les détections correctes
  + Les faux positifs
  + Les faux négatifs

###### Analyse des résultats

* Identification des points forts du modèle (bonne détection des attaques).
* Analyse des erreurs de classification.
* Évaluation de la capacité du système à généraliser sur des données non vues.

La boucle de simulation démontre le fonctionnement opérationnel du système et valide sa capacité à détecter automatiquement les intrusions réseau dans des conditions proches du réel.

### Modèle pris en exemple

Le projet académique dont nous nous sommes inspirés porte sur la conception d’un système de détection d’intrusions réseau (IDS) basé sur des techniques d’apprentissage automatique. L’objectif principal est d’identifier automatiquement les activités malveillantes présentes dans un trafic réseau en analysant différentes caractéristiques techniques issues des connexions, telles que la durée des flux, le nombre de paquets échangés, le volume de données transférées, les protocoles utilisés ou encore certains indicateurs statistiques du trafic. Contrairement aux approches traditionnelles basées uniquement sur des règles fixes ou des signatures connues, ce projet propose une approche intelligente où le modèle apprend à reconnaître les comportements suspects à partir des données.

Le travail suit une méthodologie structurée. Il commence par la collecte et la préparation du jeu de données, incluant le nettoyage des valeurs manquantes ou aberrantes, la transformation des variables, la normalisation des données numériques et l’encodage des classes (trafic normal ou attaque). Les données sont ensuite divisées en un ensemble d’entraînement et un ensemble de test afin d’évaluer objectivement les performances du modèle. Plusieurs algorithmes de classification sont testés afin de comparer leur efficacité dans la détection des attaques. Les résultats sont analysés à l’aide de métriques telles que l’accuracy, la précision, le rappel, le F1-score et la matrice de confusion, permettant d’évaluer la capacité du système à détecter correctement les intrusions tout en limitant les faux positifs.

L’architecture globale du projet repose sur une chaîne logique composée de plusieurs étapes : acquisition des données, prétraitement, entraînement du modèle, validation, puis analyse des résultats. Cette organisation montre l’importance d’un pipeline clair et rigoureux pour obtenir un système fiable et exploitable. Le projet met également en évidence certaines limites, notamment la dépendance à la qualité des données et la difficulté à détecter des attaques totalement nouvelles non présentes dans le jeu d’entraînement.

Dans le cadre de notre projet, nous avons repris cette méthodologie académique comme base de travail, tout en l’adaptant à notre propre approche. Nous avons développé notre modèle de détection, structuré un pipeline complet incluant le nettoyage des données, l’entraînement, l’évaluation, la sauvegarde des modèles et une simulation de détection en temps réel. De plus, nous avons intégré l’ensemble de la solution dans un environnement cloud via Azure Machine Learning sur la plateforme Microsoft Azure. Cette intégration nous a permis de bénéficier de ressources de calcul adaptées, de gérer nos expérimentations de manière professionnelle, de versionner nos modèles et de préparer un déploiement évolutif, rapprochant ainsi notre travail d’un contexte réel de cybersécurité en entreprise.

### Structure du programme

Le code de notre projet est organisé de manière structurée afin de garantir une bonne lisibilité, une logique claire et une séparation des différentes étapes du processus de détection d’intrusions. Il est divisé en plusieurs parties principales :

###### Création du Workspace Azure

La première étape a été la création d’un Workspace sur Azure Machine Learning via la plateforme Microsoft Azure.

Le workspace constitue l’espace de travail central du projet. Il permet de regrouper :

* Les expériences d’entraînement
* Les datasets
* Les modèles entraînés
* Les environnements d’exécution
* Les métriques et résultats

Ce workspace sert donc de centre de gestion du projet Machine Learning, en assurant l’organisation, la traçabilité et la reproductibilité des expérimentations.

###### Configuration des Ressources

Après la création du workspace, nous avons configuré :

* Un **Compute Instance** pour exécuter le code et entraîner les modèles
* Un environnement Python contenant les bibliothèques nécessaires (scikit-learn, pandas, numpy, matplotlib)
* L’importation du dataset dans l’espace Azure

Cette étape garantit que l’entraînement du modèle se fait dans un environnement contrôlé.

* *Implémentation du Code ML*

###### Importation des bibliothèques

Cette première section regroupe l’ensemble des bibliothèques nécessaires au fonctionnement du projet. On y retrouve notamment numpy et pandas pour la manipulation des données, matplotlib pour la visualisation des résultats, ainsi que les modules de scikit-learn pour le prétraitement (StandardScaler, LabelEncoder), la division des données (train\_test\_split), l’entraînement des modèles (LogisticRegression, RandomForestClassifier) et l’évaluation des performances (classification\_report, confusion\_matrix, ROC curve). La bibliothèque joblib est utilisée pour sauvegarder les modèles entraînés.

###### Configuration des paramètres

Cette partie définit les variables globales du projet, comme le chemin du fichier CSV, le seuil de détection (threshold) pour considérer un flux comme attaque, le délai utilisé dans la simulation temps réel, ainsi que le nombre de flux affichés lors du test. Cela permet de modifier facilement le comportement du système sans toucher à la logique principale.

###### Chargement et préparation des données

Le dataset est chargé à partir du fichier CSV. Les colonnes sont nettoyées, la colonne “Label” est séparée des caractéristiques (features), et seules les colonnes numériques sont conservées. Les valeurs infinies sont remplacées, les valeurs manquantes sont traitées (remplacées par la médiane) et les valeurs extrêmes sont limitées afin d’éviter des erreurs lors de l’entraînement.

###### Transformation des labels

Deux approches sont utilisées :

* Une classification binaire (BENIGN vs ATTACK) pour détecter si un flux est normal ou malveillant.
* Une classification multi-classes pour identifier le type exact d’attaque.

Les labels textuels sont transformés en valeurs numériques pour être exploitables par les modèles.

###### Division des données (Train/Test Split)

Le dataset est séparé en données d’entraînement et de test (80% / 20%) avec stratification afin de conserver la même proportion de classes dans les deux ensembles.

###### Normalisation des données

Les données sont standardisées à l’aide de StandardScaler pour améliorer la stabilité et les performances des modèles de machine learning.

###### Entraînement des modèles

Deux modèles principaux sont utilisés :

* Régression logistique (modèle simple et rapide).
* Random Forest (modèle principal retenu pour ses meilleures performances).

Le temps d’entraînement est mesuré pour évaluer l’efficacité du système.

###### Évaluation des performances

Les performances du modèle sont analysées à l’aide du classification report (précision, rappel, F1-score), de la matrice de confusion, de la courbe ROC et du score AUC. Des histogrammes de probabilités sont également générés pour visualiser la séparation entre trafic normal et attaques.

###### Simulation en temps réel

Une boucle de simulation permet de reproduire un scénario de détection en direct. Des flux de test (normaux et malveillants) sont sélectionnés aléatoirement, puis analysés un par un. Pour chaque flux, la probabilité d’attaque est calculée et comparée à un seuil défini. Le système affiche alors si le flux est considéré comme BENIGN ou ATTACK.

Sauvegarde du modèle   
Les modèles entraînés (binaire et multi-classes), ainsi que les scalers et encodeurs, sont sauvegardés sous forme de fichiers.pkl. Cela permet un futur déploiement sans devoir ré- entrainer le modèle.

### Fonctionnalités principales

Notre système de détection d’intrusions réseau présente plusieurs fonctionnalités essentielles permettant d’assurer une analyse efficace du trafic réseau et une identification précise des comportements malveillants.

###### Détection binaire des intrusions

Le système est capable de classifier automatiquement un flux réseau en deux catégories :

* Trafic normal (BENIGN)
* Trafic malveillant (ATTACK)

Cette détection repose sur un modèle supervisé entraîné sur des données réelles de trafic réseau. Une probabilité d’attaque est calculée pour chaque flux, puis comparée à un seuil de décision afin de déclencher une alerte si nécessaire.

###### Classification multi-classes des attaques

En plus de la détection simple attaque/non attaque, le système peut identifier le type spécifique d’attaque (Dans notre cas (Dos)).

Cette fonctionnalité permet une analyse plus détaillée et facilite la compréhension du comportement malveillant détecté.

###### Prétraitement automatique des données

Le système intègre un module de nettoyage et de transformation des données incluant :

* Suppression des valeurs infinies
* Gestion des valeurs manquantes
* Normalisation des variables numériques
* Encodage des labels

Cela garantit la stabilité et la robustesse du modèle.

###### Évaluation avancée des performances

Le système fournit plusieurs indicateurs de performance :

* Matrice de confusion
* Précision, rappel, F1-score
* Courbe ROC et score AUC

Ces métriques permettent d’analyser la qualité du modèle et son efficacité dans la détection des attaques.

###### Simulation de détection en temps réel

Une boucle de simulation permet d’analyser des flux un par un, comme dans un environnement réel.

Pour chaque flux, le système affiche :

* L’index du flux
* La classe réelle
* La classe prédite
* La probabilité d’attaque

Cela simule un fonctionnement en temps réel d’un IDS.

###### Sauvegarde et réutilisation du modèle

Les modèles entraînés ainsi que les scalers sont sauvegardés au format.pkl.  
Cette fonctionnalité permet :

* Un déploiement futur
* Une intégration dans une API
* Une utilisation dans un environnement cloud

###### Intégration cloud

Le système est intégré dans Azure Machine Learning via Microsoft Azure, permettant :

* La gestion des expériences
* L’exécution sur ressources cloud
* La scalabilité
* Une approche professionnelle orientée production

# Réalisation

## Évolution des versions

### Mise en place de l’environnement Azure Machine Learning

Dans le cadre de ce projet, nous avons utilisé la plateforme **Microsoft Azure Machine Learning** afin de développer, entraîner et tester notre système de détection d’intrusion basé sur le Machine Learning.

Azure Machine Learning est une plateforme cloud fournie par Microsoft qui permet de créer, entraîner et déployer des modèles de Machine Learning dans un environnement sécurisé et scalable. Elle offre plusieurs outils facilitant le développement, notamment la gestion des datasets, la création de machines virtuelles, l’exécution de scripts Python et le déploiement de modèles.

La première étape a consisté à créer un **Workspace Azure Machine Learning**, qui constitue l’espace de travail principal du projet. Ce Workspace permet de centraliser toutes les ressources nécessaires, notamment :

* Les datasets
* Les machines de calcul
* Les notebooks
* Les modèles entraînés

Ce Workspace agit comme un environnement centralisé pour le développement du système IDS.

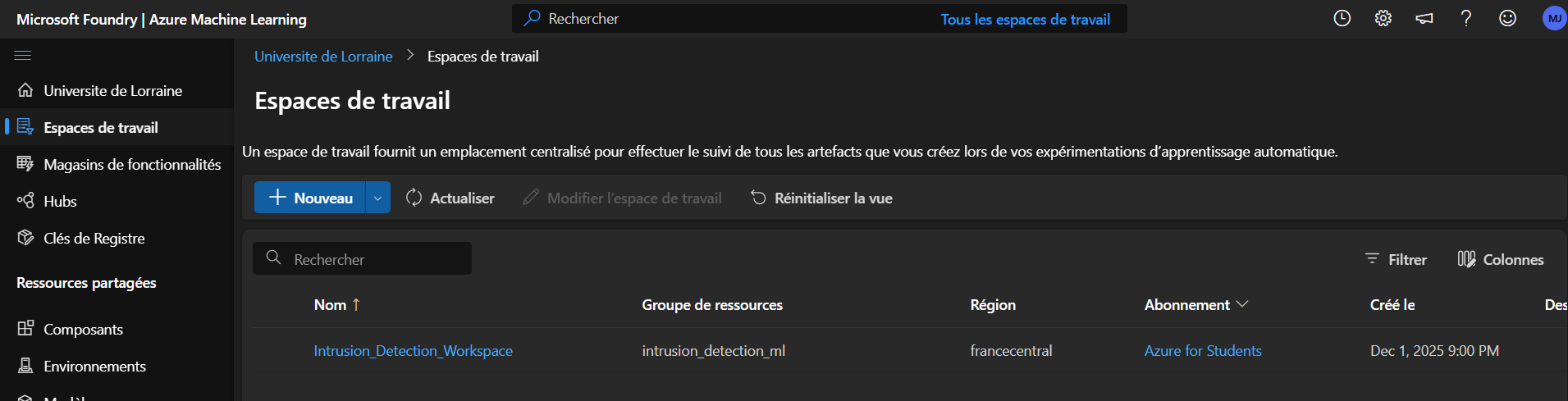


Figure 8 : Création du Workspace Azure Machine Learning

### Création de la machine de calcul (Compute Instance)

Une fois le Workspace créé, nous avons configuré une **Compute Instance**, qui est une machine virtuelle hébergée sur Azure.

Cette machine virtuelle permet d’exécuter les scripts Python nécessaires à :

* L’entraînement du modèle
* Le traitement des données
* La simulation du système IDS

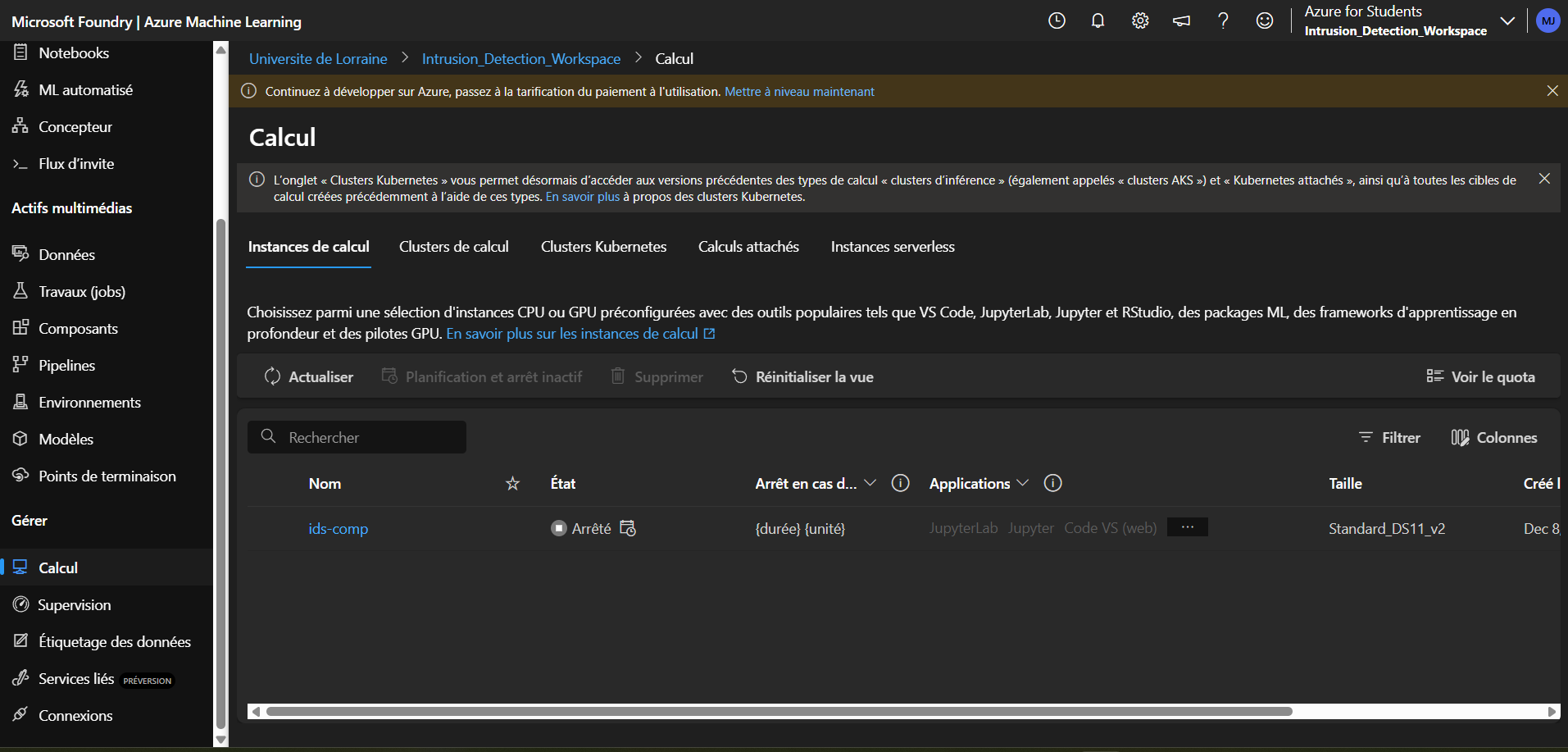


Figure 9 : Configuration de la Compute Instance Azure utilisée pour l’entraînement

*La configuration choisie comprend :*

* Un processeur CPU adapté :

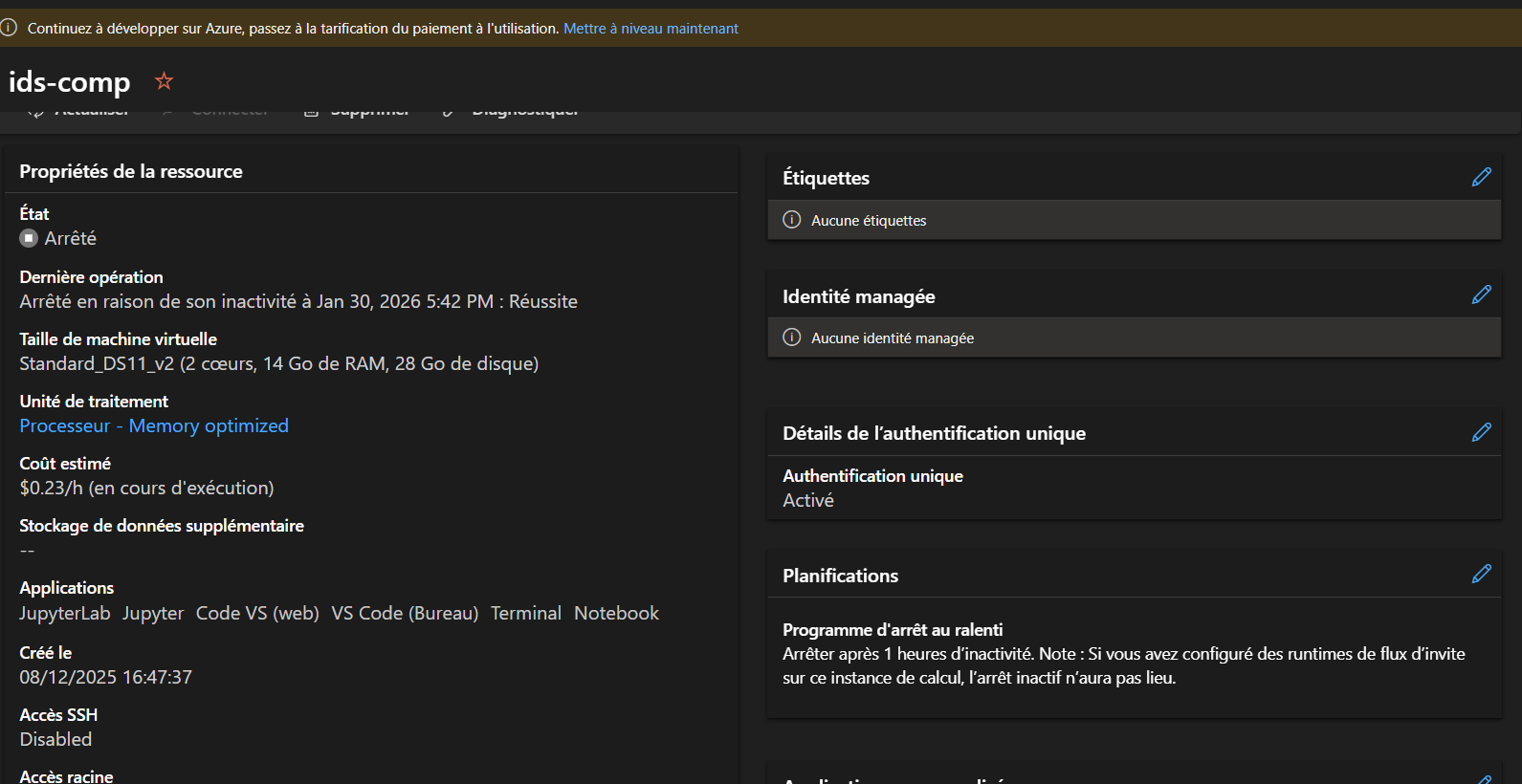


Figure 10 : Configuration de la Compute Instance Azure utilisée pour l’entraînement

* Une mémoire RAM suffisante pour traiter le dataset CICIDS2017
* Un environnement Python avec les bibliothèques nécessaires, notamment : Pandas, numpy, scikit-learn, matplotlib.

Cette machine virtuelle joue un rôle essentiel car elle constitue l’environnement d’exécution du modèle.

### Importation et préparation de la base de données

Après la configuration de l’environnement Azure, nous avons importé la base de données **CICIDS2017** dans le Workspace.

*Le fichier utilisé est:*

*Wednesday-workingHours.pcap\_ISCX.csv*

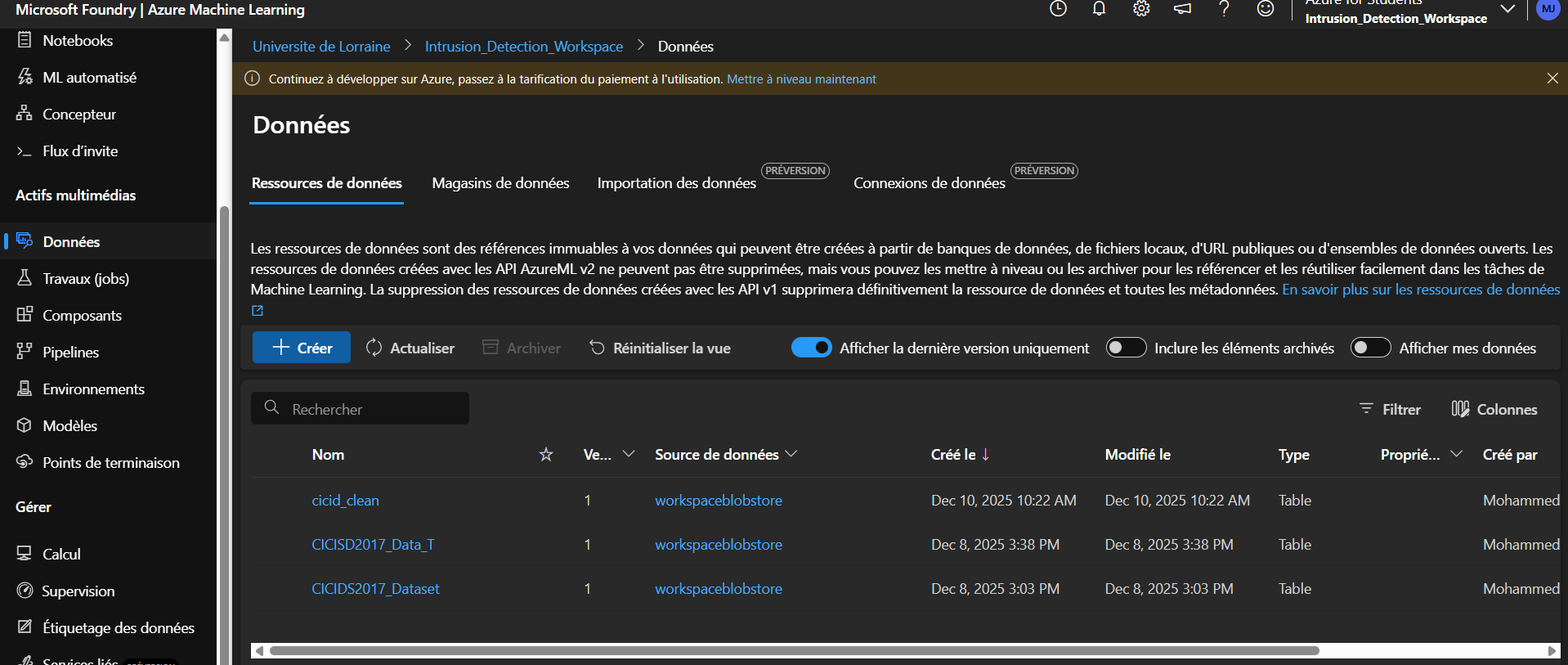
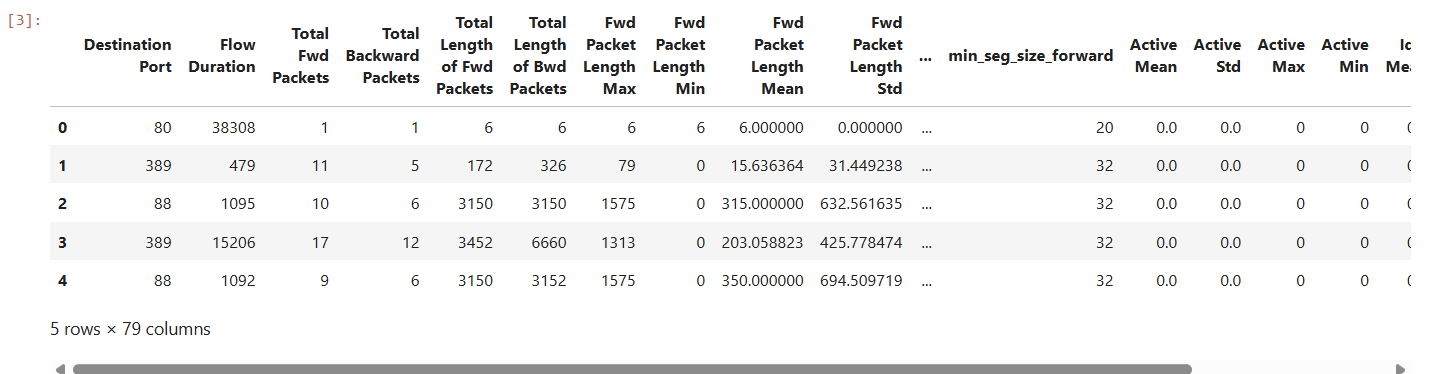


Figure 11 : Détails de la machine virtuelle Compute Instance (CPU, RAM, environnement Python)

Ce dataset contient des informations détaillées sur les flux réseau, incluant :

* Le nombre de paquets
* La durée du flux
* Le nombre d’octets envoyés
* Les statistiques temporelles
* Le label indiquant si le flux est une attaque ou non :



* + Ces données ont été chargées dans l’environnement Azure afin d’être utilisées pour l’entraînement du modèle.

### Prétraitement des données

Avant d’entraîner le modèle, une étape de prétraitement des données a été réalisée en utilisant Python dans Azure Machine Learning.

Cette étape consiste à nettoyer et préparer les données afin de les rendre exploitables par l’algorithme de Machine Learning.

Premièrement, les valeurs infinies ont été remplacées par des valeurs nulles (NaN), puis les lignes contenant des valeurs manquantes ont été supprimées.

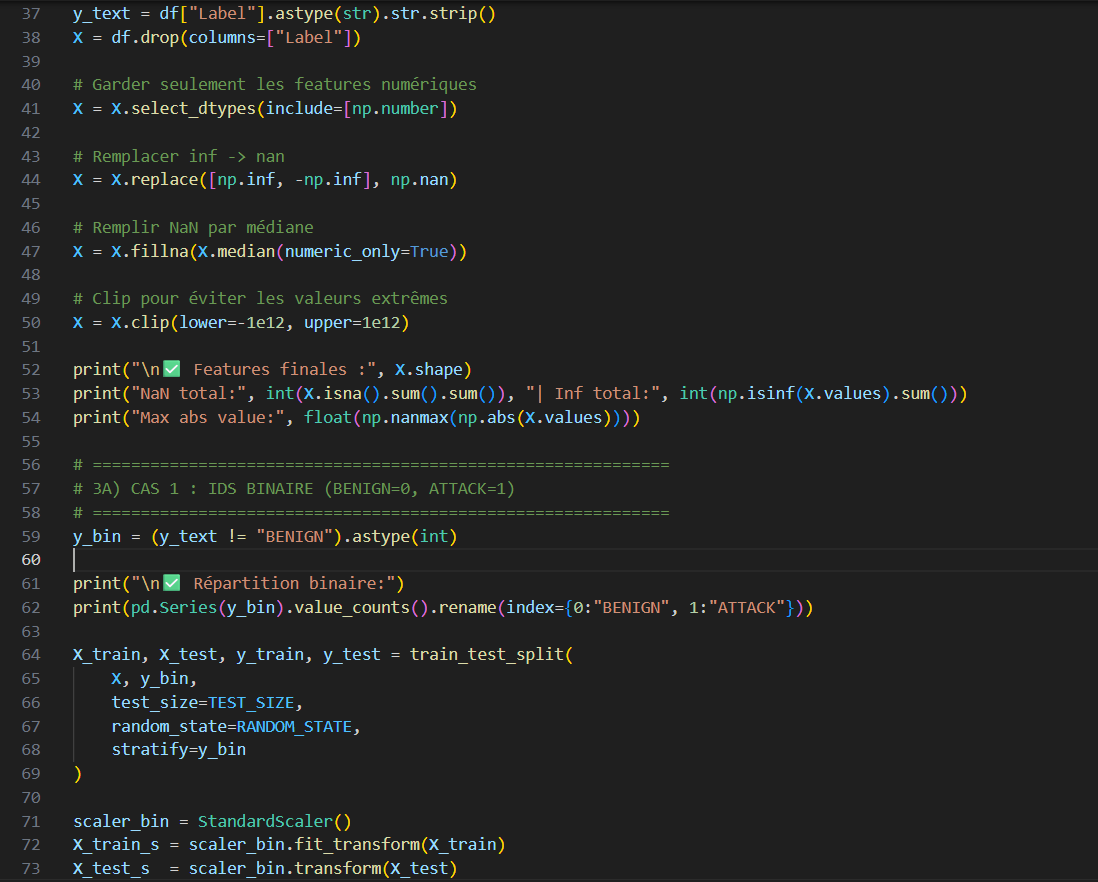
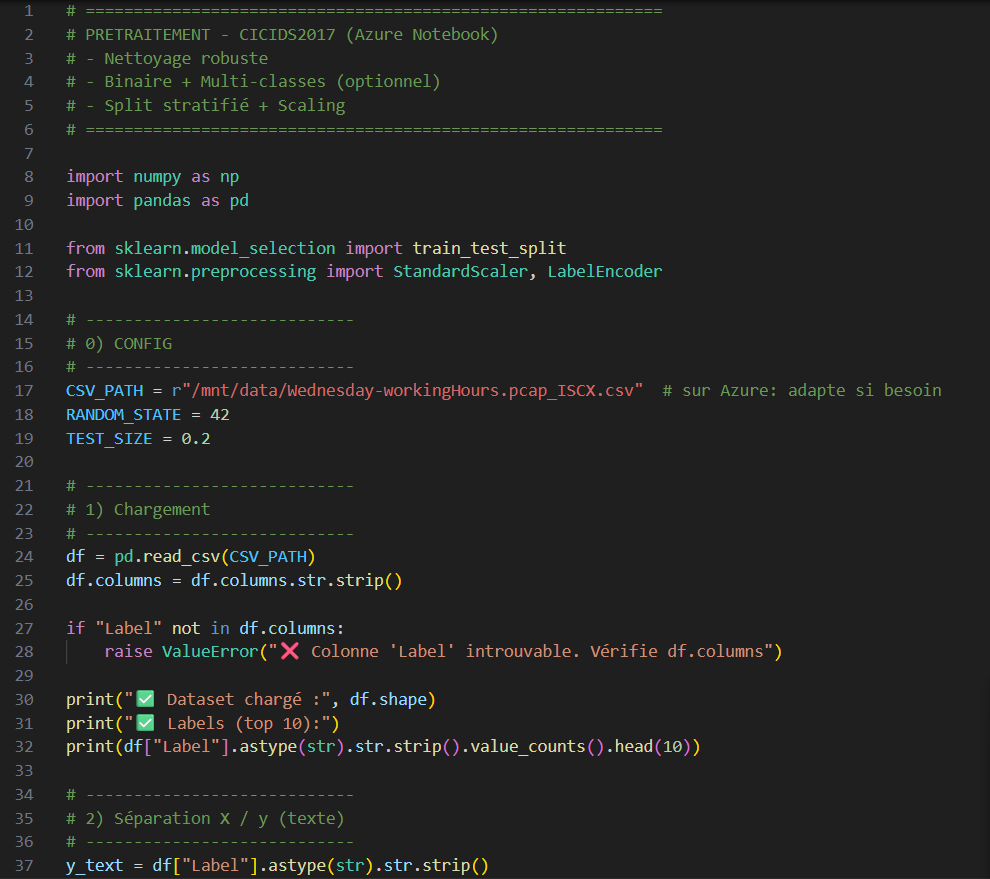
Ensuite, la variable cible Label a été transformée en format numérique :

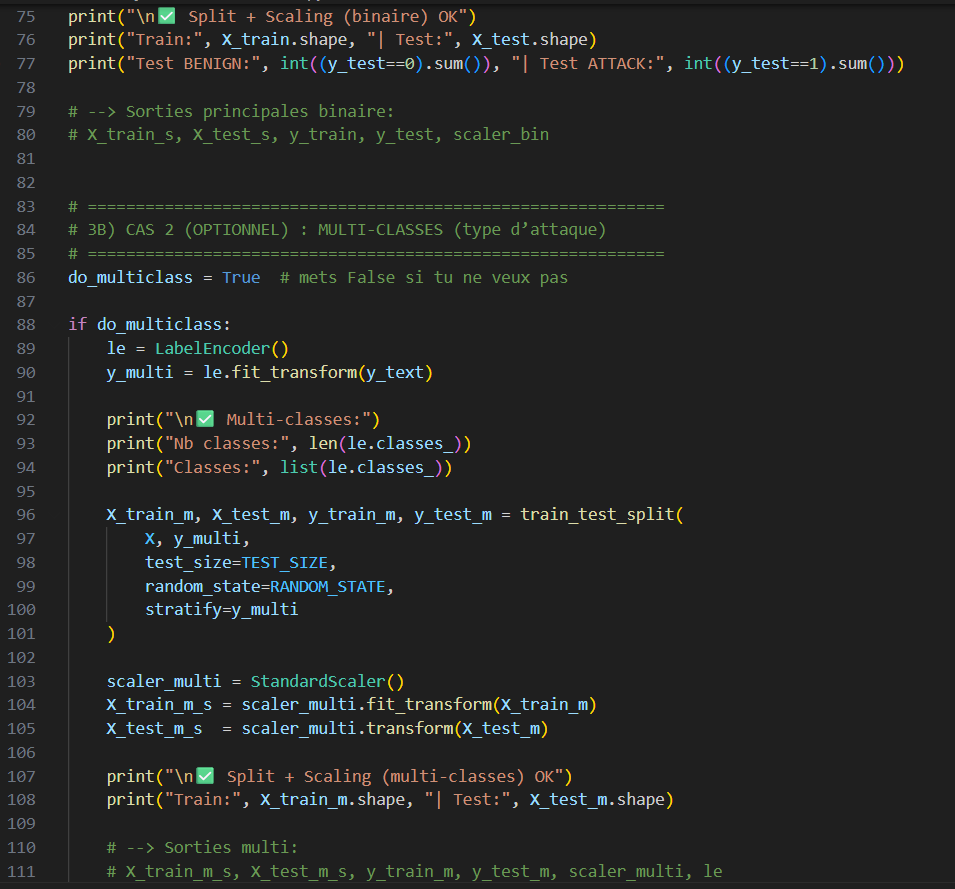
* 0 pour le trafic normal (BENIGN)
* 1 pour les attaques (Dos)

Cette étape est nécessaire car les algorithmes de Machine Learning utilisent uniquement des valeurs numériques.

Enfin, les données ont été séparées en deux parties :

* X : les caractéristiques réseau
* Y : la variable cible (attaque ou normal)

Ces données sont ensuite utilisées pour entraîner le modèle.



### Entraînement du modèle

Après la phase de prétraitement, le modèle de Machine Learning a été entraîné en utilisant la plateforme Azure Machine Learning avec le langage Python.

Les données ont été divisées en deux parties :

* 80% pour l’entraînement
* 20% pour le test

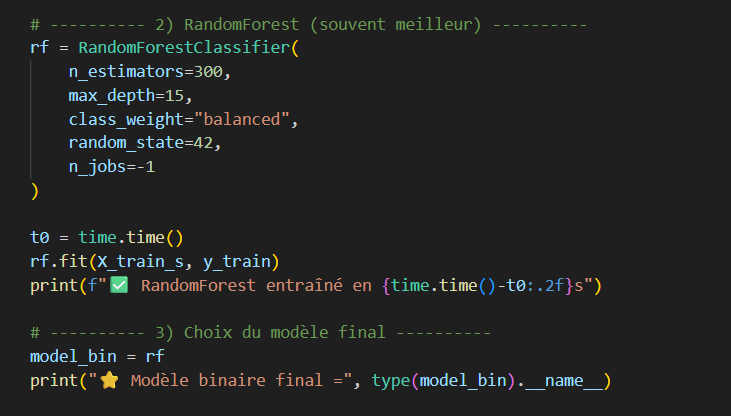
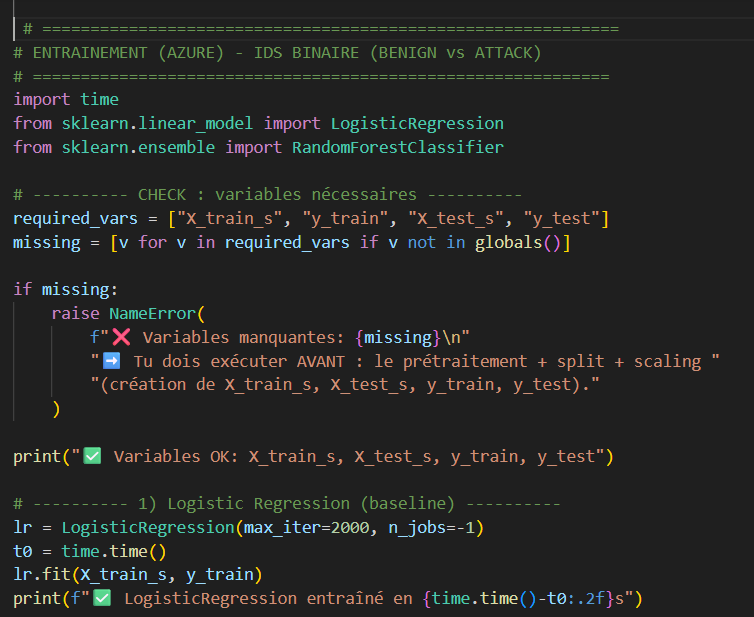
Cette étape permet d’entraîner le modèle et ensuite d’évaluer ses performances sur des données qu’il n’a jamais vues.

Ensuite, une étape de normalisation a été réalisée en utilisant StandardScaler.

La normalisation permet de mettre toutes les caractéristiques sur la même échelle afin d’améliorer la performance du modèle.

Le modèle utilisé est Random Forest Classifier, qui est un algorithme de Machine Learning supervisé basé sur un ensemble d’arbres de décision.

Enfin, la fonction model.fit() a été utilisée pour entraîner le modèle à apprendre à détecter les attaques à partir des données réseau.



### Matrice de confusion

La matrice de confusion est un outil utilisé pour évaluer les performances du modèle de détection d’intrusion. Elle permet de comparer les prédictions du modèle avec les vraies classes. La matrice est composée de quatre parties :

• Vrai négatif (True Negative) : trafic normal correctement détecté

• Faux positif (False Positive) : trafic normal détecté comme attaque

• Faux négatif (False Negative) : attaque détectée comme trafic normal

• Vrai positif (True Positive) : attaque correctement détectée

Les résultats obtenus montrent que le modèle est capable de détecter efficacement les attaques avec une très grande précision.

Le nombre élevé de prédictions correctes confirme la performance du modèle Random Forest utilisé dans ce projet.

### Simulation de détection en temps réel

Afin de valider le fonctionnement du système de détection d’intrusion dans des conditions proches du réel, une simulation de détection en temps réel a été mise en place sur la plateforme Azure Machine Learning.

Dans un réseau informatique réel, les données réseau ne sont pas analysées en une seule fois, mais arrivent sous forme de flux continus appelés flows. Chaque flow correspond à une communication entre deux machines, contenant différentes caractéristiques telles que la durée, le nombre de paquets, le nombre d’octets, et d'autres statistiques réseau.

Dans ce projet, nous avons simulé ce comportement en envoyant les flux réseau un par un au modèle de Machine Learning précédemment entraîné.

Concrètement, les données de test ont été utilisées pour créer un flux simulé. Chaque flow est analysé individuellement par le modèle Random Forest. Le modèle calcule alors la probabilité que ce flow corresponde à une attaque.

Ensuite, le modèle prend une décision :

* + Si la probabilité est supérieure à un seuil défini (par exemple 0.70), le flow est classé comme une attaque
  + Sinon, il est classé comme un trafic normal

Pour chaque flow analysé, le système affiche les informations suivantes :

* + L’index du flow analysé
  + La vraie classe (BENIGN ou ATTACK)
  + La prédiction du modèle
  + La probabilité d’attaque

*Exemple :*

[001] idx=52341 | TRUE=ATTACK | PRED=ATTACK | proba\_attack=0.98

Cette simulation permet de reproduire le comportement réel d’un système IDS, qui analyse le trafic réseau en continu afin de détecter les attaques dès leur apparition.

De plus, une fenêtre glissante a été utilisée afin de surveiller le taux d’attaques détectées sur une période récente. Si ce taux dépasse un seuil défini, une alerte est générée.

Cela permet de simuler le fonctionnement d’un centre de sécurité (SOC), capable de détecter une activité malveillante en temps réel.

Les résultats obtenus montrent que le modèle est capable de détecter efficacement les attaques réseau, avec une forte précision et une capacité de réaction rapide.

Cette simulation confirme que le système développé peut être utilisé dans un contexte réel pour la détection d’intrusions réseau.

# Tests et Validation

## Procédures de Test :

### Description des Scénarios de Test :

#### Chargement et vérification du dataset

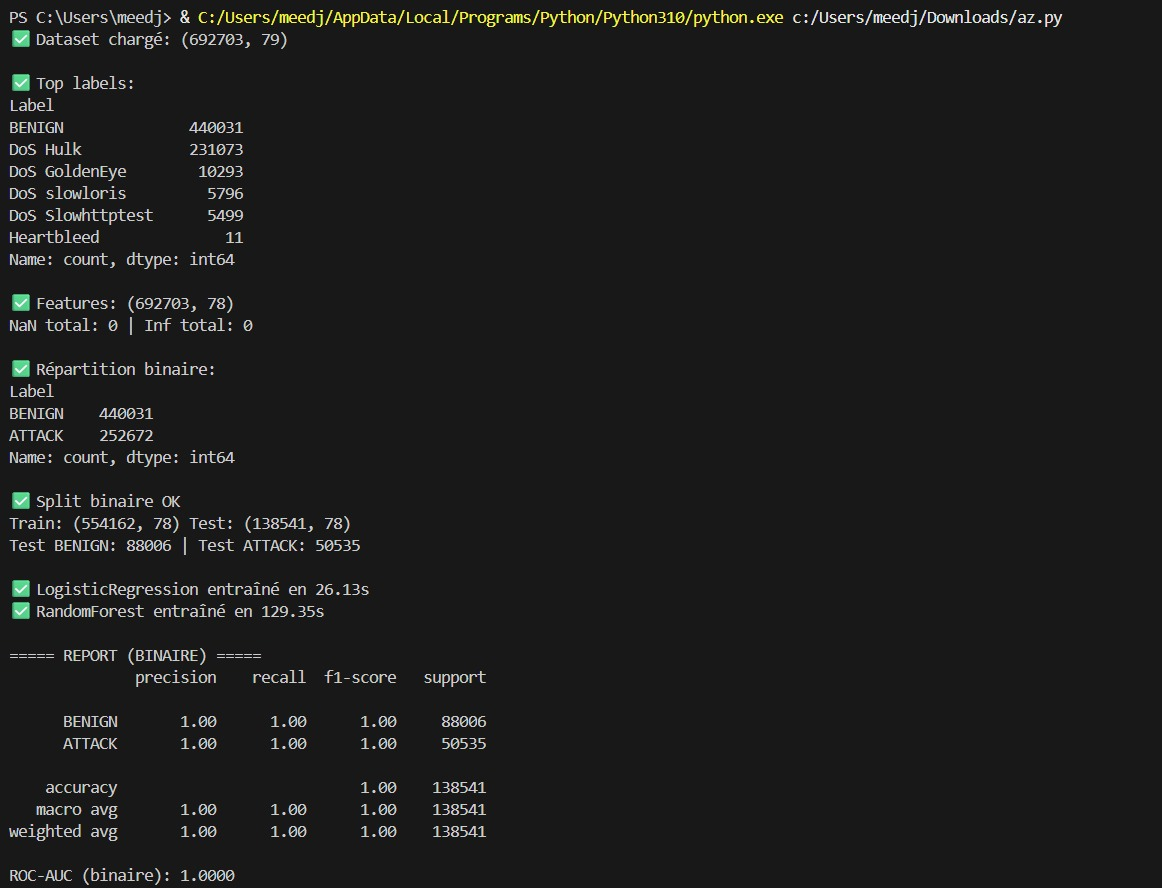
* Vérification de la présence de la colonne Label.
* Analyse de la distribution des classes.

#### Nettoyage des données

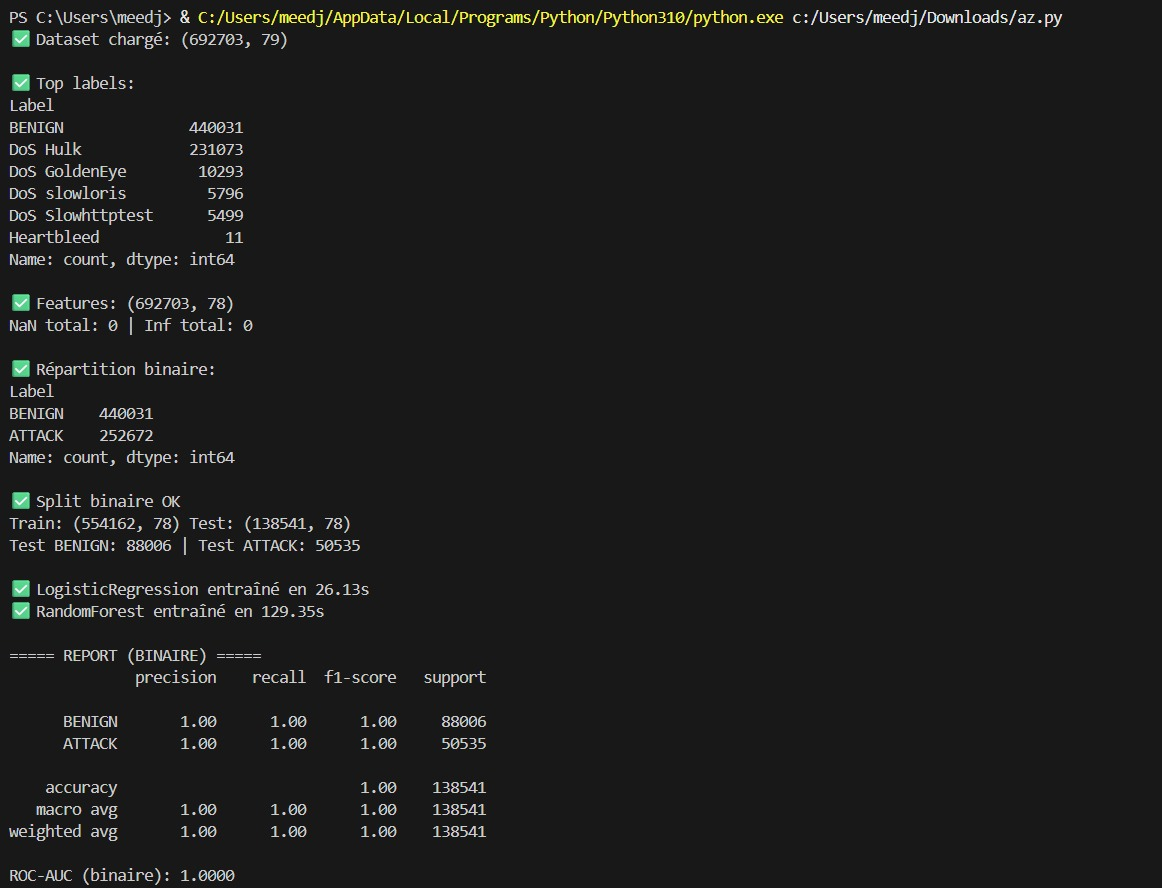
* Suppression des colonnes non numériques.
* Remplacement des valeurs infinies par NaN.
* Remplissage des valeurs manquantes par la médiane.
* Normalisation des données avec StandardScaler.

#### Séparation des données

* 80 % pour l’entraînement.
* 20 % pour le test.

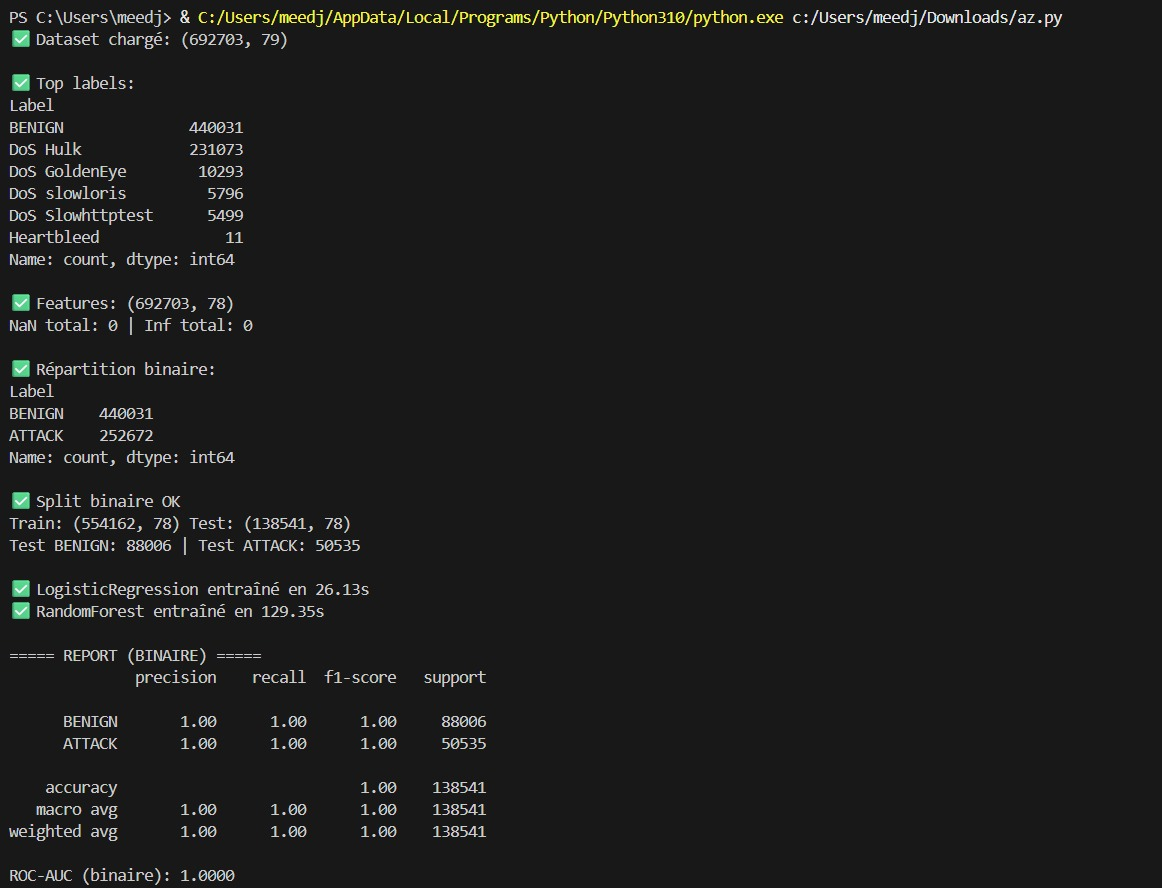
******

* Séparation stratifiée pour conserver la proportion BENIGN / ATTACK.

******

#### Entraînement des modèles

* Logistic Regression (modèle baseline).
* Random Forest (modèle principal retenu).

******

Cette capture montre les différentes étapes de préparation et d’entraînement du modèle de détection d’intrusion.

Tout d’abord, le dataset CICIDS2017 a été chargé avec succès. Il contient 692 703 flux réseau et 79 caractéristiques, ce qui constitue une base de données très riche pour entraîner un modèle de Machine Learning.

Ensuite, la répartition des classes a été affichée. On observe que le dataset contient principalement du trafic normal (BENIGN), ainsi que plusieurs types d’attaques comme DoS Hulk, DoS GoldenEye et Heartbleed.

Après la phase de prétraitement, les données ont été divisées en deux parties : 80 % pour l’entraînement et 20 % pour le test. Le modèle Random Forest a ensuite été entraîné avec succès.

Enfin, le rapport de classification montre que le modèle obtient :

**Précision : 1.00, Recall : 1.00, Accuracy : 1.00**

Cela signifie que le modèle est capable de détecter les attaques avec une très grande précision.

#### Évaluation des performances

* Accuracy, Precision, Recall, F1-score.
* Matrice de confusion.

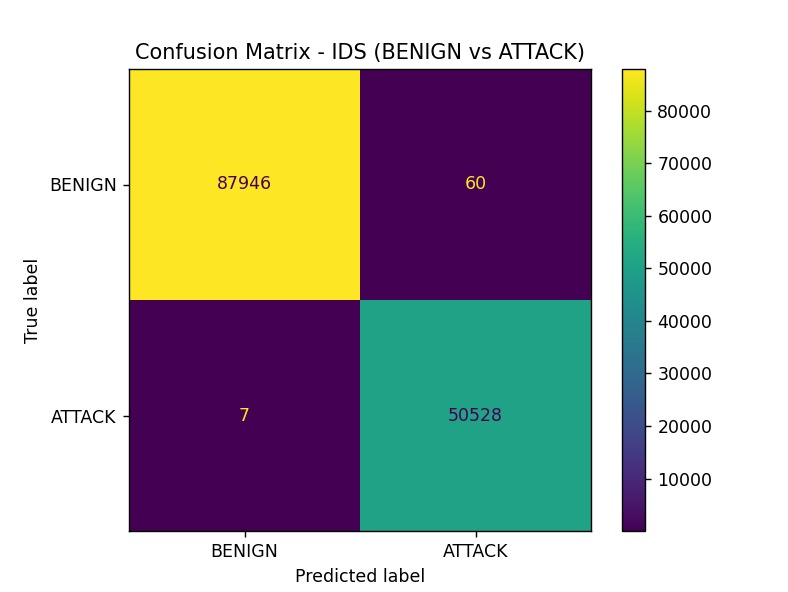
******

Figure 12 : Matrice de confusion du modèle de détection d’intrusion

Cette matrice de confusion illustre les performances du système pour classer le trafic réseau en deux catégories : BENIGN (trafic légitime) et ATTACK (attaques malveillantes). Parmi les 87 946 flux bénins, 87 946 sont correctement identifiés comme tels (vrais négatifs), avec seulement 60 faux positifs classés par erreur comme attaques ; de même, sur les 50 535 attaques, 50 528 sont détectées avec précision (vrais positifs), tandis que 7 seulement échappent à la vigilance (faux négatifs).

Ces chiffres traduisent **une exactitude globale de 99,86%**, **une précision de 99,88% et un rappel de 99,86%**, démontrant une fiabilité remarquable en cybersécurité : le modèle minimise efficacement les alertes superflues tout en capturant presque toutes les menaces réelles, ce qui en fait un outil robuste pour la protection des réseaux

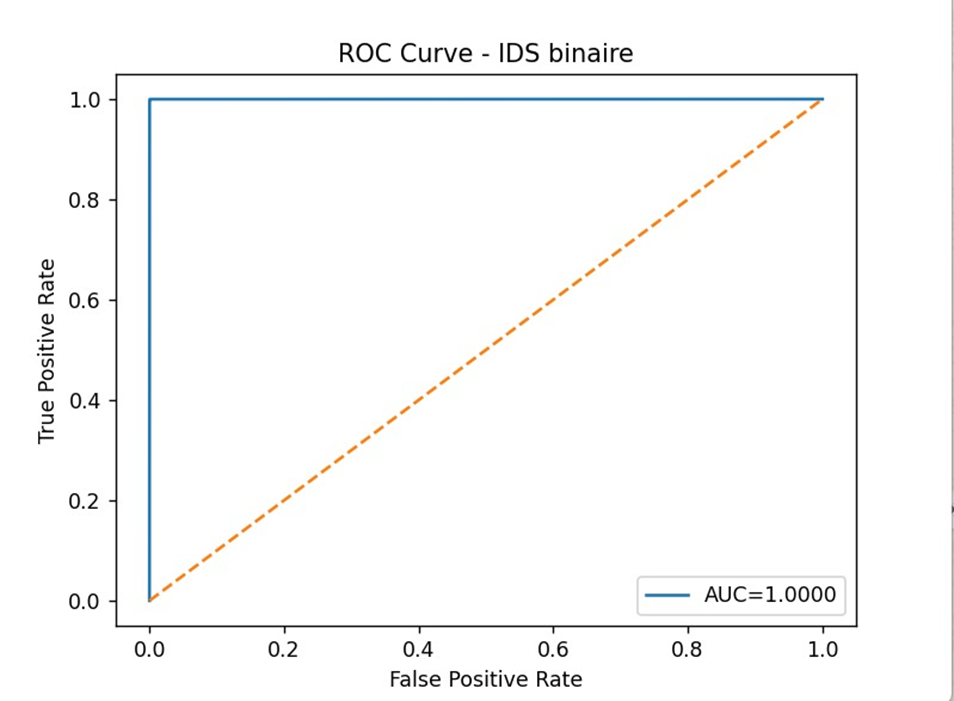
* Courbe ROC et score AUC.

Figure 13 : Courbe ROC du modèle IDS

La courbe ROC permet d’évaluer la capacité du modèle à distinguer le trafic normal des attaques.

On observe que la valeur AUC est égale à :

**AUC = 1.0000**

Cette valeur correspond à une performance parfaite.

Cela signifie que le modèle est capable de distinguer parfaitement les attaques du trafic normal.

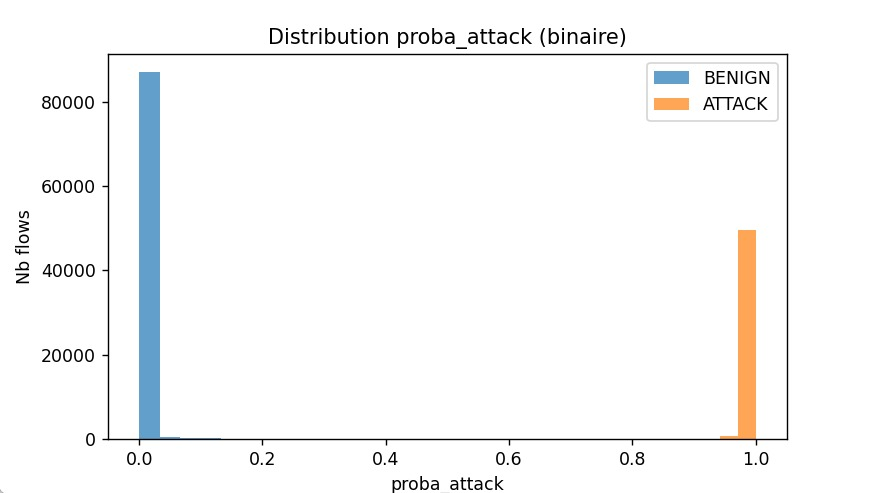
* ******Distribution des probabilités d’attaque.

Figure 14 : Distribution des probabilités de détection d’attaque

* **Une fois toutes ces étapes réalisées, nous obtenons ceci :**

#### Simulation temps réel

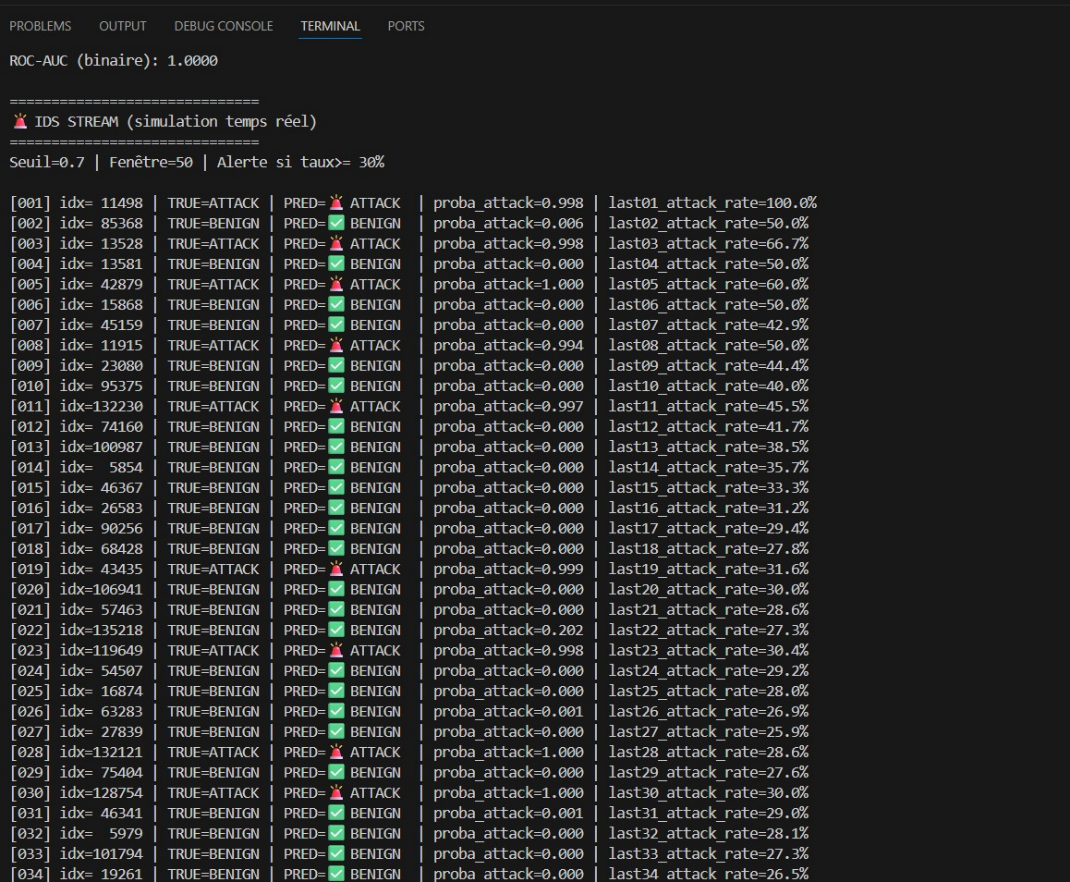
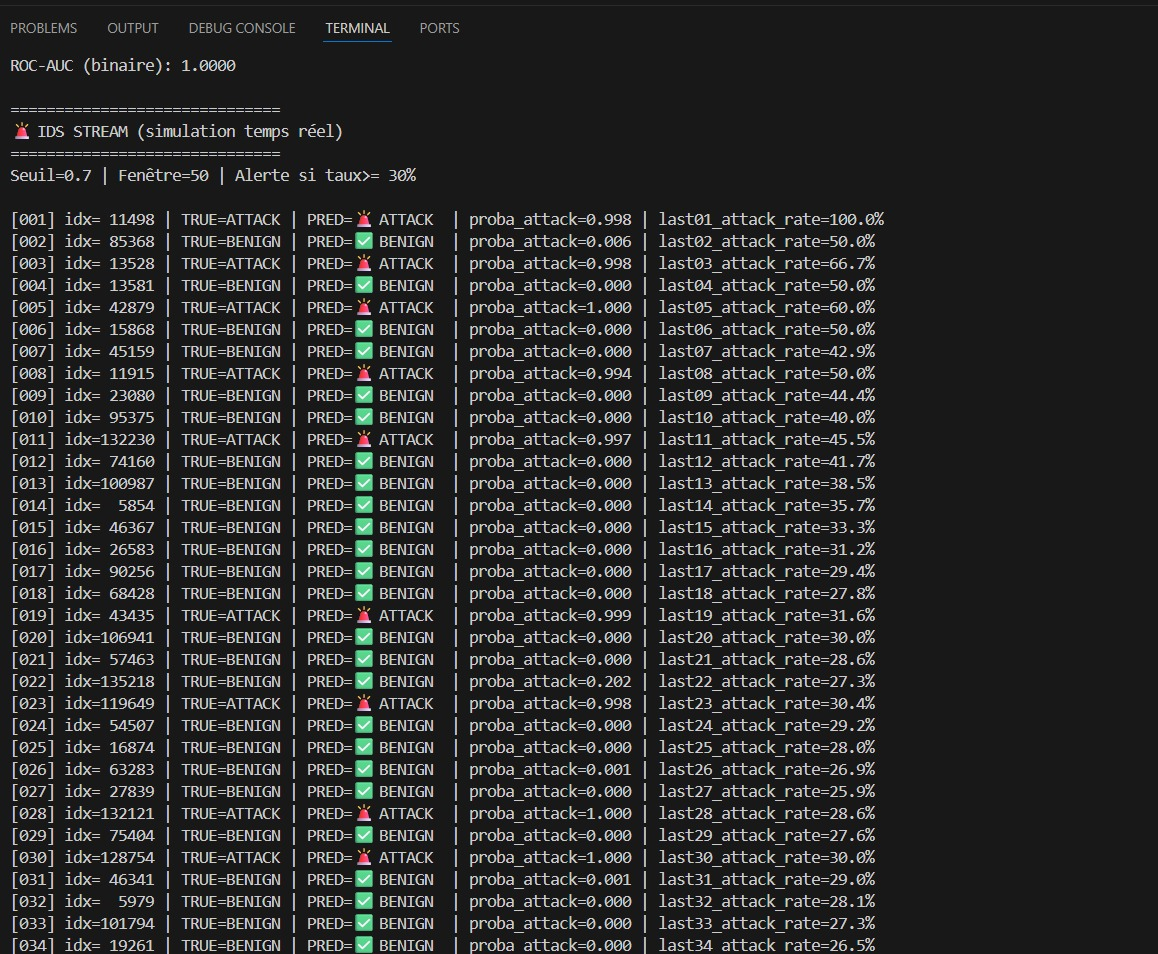
* ******Streaming de 200 flux réseau.

Figure 13 : Résultats du modèle multi-classes pour la classification des types d’attaques

*Précision : nous en avons capturé seulement 34 pour le rapport.*

* Seuil de décision : 0.70.
* Fenêtre glissante de 50 flux.
* Alerte SOC si taux d’attaque ≥ 30 %.

******

200 flux réseau ont été analysés de manière séquentielle afin de reproduire un comportement proche d’un environnement opérationnel (nous en avons capturé seulement 34). Un seuil de décision fixé à 0,70 a été appliqué pour classifier un flux comme malveillant, et une fenêtre glissante de 50 flux a été utilisée pour calculer dynamiquement le taux d’attaques récent. Une alerte SOC est déclenché lorsque ce taux atteint ou dépasse 30 %, simulant ainsi un mécanisme de supervision en centre opérationnel de sécurité. Les observations montrent que le modèle réagit correctement flux par flux, avec des probabilités d’attaque cohérentes par rapport aux étiquettes réelles. Lorsque la proportion d’attaques augmente dans la fenêtre d’analyse, le système génère automatiquement des alertes, démontrant sa capacité à détecter des pics d’activité malveillante. Cette approche confirme que la solution peut simuler efficacement le comportement d’un IDS en conditions quasi réelles et qu’elle est adaptée à une intégration dans un environnement de production.

#### Classification multi-classes

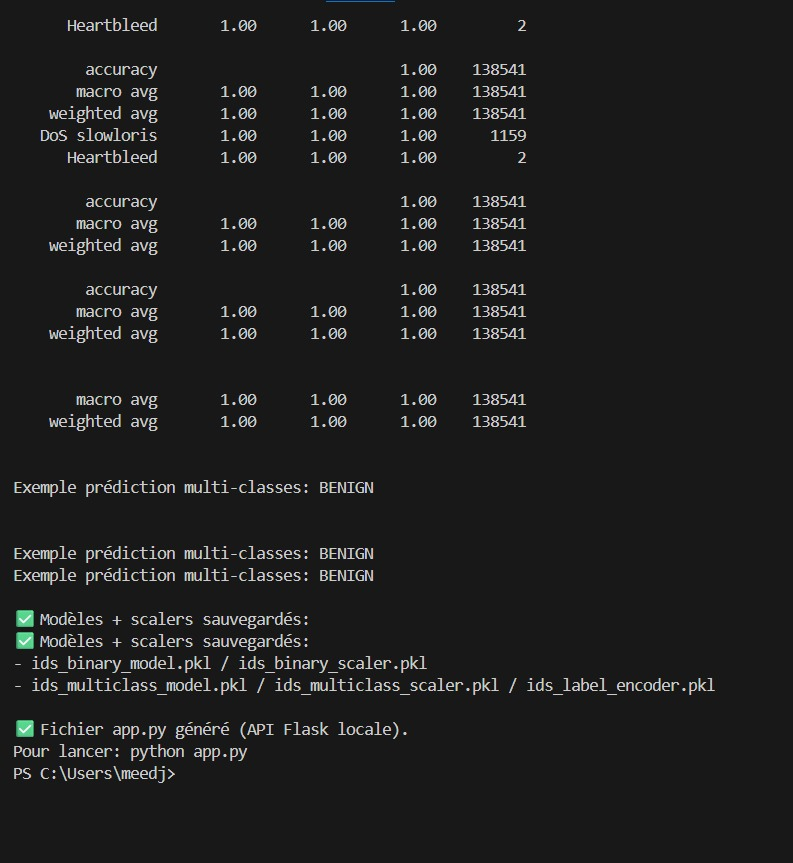
* ******Identification précise du type d’attaque.

Figure 15 : Classification du modèle multi-classes pour la classification des types d’attaques

* La distribution des probabilités d’attaque prédites par le modèle.

On observe deux groupes distincts :

* Les flux BENIGN ont une probabilité proche de 0.
* Les flux ATTACK ont une probabilité proche de 1.

Cela signifie que le modèle est capable de séparer clairement les attaques du trafic normal.

Cette séparation confirme l’efficacité du modèle.

* Évaluation via matrice de confusion multi-classes.

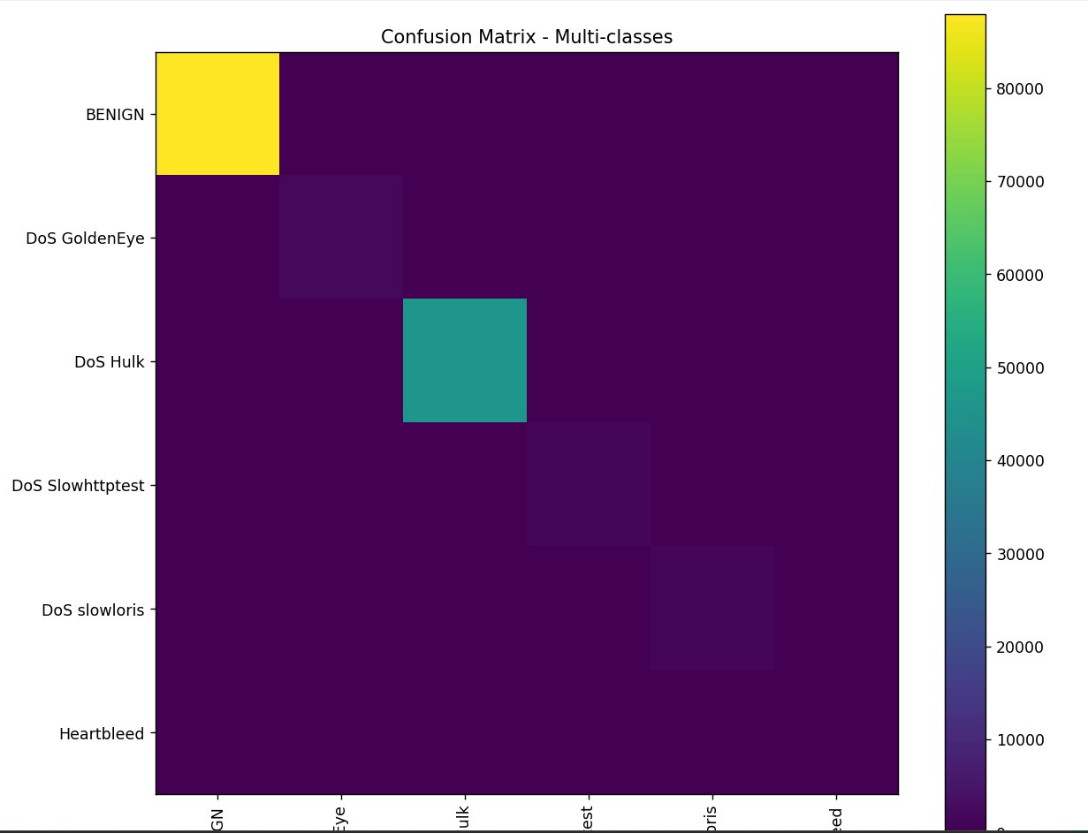
******

Figure 16 : Matrice de confusion en temps réel

Le scénario multi-classes vise à aller au-delà de la simple détection d’une attaque en identifiant précisément le type de menace associé à chaque flux réseau. À partir du dataset CICIDS2017, les différentes étiquettes d’attaques (DoS, PortScan, etc.) ont été encodées, puis un modèle Random Forest plus robuste a été entraîné afin de classifier chaque flux selon sa catégorie exacte. L’évaluation a été réalisée sur un jeu de test séparé, en analysant les métriques de précision, rappel et F1-score pour chaque classe, ainsi que la matrice de confusion multi-classes. Les résultats montrent de bonnes performances globales, notamment pour les classes majoritaires, avec une capacité satisfaisante à distinguer les différents types d’attaques. Toutefois, certaines classes moins représentées présentent des performances légèrement inférieures, ce qui s’explique par le déséquilibre des données. Ce scénario démontre que le système ne se limite pas à signaler une menace, mais qu’il peut également fournir une analyse plus fine et exploitable du type d’attaque détecté.

**Les tests ont été réalisés sur le dataset CICIDS2017 (Wednesday-workingHours), contenant du trafic réseau normal (BENIGN) et plusieurs types d’attaques, car les enregistrements des autres ne contiennent pas d’attaques.**

### Résultat des tests

L’objectif principal de ces scénarios est de vérifier que le modèle est capable de distinguer correctement un flux réseau normal d’un flux malveillant. Pour cela, un modèle Random Forest composé de 300 arbres avec une profondeur maximale fixée à 15 a été entraîné sur 80 % des données du dataset CICIDS2017, tandis que 20 % ont été réservés pour la phase de test afin d’évaluer ses performances sur des données jamais vues. Un seuil de décision de 0,70 a été appliqué sur la probabilité prédite afin de classer un flux comme attaque uniquement lorsque le niveau de confiance du modèle est suffisamment élevé.

Les résultats attendus incluent un taux de détection élevé des attaques (fort recall), un faible nombre de faux positifs afin d’éviter les alertes inutiles, ainsi qu’un score AUC proche de 1, traduisant une excellente capacité de discrimination entre les classes. Une bonne séparation entre les flux BENIGN et ATTACK dans les distributions de probabilités confirme également la robustesse du modèle dans des conditions normales d’évaluation.

## Conclusion des observations :

#### Performance optimale en conditions idéales

Les résultats obtenus montrent que le modèle développé présente d’excellentes performances dans des conditions expérimentales contrôlées. En effet, après la phase de préparation des données, de normalisation et d’entraînement, le modèle est capable de distinguer efficacement le trafic normal du trafic malveillant.

Les métriques d’évaluation, notamment l’accuracy, la précision, le recall et le score F1, indiquent un taux de détection très élevé, avec un faible taux de faux positifs et de faux négatifs. Cela confirme la capacité du modèle à apprendre les caractéristiques discriminantes des attaques présentes dans la base de données CICIDS2017.

Cependant, il est important de souligner que ces performances sont obtenues dans un environnement hors-ligne, sur des données déjà collectées et étiquetées.

### **Synthèse**

Les principales observations sont les suivantes :

* La phase de prétraitement des données joue un rôle essentiel dans la qualité des résultats
* Le fichier Wednesday-workingHours constitue une base pertinente car il contient à la fois du trafic normal et des attaques
* Les modèles de Machine Learning utilisés permettent une classification efficace
* Le système développé est capable de détecter les intrusions avec une grande précision
* Le modèle est actuellement adapté à une détection hors-ligne

Ainsi, ces résultats valident la pertinence de l’approche adoptée.

# Conclusion

Ce projet avait pour objectif de concevoir et de développer un système de détection d’intrusions basé sur des techniques de Machine Learning en utilisant la base de données CICIDS2017.

Les différentes étapes du projet ont permis de :

* Comprendre le fonctionnement des systèmes de détection d’intrusions
* Analyser et préparer les données réseau
* Entraîner et évaluer un modèle de classification
* Mesurer les performances du système

Les résultats obtenus sont très satisfaisants et démontrent l’efficacité des techniques de Machine Learning pour la détection d’attaques réseau.

## Bilan

Le bilan de ce projet est globalement positif.

Les principaux acquis sont :

#### Sur le plan technique :

* Manipulation de bases de données réseau
* Prétraitement et nettoyage des données
* Utilisation de Python et des bibliothèques Machine Learning (Scikit-learn)
* Entraînement et évaluation de modèles
* Analyse des performances

#### Sur le plan conceptuel :

* Compréhension du fonctionnement des attaques réseau
* Compréhension des systèmes IDS (Intrusion Detection System)
* Compréhension des méthodes de classification

#### Sur le plan pratique :

* Développement d’un système complet de détection
* Analyse et interprétation des résultats

## Perspectives

Plusieurs perspectives peuvent être envisagées pour améliorer ce projet.

Tout d’abord, il serait intéressant de tester d’autres modèles plus avancés, notamment les modèles de Deep Learning tels que :

* Les réseaux de neurones artificiels (ANN)
* Les réseaux de neurones profonds (Deep Neural Networks)
* Les réseaux LSTM

Ensuite, il serait pertinent d’évaluer le système sur d’autres bases de données afin de vérifier sa robustesse.

Enfin, l’objectif principal serait d’adapter ce système pour une utilisation en temps réel.

## Extensions du projet

Plusieurs extensions peuvent être envisagées :

#### Extension 1 : Détection en temps réel

* Transformer le système actuel en système capable de détecter les attaques en temps réel à partir du trafic réseau réel.

#### Extension 2 : Interface graphique

* Développer une interface graphique permettant de :
* Visualiser le trafic
* Afficher les attaques détectées
* Surveiller le réseau

#### Extension 3 : Utilisation du Deep Learning

* Améliorer les performances en utilisant :
* TensorFlow
* Keras
* PyTorch

#### Extension 4 : Déploiement

* Déployer le modèle sous forme :
* D’application
* D’API
* Intégration dans un système IDS réel

# Références

Bases de données et benchmarks

###### CICIDS2017

Sharafaldin, I., Lashkari, A.H., & Ghorbani, A.A. (2018).  
"Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization."  
International Conference on Information Systems Security and Privacy (ICISSP).

Algorithmes de détection

###### Random Forest

Breiman, L. (2001).  
"Random Forests."  
Machine Learning Journal.

###### Logistic Regression

Hosmer, D.W., Lemeshow, S., & Sturdivant, R.X. (2013).  
"Applied Logistic Regression."  
Wiley.

###### Deep Learning pour IDS

Kim, G., Lee, S., & Kim, S. (2014).  
"A Novel Hybrid Intrusion Detection Method Integrating Anomaly Detection with Misuse Detection."  
Expert Systems with Applications.

Applications et contextes spécifiques

###### DS en environnement cloud

Modi, C., Patel, D., Borisaniya, B., Patel, H., & Rajarajan, M. (2013).  
"A Survey of Intrusion Detection Techniques in Cloud."  
Journal of Network and Computer Applications.

###### IDS pour IoT

Meidan, Y., Bohadana, M., et al. (2018).  
"N-BaIoT: Network-based Detection of IoT Botnet Attacks Using Deep Autoencoders."  
IEEE Pervasive Computing.

Outils et frameworks

###### Azure Machine Learning

Microsoft.  
"Azure Machine Learning Documentation."  
Documentation officielle Microsoft.

# Annexe

<https://github.com/zak-soft/Network-Intrusion-Detection-System-NIDS.git>