

Université Hassan 2
ECOLE NORMALE SUPÉRIEURE DE CASABLANCA

Rapport de min projet

Sous le Thème :

Système de détection d'intrusion

Réalisé par : Boujrada Yassine

Année Universitaire : 2024-2025

TABLE DES MATIÈRES

Résumé	3
Introduction générale	4
1 CONTEXTE GÉNÉRALE SUR IDS	5
1.1 Introduction	6
1.2 Qu'est-ce que systèmes de détection d'intrusion (IDS)	6
1.3 Types de systèmes de détection d'intrusion (IDS)	6
1.4 Méthode de détection de IDS	7
1.5 Conclusion	8
2 Exporter DataSet	9
2.1 Introduction	10
2.2 Les informations sur les données	10
2.3 Data Visualization	13
2.4 Data Correlation	15
2.5 Conclusion	17
3 Model D'apprentissage	18
3.1 Introduction	19
3.2 Fractionnement des données	19
3.2.1 Train Test Split	20
3.2.2 Cross Validation ou k-Folds	21
3.3 Modeling	22
3.3.1 Algorithmes d'apprentissage automatique	22
3.3.1.1 Phase Entraînement	22
3.3.1.2 Phase Evaluation	26
3.3.2 Deep Learning :	28
3.4 Conclusion	33
Conclusion	34

TABLE DES FIGURES

1.1	Schéma de IDS/IP	6
2.1	Informations sur les colonnes du dataset	10
2.2	les valeurs nulles en dataset	11
2.3	Les types d'attaques en dataset	12
2.4	Les outcomes en dataset	12
2.5	graphique des outcomes en dataset	13
2.6	les types de protocoles et sortie en dataset	14
2.7	les types de protocoles en fonction d'attaque	14
2.8	logged in column	15
2.9	exemple de Label Encoder	16
2.10	Correlation de chaque attribut avec les autres attributs	17
3.1	Séparer les attributs	19
3.2	Schéma de train test split	20
3.3	implémentation de train test split en notre code	20
3.4	Schéma de Cross Validation	21
3.5	implémentation de Cross Validation en notre code	22
3.6	formule mathématique de régression logistique	23
3.7	Schéma de k plus proches voisins	23
3.8	Schéma d'arbre de décision	24
3.9	Schéma de Random Forest	24
3.10	initialiser chaque algorithme	25
3.11	Entraînement ces algorithme par train test datasplit	25
3.12	Entraînement ces algorithme par Cross Validation méthode	25
3.13	Evaluation de ces algorithme	26
3.14	Métriques d'évaluation pour les modèles formés à l'aide de Train-Test Split	26
3.15	Matrice de confusion	27
3.16	Comparaison de Accuracy : Trian Test Split vs Cross Validation	28
3.17	Exemple d'un réseau de neurone	29
3.18	Définir notre modèle DL	29
3.19	Compile le modèle	30
3.20	Architecture de modèle	30
3.21	Entraînement notre DL modèle	31
3.22	Epochs notre DL modèle	31
3.23	Epochs en fonction de SCCE Loss	32
3.24	Epochs en fonction d'accuracy	32
3.25	code de sauvegarde le modèle	32
3.26	path de sauvegarde	33

RÉSUMÉ

Dans le cadre de notre parcours pour obtenir d'un diplôme de master en Cryptography et Cybersecurity, nous avons dû réaliser un mini projet pour ramener tous les concepts qui ont déjà été mis en œuvre .

Ce projet portait sur un système de détection d'intrusion, une application essentielle dans le domaine de la sécurité informatique.

Au début, je me suis concentré sur l'auto-formation sur les outils et technologies nécessaires, tels que les frameworks comme tensorflow et keras. Cette phase m'a permis de me familiariser avec les exigences du projet et de contribuer efficacement à la conception et à la mise en œuvre de la logique ainsi qu'à l'architecture de la base de données.

INTRODUCTION GÉNÉRALE

La conception d'applications pour gérer les processus est devenue cruciale pour les entreprises sur le marché actuel du développement de logiciels, donnant aux administrateurs la possibilité de suivre et d'améliorer les performances de l'entreprise et du personnel.

Simultanément, les développeurs utilisent des principes de test pour détecter et corriger les comportements logiciels problématiques, améliorant ainsi la qualité des produits.

Le projet décrit ici se concentre sur le développement d'un modèle de machine learning pour un système de détection d'intrusion, ou IDS, et compare différents algorithmes de machine learning, qui constituent un élément essentiel de la sécurité informatique. Pour permettre aux administrateurs de prendre des mesures préventives ou correctives, les systèmes de détection d'intrusion (IDS) sont conçus pour reconnaître et répondre aux activités suspectes ou malveillantes sur un réseau informatique.

Ce rapport permet d'avoir une présentation exhaustive de la démarche conduite afin de mettre à bien ce travail. Le présent rapport est structuré comme suit :

Chapitre 1 : Il porte sur le contexte général et la problématique du travail réalisé.

Chapitre 2 : Il appuie sur exportation, l'importation, la manipulation et la visualisation des données.

Chapitre 3 : Il présente la partie d'apprentissage et evaluation les résultat de modèle.

Au terme de ce rapport une conclusion avec perspectives est donnée qui synthétise tout le travail réalisé

CHAPTER 1

CONTEXTE GÉNÉRALE SUR IDS

1.1 Introduction

Ce chapitre sera une présentation du contexte générale ainsi que la problématique abordée.

1.2 Qu'est-ce que systèmes de détection d'intrusion (IDS)

Un système de détection des intrusions (IDS) [4] est une application qui surveille le trafic réseau et recherche les menaces connues et les activités suspectes ou malveillantes. L'IDS envoie des alertes aux équipes informatiques et de sécurité lorsqu'il détecte des risques et des menaces de sécurité. La plupart des solutions IDS surveillent et signalent simplement les activités et le trafic suspects lorsqu'elles détectent une anomalie.

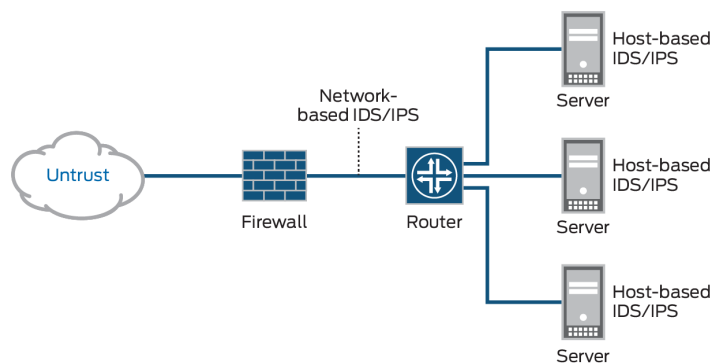


FIGURE 1.1 – Schéma de IDS/IP

Les outils IDS sont généralement des applications logicielles qui s'exécutent sur le matériel des organisations ou en tant que solution de sécurité réseau. Il existe également des solutions IDS basées sur le cloud qui protègent les données, les ressources et les systèmes des organisations dans leurs environnements et déploiements cloud.

1.3 Types de systèmes de détection d'intrusion (IDS)

Les types courants de systèmes de détection des intrusions (IDS) comprennent :

- Système de détection des intrusions réseau (NIDS) : une solution NIDS est déployée à des points stratégiques du réseau d'une organisation pour surveiller le trafic entrant et sortant. Cette approche IDS surveille et détecte le trafic malveillant et suspect entrant et sortant de tous les appareils connectés au réseau.

- Système de détection d'intrusion hôte (HIDS) : un système HIDS est installé sur des appareils individuels connectés à Internet et au réseau interne d'une organisation. Cette solution peut détecter les paquets provenant de l'intérieur de l'entreprise et le trafic malveillant supplémentaire qu'une solution NIDS ne peut pas détecter.
- Système de détection des intrusions (SIDS) basé sur les signatures : une solution SIDS surveille tous les paquets sur le réseau d'une organisation et les compare aux signatures d'attaque sur une base de données de menaces connues.
- Système de détection des intrusions (SIDA) basé sur les anomalies : cette solution surveille le trafic sur un réseau et le compare à une ligne de base prédéfinie considérée comme « normale ». Il détecte les activités et comportements anormaux sur le réseau, y compris la largeur de bande appareils, les ports et les protocoles.

1.4 Méthode de détection de IDS

Méthode basée sur les signatures : IDS basé sur les signatures détecte les attaques sur la base de modèles spécifiques tels que le nombre d'octets ou le nombre de 1 ou le nombre de 0 dans le trafic réseau. Il détecte également sur la base de la séquence d'instructions malveillantes déjà connue et utilisée par le logiciel malveillant. Les modèles détectés dans l'IDS sont appelés signatures. L'IDS basé sur les signatures peut facilement détecter les attaques dont le modèle (signature) existe déjà dans le système, mais il est assez difficile de détecter les nouvelles attaques de logiciels malveillants car leur modèle (signature) n'est pas connu.

Méthode basée sur les anomalies : l'IDS basé sur les anomalies a été introduit pour détecter les attaques de logiciels malveillants inconnus à mesure que de nouveaux logiciels malveillants se développent rapidement. Dans l'IDS basé sur les anomalies, l'apprentissage automatique est utilisé pour créer un modèle d'activité fiable et tout ce qui arrive est comparé à ce modèle et est déclaré suspect s'il n'est pas trouvé dans le modèle. La méthode basée sur l'apprentissage automatique a une propriété mieux généralisée par rapport à l'IDS basé sur les signatures, car ces modèles peuvent être formés en fonction des applications et des configurations matérielles.

1.5 Conclusion

Au cours de ce chapitre, j'ai présenté le contexte général sur le topic, les types, et méthode de détection qui selon on peut utiliser pour notre modèle.

CHAPTER2

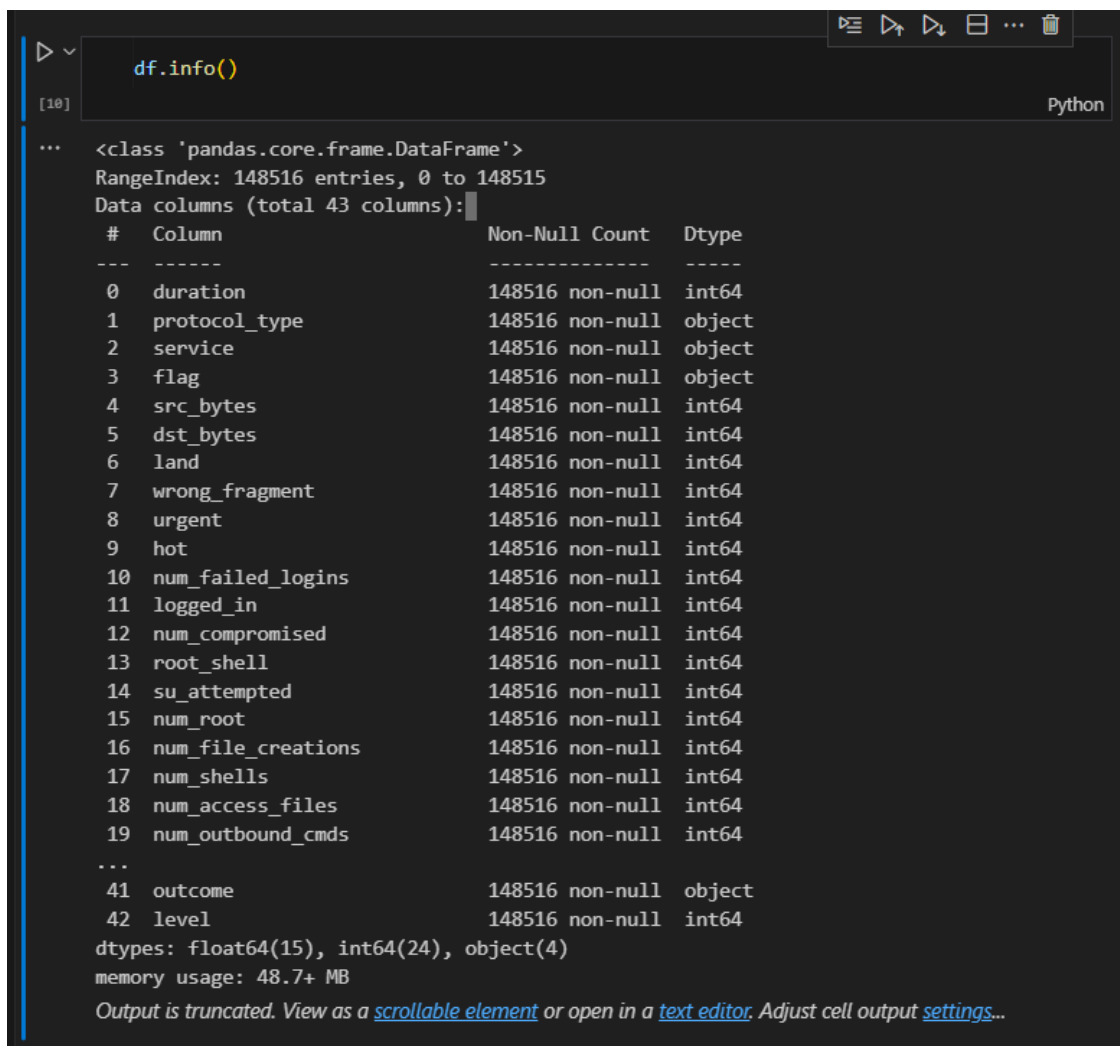
EXPORTER DATASET

2.1 Introduction

Dans le domaine du Machine Learning, l'efficacité de votre modèle dépend de la qualité de vos jeux de données (DataSet). Ainsi, l'exportation du DataSet est une étape cruciale. Pour cela, il existe des bibliothèques utilisées pour manipuler les données, et dans notre cas, nous avons utilisé Pandas. Pour notre DataSet, nous avons utilisé dans ce projet un ensemble de données provenant de [Kaggle](#), d'une taille de 1,2 Go. Voici un exemple d'échantillon : Exportation de DataSet de taille 55.87 MB.

2.2 Les informations sur les données

Pour obtenir des informations sur nos données, nous commençons par importer le fichier, puis nous utilisons la fonction ".info()".



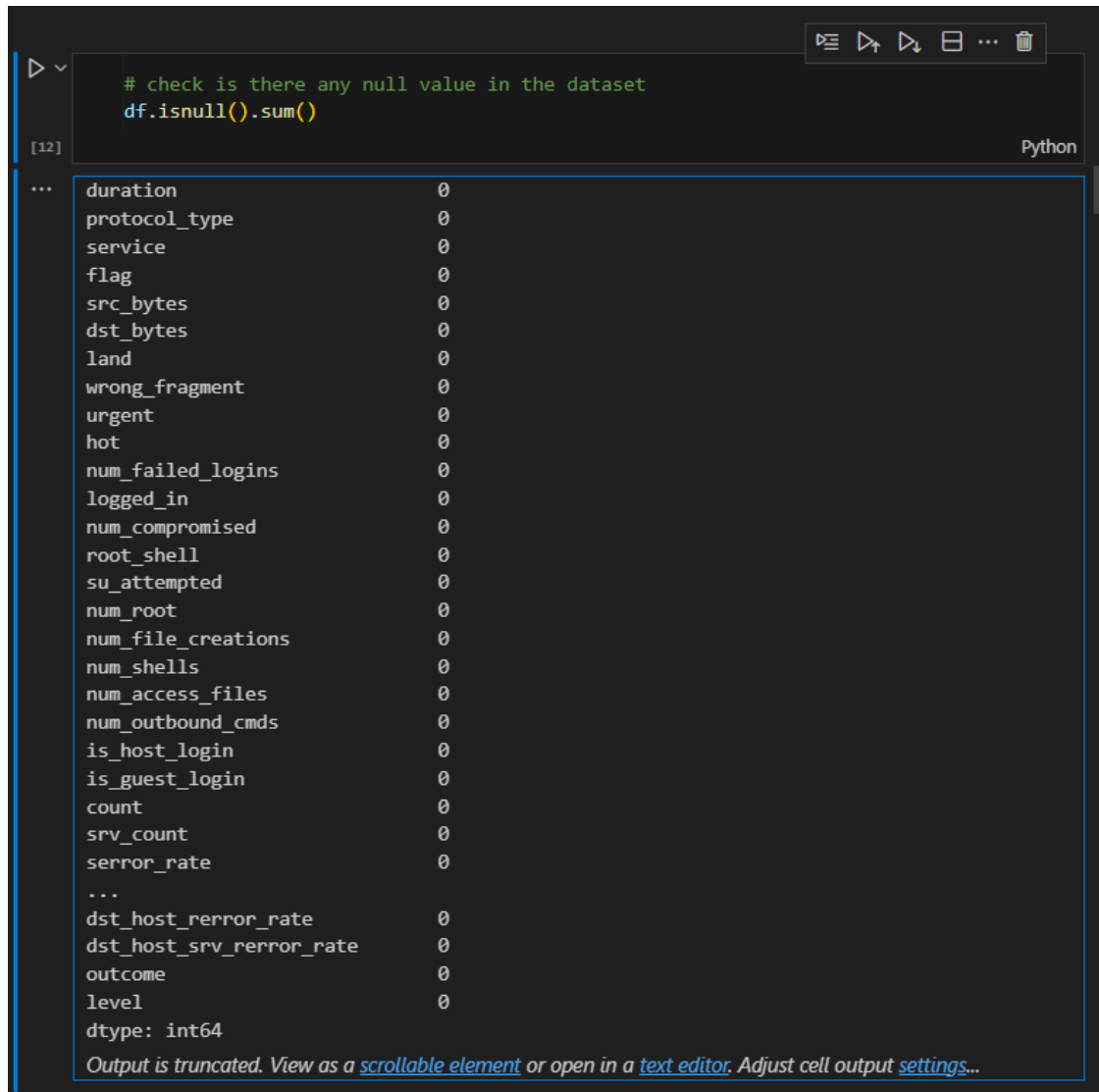
```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 148516 entries, 0 to 148515
Data columns (total 43 columns):
 #   Column                Non-Null Count  Dtype
---  -
 0   duration              148516 non-null  int64
 1   protocol_type         148516 non-null  object
 2   service               148516 non-null  object
 3   flag                 148516 non-null  object
 4   src_bytes             148516 non-null  int64
 5   dst_bytes             148516 non-null  int64
 6   land                 148516 non-null  int64
 7   wrong_fragment       148516 non-null  int64
 8   urgent               148516 non-null  int64
 9   hot                  148516 non-null  int64
10  num_failed_logins    148516 non-null  int64
11  logged_in            148516 non-null  int64
12  num_compromised      148516 non-null  int64
13  root_shell          148516 non-null  int64
14  su_attempted         148516 non-null  int64
15  num_root             148516 non-null  int64
16  num_file_creations   148516 non-null  int64
17  num_shells           148516 non-null  int64
18  num_access_files     148516 non-null  int64
19  num_outbound_cmds    148516 non-null  int64
...
41  outcome              148516 non-null  object
42  level                148516 non-null  int64
dtypes: float64(15), int64(24), object(4)
memory usage: 48.7+ MB

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

FIGURE 2.1 – Informations sur les colonnes du dataset

Ensuite, pour déterminer s'il existe des valeurs nulles à éliminer, nous utilisons la fonction `".isnull().sum()"`.



```
# check is there any null value in the dataset
df.isnull().sum()
```

duration	0
protocol_type	0
service	0
flag	0
src_bytes	0
dst_bytes	0
land	0
wrong_fragment	0
urgent	0
hot	0
num_failed_logins	0
logged_in	0
num_compromised	0
root_shell	0
su_attempted	0
num_root	0
num_file_creations	0
num_shells	0
num_access_files	0
num_outbound_cmds	0
is_host_login	0
is_guest_login	0
count	0
srv_count	0
serror_rate	0
...	
dst_host_rerror_rate	0
dst_host_srv_rerror_rate	0
outcome	0
level	0
dtype: int64	

Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output [settings](#)...

FIGURE 2.2 – les valeurs nulles en dataset

Ensuite, pour les types d'attaques que nous devons traiter dans nos données,

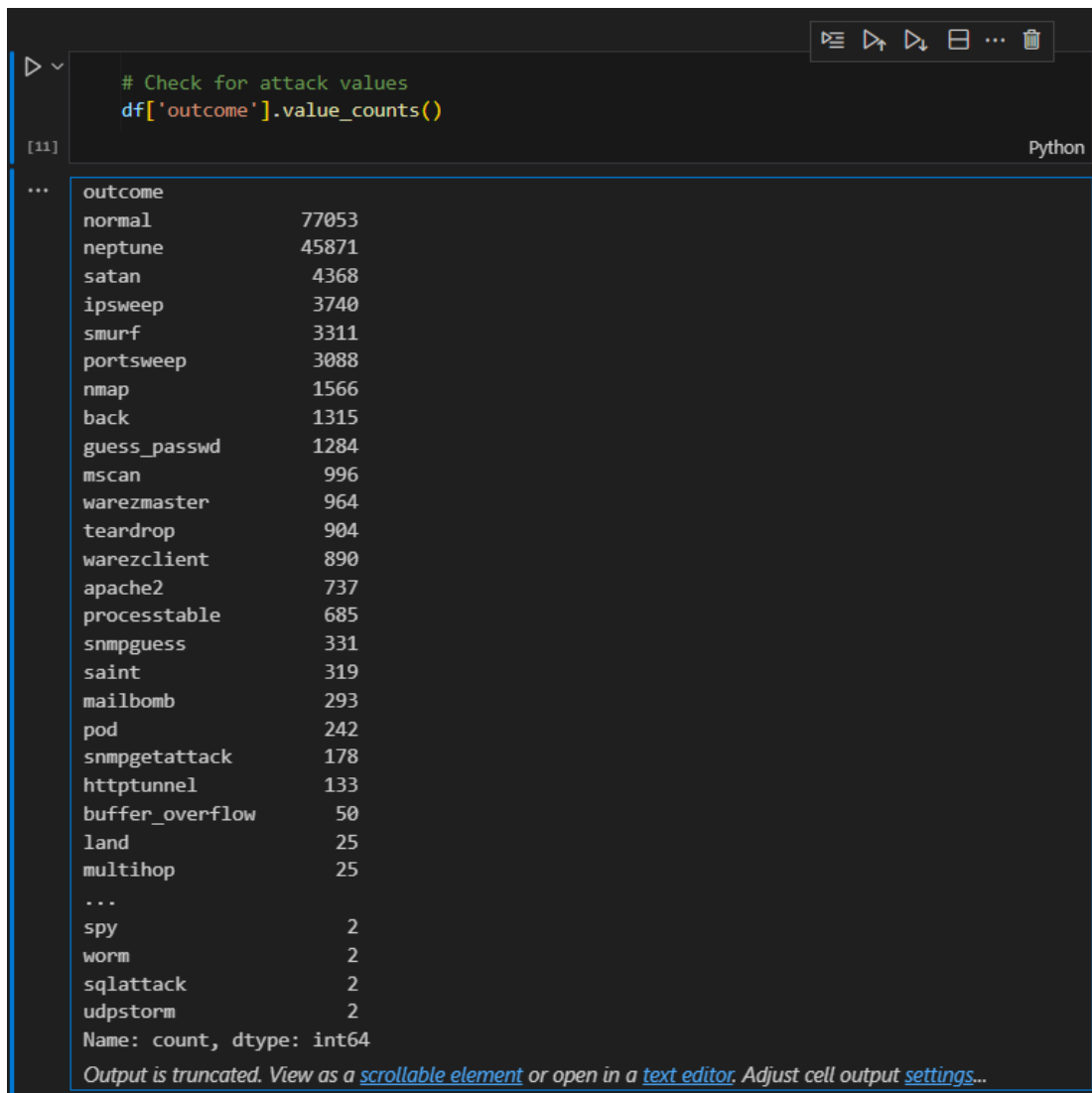


FIGURE 2.3 – Les types d’attaques en dataset

Comme nous l’observons dans la figure 2.3, il existe plusieurs types d’attaques. Afin de simplifier le travail, nous pouvons classifier tous ces résultats en deux labels : "normale" et "attaque".

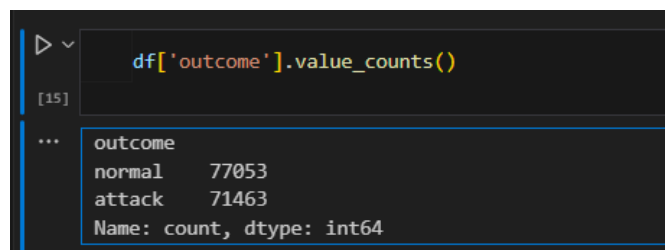


FIGURE 2.4 – Les outcomes en dataset

Pour une meilleure visualisation, on utilise Matplotlib pour afficher les données sous forme de graphiques.

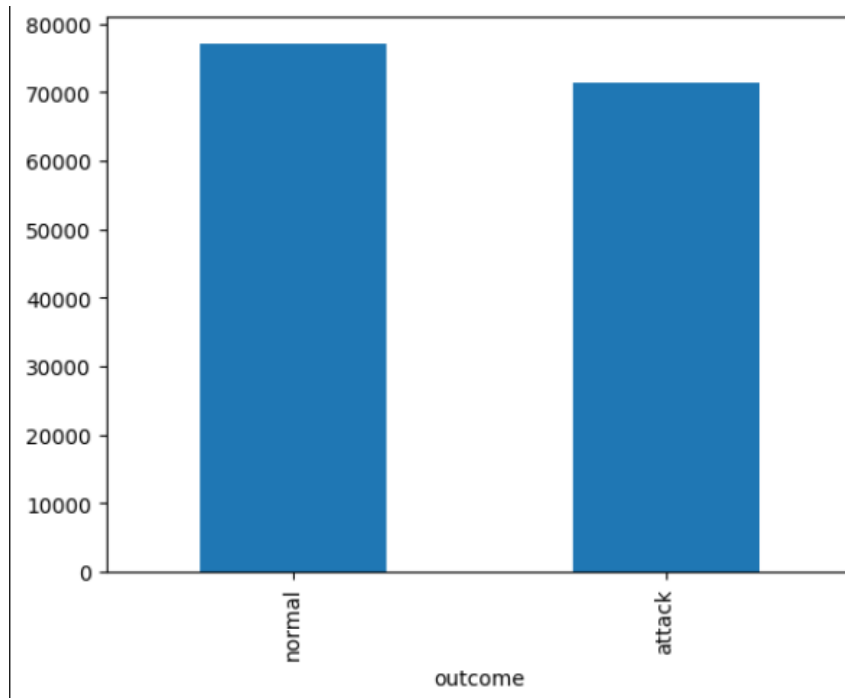


FIGURE 2.5 – graphique des outcomes en dataset

2.3 Data Visualization

La visualisation des données[3] est une composante essentielle de l'analyse et de la compréhension des données dans de nombreux domaines, notamment la science des données, la recherche, les affaires et plus encore.

Nous commençons par examiner le graphique ci-dessus, qui est un diagramme circulaire contenant deux sous-graphiques distincts. Le premier sous-graphique résume l'utilisation des protocoles, en mettant en évidence les nombres relatifs de protocoles tels que TCP, UDP et ICMP. Le deuxième sous-graphique vise à évaluer l'équilibre des valeurs de sortie (outcomes) afin de prévenir les problèmes potentiels lors de la phase d'entraînement, comme overfitting

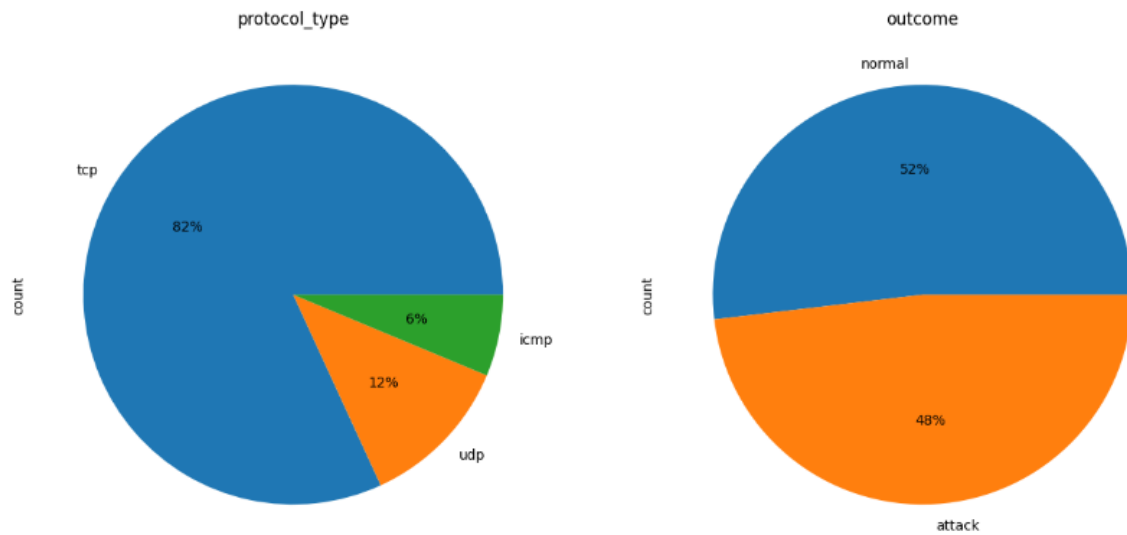


FIGURE 2.6 – les types de protocoles et sortie en dataset

Pour visualiser la répartition des différentes attaques en fonction du type de protocole dans l'ensemble de données, on a

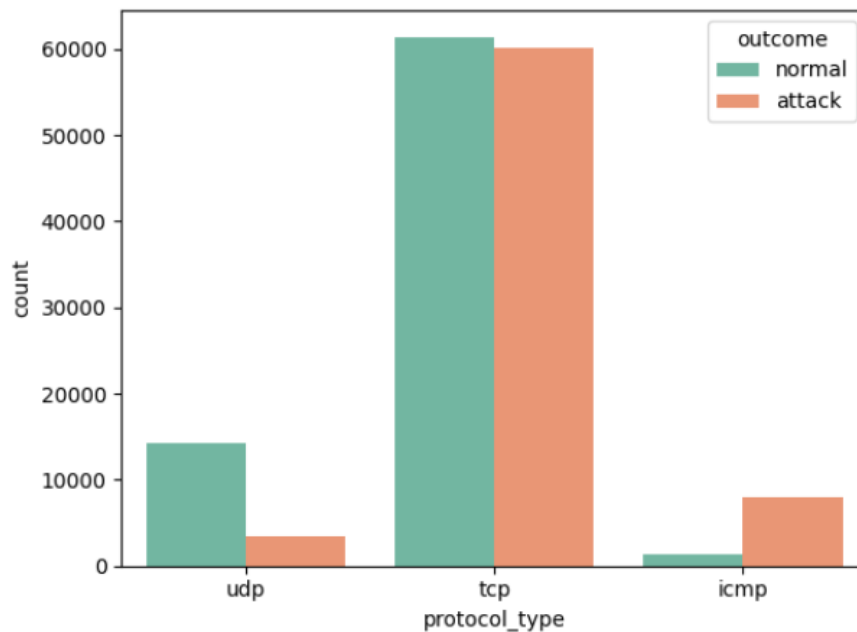


FIGURE 2.7 – les types de protocoles en fonction d'attaque

Il y'a un attribut particulièrement en notre dataset qui est interessant "logged in" car il indique si un utilisateur est actuellement connecté à un système ou à une plateforme. Cette information est souvent utilisée pour suivre et analyser le comportement des utilisateurs, notamment dans les systèmes informatiques, les applications en ligne

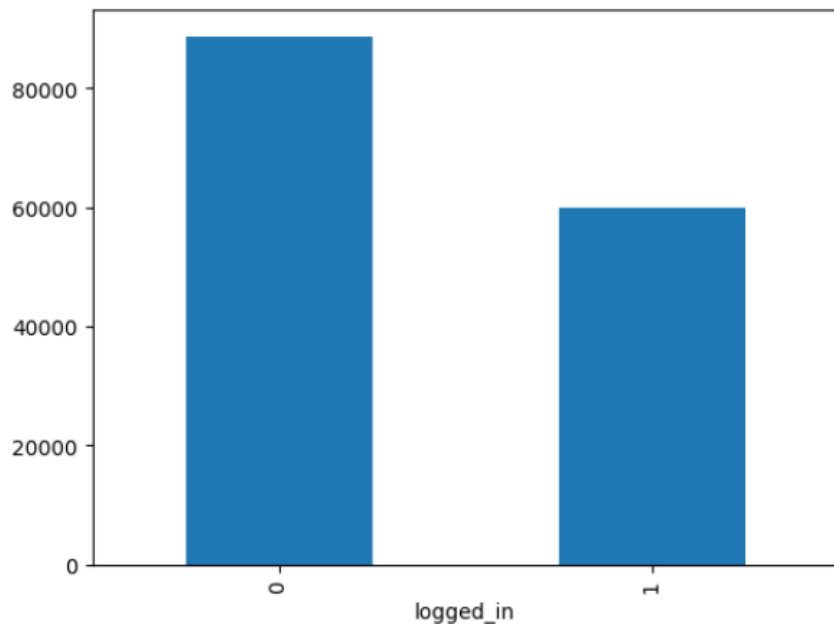


FIGURE 2.8 – logged in column

2.4 Data Correlation

Pour simplifier [5], il s'agit de la mesure dans laquelle une variable change par rapport à une autre. Il mesure la force et la direction de la relation entre deux ou plusieurs variables nous allons vérifier la corrélation entre les variables de notre ensemble de données pour voir lesquelles ne peuvent pas être utilisées dans notre modèle.

- 1 : indique une relation linéaire positive parfaite.
- 0 : indique aucune relation linéaire.
- -1 : indique une relation linéaire négative parfaite

Mais après avoir travaillé sur cet attribut, nous avons besoin d'utiliser des méthodes pour encoder les variables qui sont de type chaîne de caractères. Dans notre ensemble de données, nous avons des attributs tels que "service", "flag", "type de protocole" et "outcome". Pour cela, nous utilisons le labelEncoder, qui est une technique largement utilisée pour attribuer des valeurs numériques aux variables de catégorie.

Original Data		Label Encoded Data	
Team	Points	Team	Points
A	25	0	25
A	12	0	12
B	15	1	15
B	14	1	14
B	19	1	19
B	23	1	23
C	25	2	25
C	29	2	29

FIGURE 2.9 – exemple de Label Encoder

Chaque catégorie de la variable reçoit un identifiant numérique distinct par labelEncoder, permettant une représentation numérique des catégories dans notre analyse. Alors après avoir calculé la corrélation de chaque attribut de notre ensemble de données avec les autres, nous obtenons comme résultat final ce graphique.

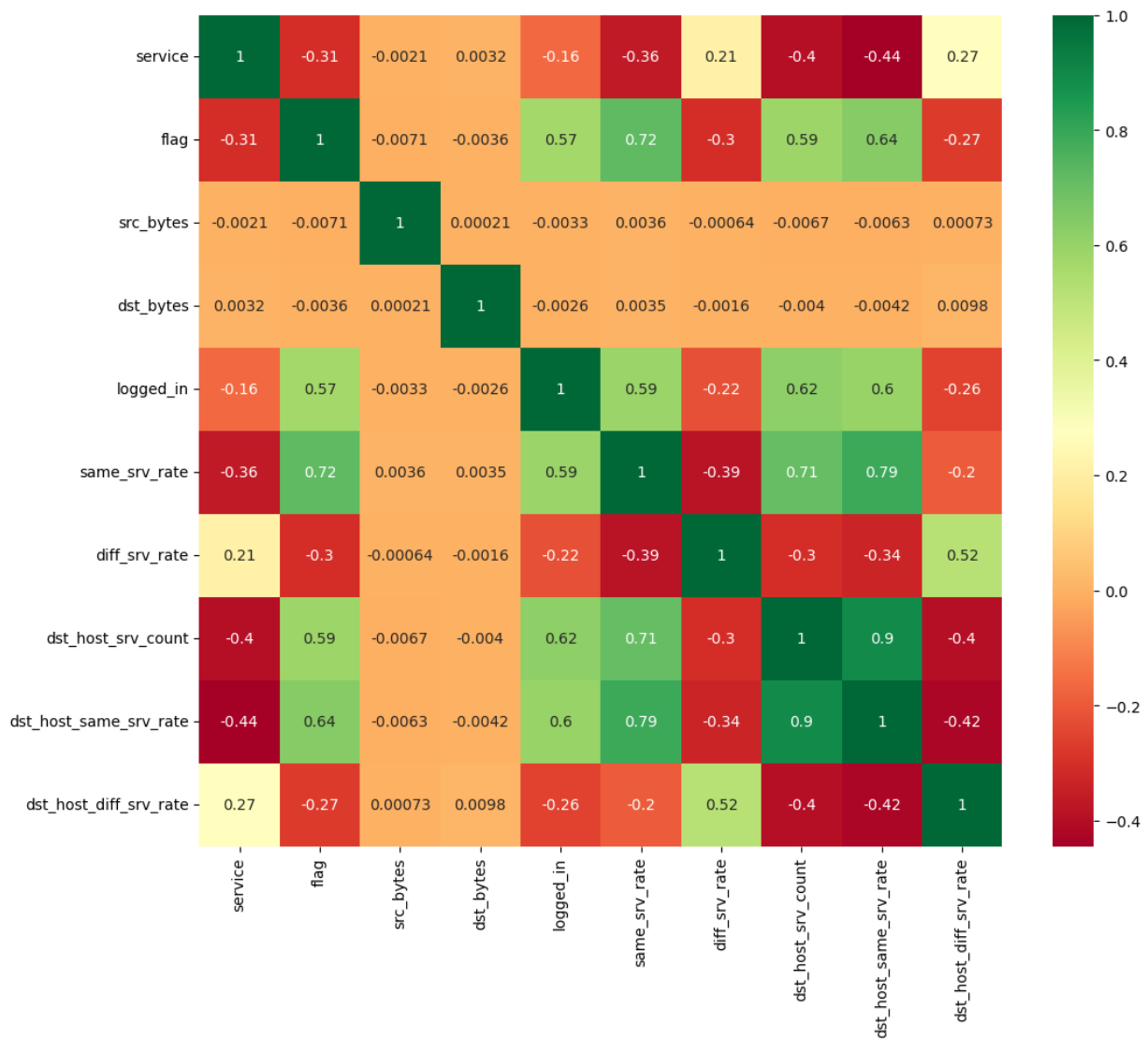


FIGURE 2.10 – Correlation de chaque attribut avec les autres attributs

2.5 Conclusion

Après avoir normalisé nos données et réalisé les visualisations nécessaires, la prochaine étape consiste à travailler sur le modèle et à choisir le bon algorithme pour notre problématique.

CHAPTER 3

MODEL D'APPRENTISSAGE

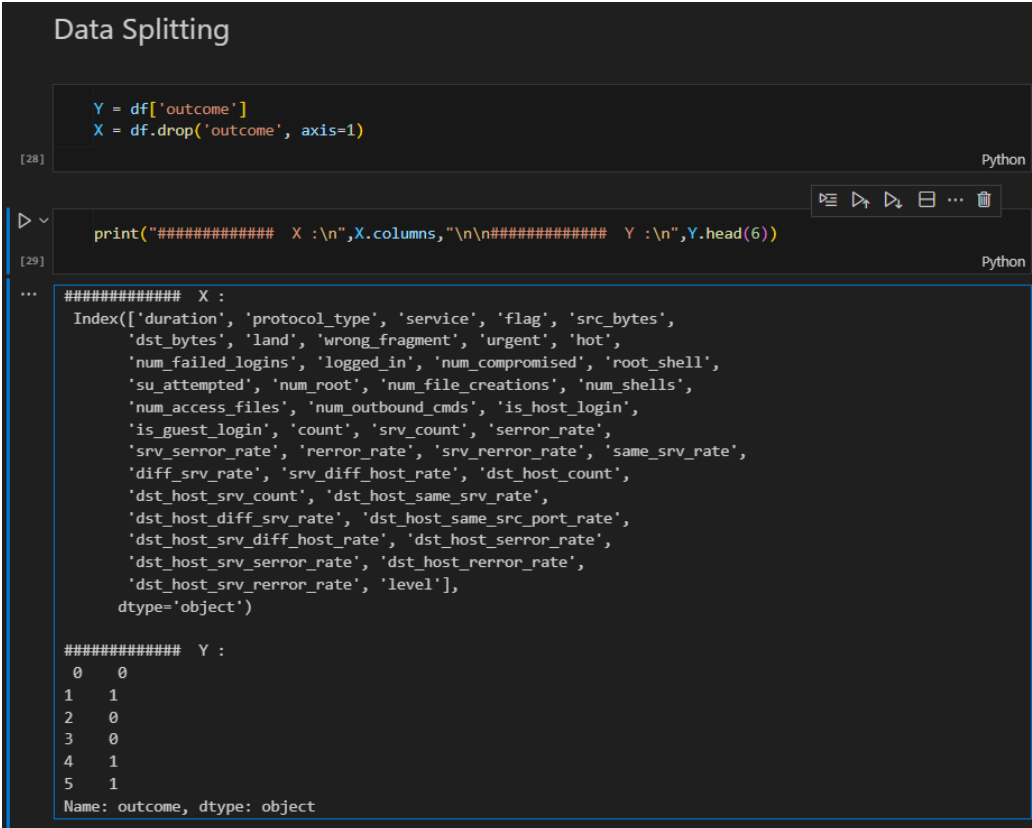
3.1 Introduction

Ce chapitre présente tout ce qui concerne notre modèle, notamment la séparation de notre ensemble de données en ensembles d'apprentissage et de test, ainsi que l'analyse de chaque algorithme utilisée.

3.2 Fractionnement des données

Pour cette problématique, nous nous concentrons sur un problème de classification, une tâche fondamentale en apprentissage automatique. La classification consiste à attribuer des étiquettes ou des catégories à des données en fonction de leurs caractéristiques, permettant ainsi de distinguer différentes classes ou catégories. En fonction des données fournies à notre modèle, celui-ci sera en mesure de prédire si une observation est considérée comme normale ou si elle constitue une attaque.

Donc attribut que nous devons prédire est désigné sous le nom de **Outcome**. Pour séparer les attributs et les stocker dans différentes variables, nous réalisons une opération de partitionnement de notre ensemble de données.



```
Data Splitting

Y = df['outcome']
X = df.drop('outcome', axis=1)

[28] Python

print("##### X :\n",X.columns,"\n\n##### Y :\n",Y.head(6))

[29] Python

...
##### X :
Index(['duration', 'protocol_type', 'service', 'flag', 'src_bytes',
       'dst_bytes', 'land', 'wrong_fragment', 'urgent', 'hot',
       'num_failed_logins', 'logged_in', 'num_compromised', 'root_shell',
       'su_attempted', 'num_root', 'num_file_creations', 'num_shells',
       'num_access_files', 'num_outbound_cmds', 'is_host_login',
       'is_guest_login', 'count', 'srv_count', 'error_rate',
       'srv_error_rate', 'rerror_rate', 'srv_rerror_rate', 'same_srv_rate',
       'diff_srv_rate', 'srv_diff_host_rate', 'dst_host_count',
       'dst_host_srv_count', 'dst_host_same_srv_rate',
       'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate',
       'dst_host_srv_diff_host_rate', 'dst_host_error_rate',
       'dst_host_srv_error_rate', 'dst_host_rerror_rate',
       'dst_host_srv_rerror_rate', 'level'],
      dtype='object')

##### Y :
0    0
1    1
2    0
3    0
4    1
5    1
Name: outcome, dtype: object
```

FIGURE 3.1 – Séparer les attributs

Après cette étape, nous avons utilisé deux méthodes de division de données afin de les comparer et de déterminer si la méthode utilisée influence l'efficacité du modèle. Ces méthodes sont la [Train Test Split](#) et [Cross Validation](#) .

3.2.1 Train Test Split

Une fois la fonction `train test split`[2] définie, elle renvoie un ensemble train et un ensemble test. Ce splitting des données permet d'évaluer un modèle de Machine Learning sous deux angles différents. Le modèle est entraîné sur l'ensemble train renvoyé par la fonction. Puis ses capacités prédictives sont évaluées sur l'ensemble test renvoyé par la fonction. Plusieurs métriques peuvent être utilisées pour cette évaluation.

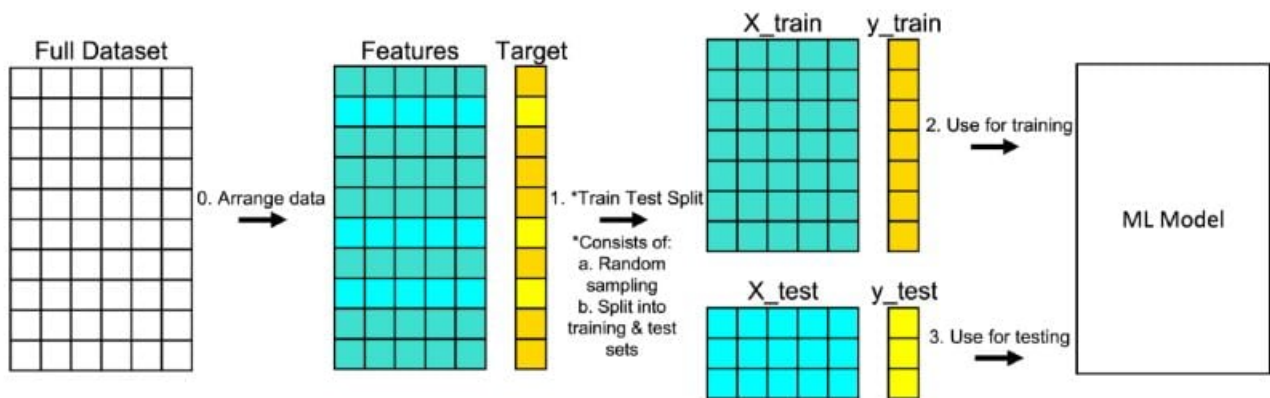


FIGURE 3.2 – Schéma de train test split

Dans le cas d'une régression linéaire, le coefficient de détermination, la RMSE et la MAE sont privilégiés. Dans le cas d'une classification, l'accuracy, la précision, le recall et le F1-score sont privilégiés. Ces scores sur l'ensemble test permettent donc de déterminer si le modèle est performant et à quel point il doit être amélioré avant de pouvoir prédire sur un nouveau dataset. Les ensembles train et test renvoyés par la fonction `train test split` jouent aussi un rôle essentiel dans la détection d'overfitting ou d'underfitting. Pour notre code :

```
x_train, x_test, y_train, y_test = train_test_split(X,Y, test_size=0.2, random_state=42)
print("Training data : \n\nx : ",x_train.shape , " y : ", y_train.shape,"\n\nTesting data : \n\nx : ",x_test.shape , " y : ", y_test.shape)
```

[31]

```
... Training data :

x : (118812, 42) y : (118812,)

Testing data :

x : (29704, 42) y : (29704,)
```

+ Code + Markdown

FIGURE 3.3 – implémentation de train test split en notre code

3.2.2 Cross Validation ou k-Folds

Facile à comprendre et très populaire auprès des professionnels, la technique K-Folds[1], propose généralement un modèle moins biaisé que les autres méthodes qui existe. La principale raison est qu'elle permet de s'assurer que toutes les observations du jeu de données original puissent apparaître dans l'ensemble d'entraînement, mais aussi dans l'ensemble utilisé pour le test. Dans la situation où les données d'input sont limitées, c'est l'une des meilleures solutions à considérer.

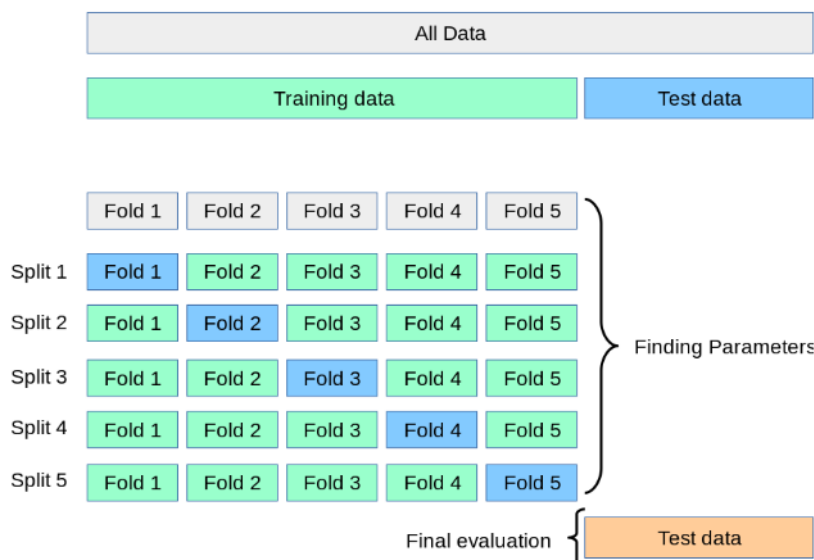
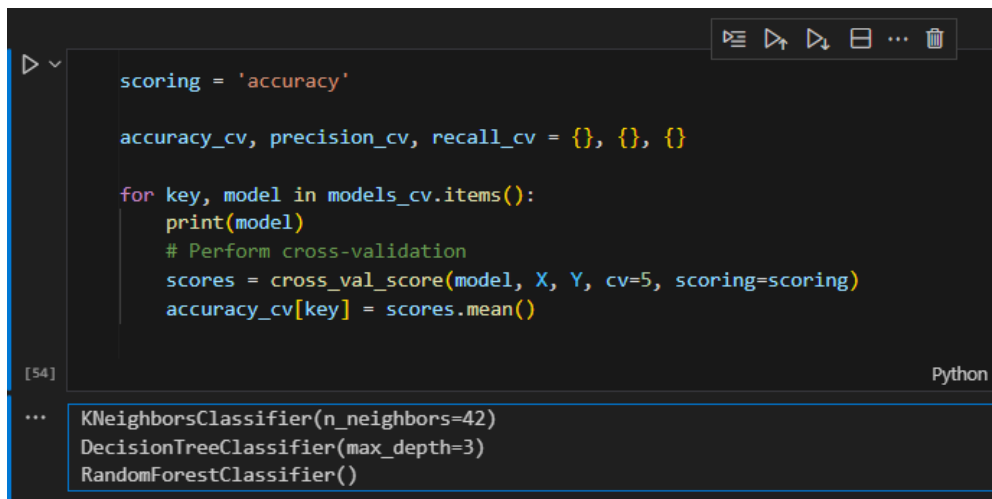


FIGURE 3.4 – Schéma de Cross Validation

Pour commencer, on effectue d'abord une séparation de l'ensemble de données de façon aléatoire en K folds. Un paramétrage unique dénommé « K » est inséré dans la procédure, se référant au nombre de groupe dans lequel l'échantillon sera réparti. K doit avoir une valeur ni trop basse ni trop haute. La plupart du temps, on choisira une valeur qui se situe entre 5 et 10 selon la taille du dataset. Dans le cas où $K=10$, cela implique qu'il faut diviser le dataset en 10 parties.

Plus la valeur K est élevée, moins le modèle risque d'être biaisé. Sans oublier qu'une variance trop large peut amener à un surajustement. Avec une valeur plus basse, l'approche rejoint simplement la méthode Train-Test Split. En notre code :



```
scoring = 'accuracy'

accuracy_cv, precision_cv, recall_cv = {}, {}, {}

for key, model in models_cv.items():
    print(model)
    # Perform cross-validation
    scores = cross_val_score(model, X, Y, cv=5, scoring=scoring)
    accuracy_cv[key] = scores.mean()
```

[54]

Python

```
... KNeighborsClassifier(n_neighbors=42)
     DecisionTreeClassifier(max_depth=3)
     RandomForestClassifier()
```

FIGURE 3.5 – implémentation de Cross Validation en notre code

3.3 Modeling

3.3.1 Algorithmes d'apprentissage automatique

En notre code, nous travaillons avec quatre algorithmes de classification : la régression logistique, la méthode des k plus proches voisins (KNN), les arbres de décision et la forêt d'arbres aléatoires (Random Forest).

3.3.1.1 Phase Entraînement

[la régression logistique](#) :[6] Ce type de modèle statistique (également appelé modèle logit) est souvent utilisé pour la classification et l'analyse prédictive. La régression logistique estime la probabilité qu'un événement se produise, comme avoir voté ou non, sur la base d'un ensemble de données donné de variables indépendantes. Puisque le résultat est une probabilité, la variable dépendante est limitée entre 0 et 1. Dans la régression logistique, une transformation logit est appliquée aux probabilités, c'est-à-dire la probabilité de succès divisée par la probabilité d'échec. Ceci est également communément appelé log des cotes, ou logarithme naturel des cotes, et cette fonction logistique est représentée par les formules suivantes :

$$h_{\theta}(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}},$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

FIGURE 3.6 – formule mathématique de régression logistique

k plus proches voisins (KNN) : également connu sous le nom de KNN ou k-NN, est un classificateur d'apprentissage supervisé non paramétrique, qui utilise la proximité pour effectuer des classifications ou des prédictions sur le regroupement d'un point de données individuel. Bien qu'il puisse être utilisé pour des problèmes de régression ou de classification, il est généralement utilisé comme algorithme de classification, partant de l'hypothèse que des points similaires peuvent être trouvés les uns à proximité des autres.

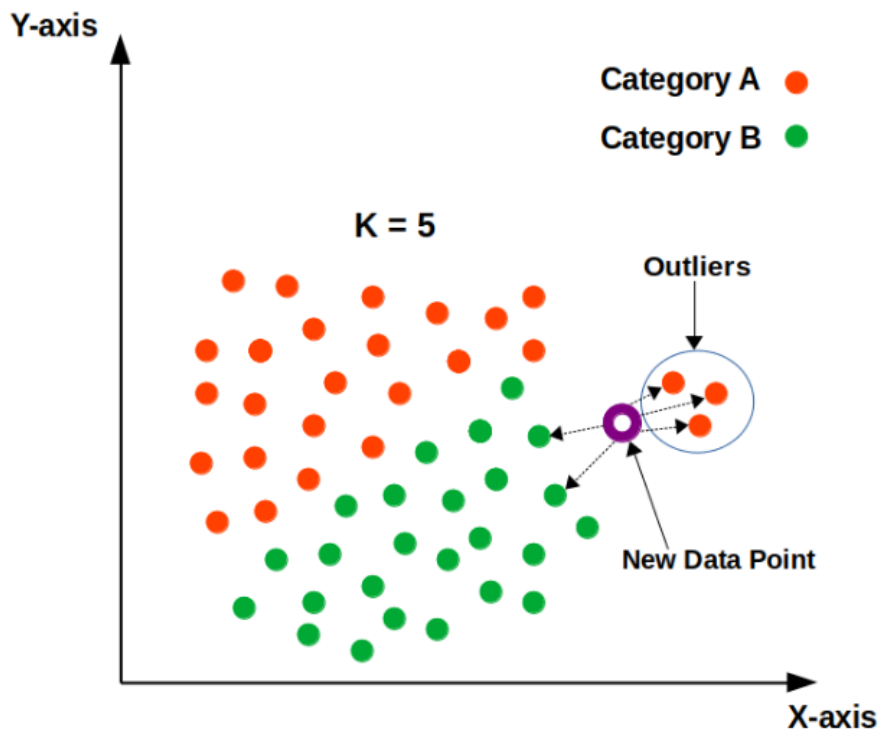


FIGURE 3.7 – Schéma de k plus proches voisins

L'arbre de décision : est l'outil de classification et de prédiction le plus puissant et le plus populaire. Un arbre de décision est une structure arborescente de type organigramme, dans laquelle chaque nœud interne désigne un test sur un attribut, chaque branche représente un résultat du test et chaque nœud feuille (nœud terminal) contient une étiquette de classe.

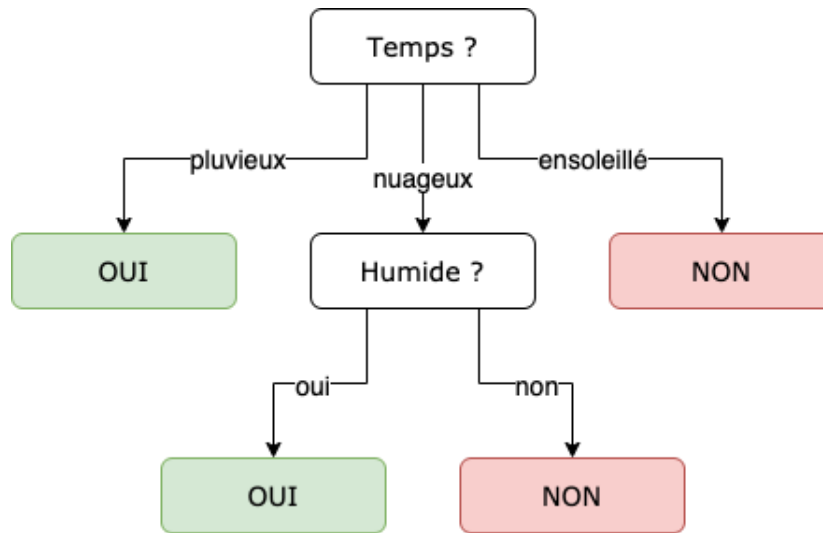


FIGURE 3.8 – Schéma d’arbre de décision

la forêt d’arbres aléatoires ([Random Forest](#)) : est l’outil de classification et de prédiction le plus puissant et le plus populaire. Un arbre de décision est une structure arborescente de type organigramme, dans laquelle chaque nœud interne désigne un test sur un attribut, chaque branche représente un résultat du test et chaque nœud feuille (nœud terminal) contient une étiquette de classe.

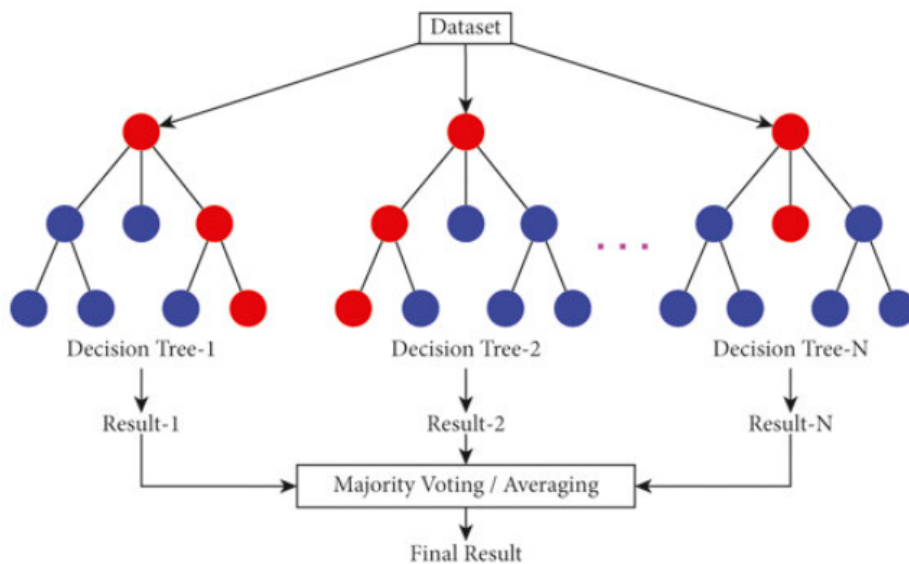


FIGURE 3.9 – Schéma de Random Forest

Pour implémenter ces algorithmes dans notre code, nous commençons par initialiser chacun d’entre eux.

```
models_to_analys = {
    'Logistic_Regression': LogisticRegression(max_iter=1000),
    'KNN': KNeighborsClassifier(n_neighbors = 42),
    'Decision_Trees': DecisionTreeClassifier(max_depth=3),
    'Random_Forest': RandomForestClassifier()
}
```

FIGURE 3.10 – initialiser chaque algorithme

Après avoir entraîné ces algorithmes avec les données d'entraînement et calculé leur exactitude (accuracy), leur précision (precision) et leur rappel (recall) pour chaque algorithme. ca par Train Test Split dataset :

```
# Initialize metrics dictionaries
accuracy_tts, precision_tts, recall_tts = {}, {}, {}

for key, model in models_to_analys.items():
    model.fit(x_train, y_train)
    predictions = model.predict(x_test)

    # Calculate metrics
    accuracy_tts[key] = accuracy_score(predictions, y_test)
    precision_tts[key] = precision_score(predictions, y_test, average='weighted')
    recall_tts[key] = recall_score(predictions, y_test, average='weighted')
```

FIGURE 3.11 – Entraînement ces algorithme par train test datasplit

et pour Cross Validation :

```
# to use cross validation for decision tree and random forest and knn
last_two_models = list(models_to_analys.items())[1:]
models_cv = dict(last_two_models)

scoring = 'accuracy'
accuracy_cv, precision_cv, recall_cv = {}, {}, {}

for key, model in models_cv.items():
    print(model)
    # Perform cross-validation
    scores = cross_val_score(model, X, Y, cv=5, scoring=scoring)
    accuracy_cv[key] = scores.mean()
```

```
... KNeighborsClassifier(n_neighbors=42)
DecisionTreeClassifier(max_depth=3)
RandomForestClassifier()
```

FIGURE 3.12 – Entraînement ces algorithme par Cross Validation méthode

3.3.1.2 Phase Evaluation

Enfin, dans cette partie, nous pouvons évaluer chaque algorithme que nous avons utilisé et répondre à la question de savoir si les méthodes de fractionnement des données peuvent affecter l'efficacité de notre modèle. Pour cela, nous calculons l'exactitude (accuracy), le rappel (recall) et la précision (precision) pour chaque algorithmes.

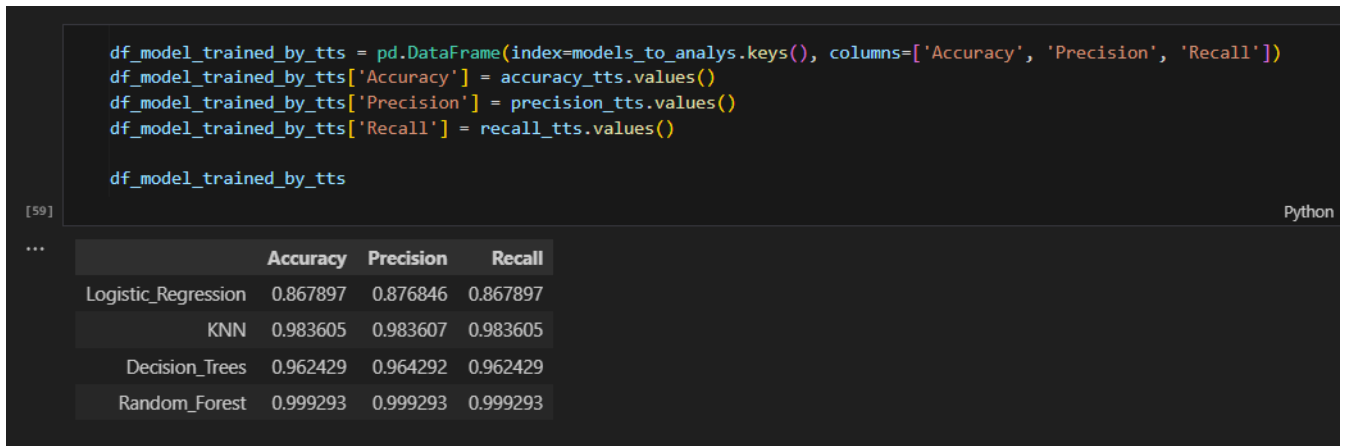


FIGURE 3.13 – Evaluation de ces algorithmes

pour rendre le résultat présentable, nous le montrons sous forme de graphique



FIGURE 3.14 – Métriques d'évaluation pour les modèles formés à l'aide de Train-Test Split

Nous utilisons également la matrice de confusion pour évaluer les performances de chaque algorithme qui est une représentation tabulaire qui permet de visualiser les performances d'un modèle de classification. Elle compare les valeurs réelles des données avec les prédictions du

modèle, en classifiant les prédictions en quatre catégories : vrai positif (TP), vrai négatif (TN), faux positif (FP) et faux négatif (FN) .

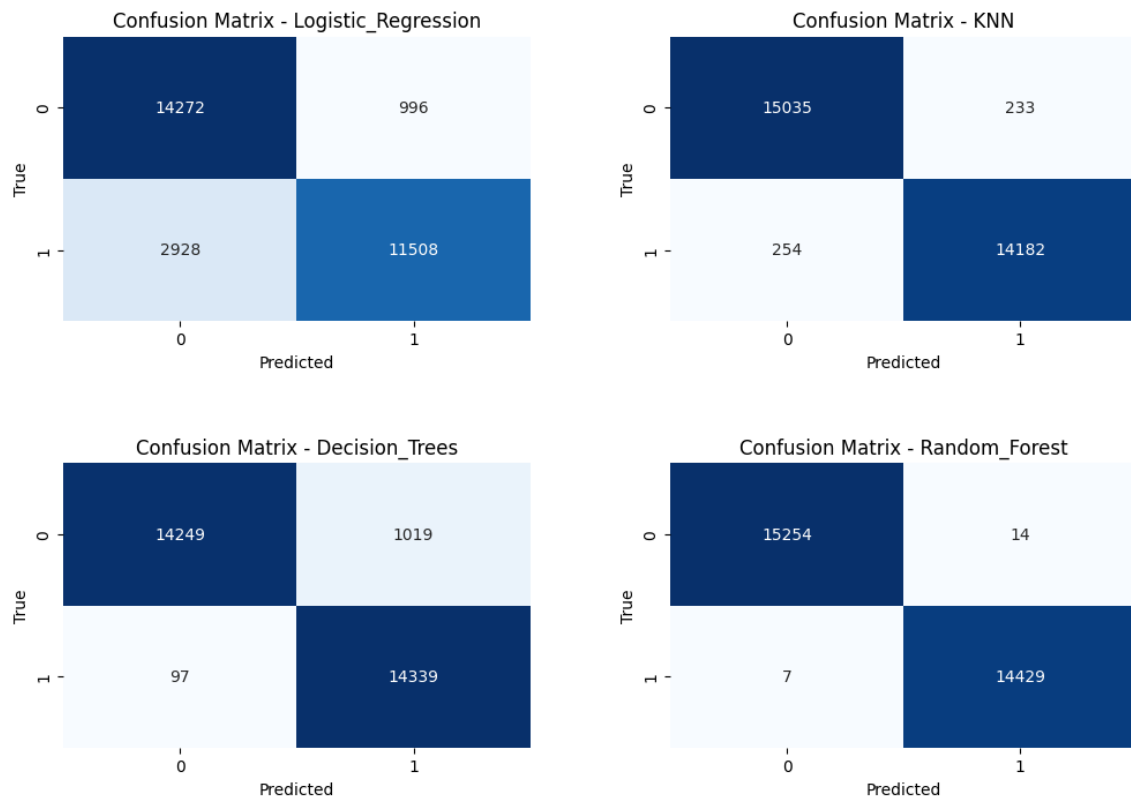


FIGURE 3.15 – Matrice de confusion

Pour répondre à la question de savoir si les méthodes de fractionnement des données (Data Splitting) peuvent affecter l'efficacité de notre modèle, nous présentons les résultats sur ce graphe.

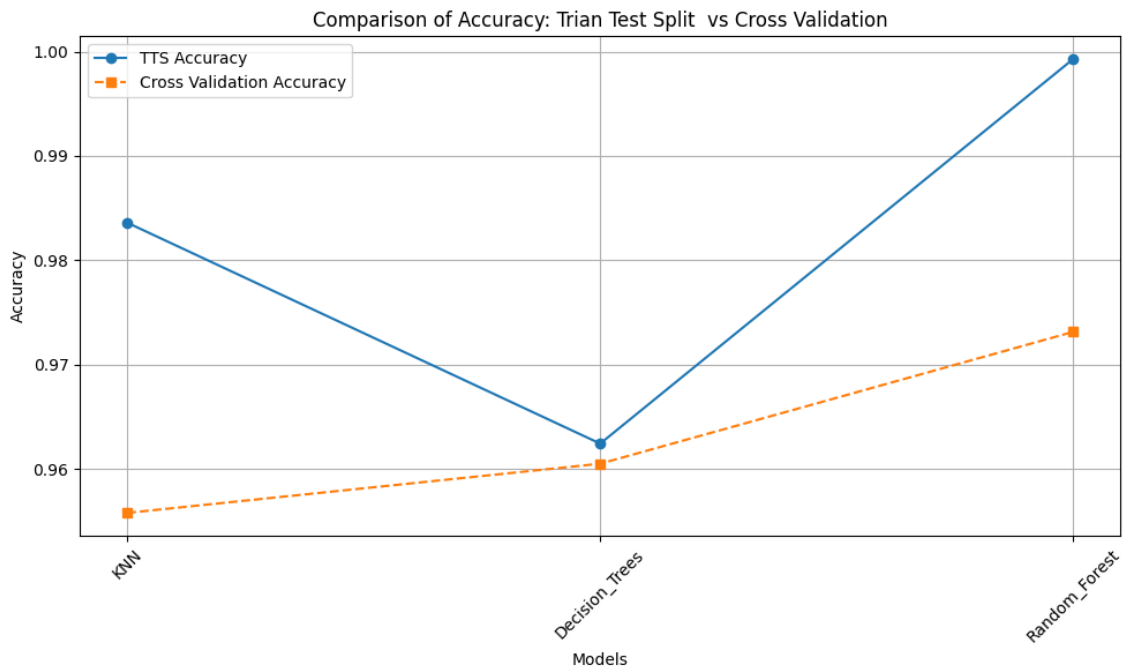


FIGURE 3.16 – Comparaison de Accuracy : Trian Test Split vs Cross Validation

Dans le graphe, nous mettons en évidence la méthode choisie pour le fractionnement des données qui joue un rôle crucial dans l'évaluation de notre modèle.

3.3.2 Deep Learning :

Les réseaux de neurones, également appelés réseaux de neurones artificiels (ANN) ou réseaux de neurones simulés (SNN), sont un sous-ensemble de l'apprentissage automatique et sont au cœur des algorithmes d'apprentissage profond. Leur nom et leur structure sont inspirés du cerveau humain, imitant la façon dont les neurones biologiques se communiquent les uns aux autres. Les réseaux de neurones artificiels (ANN) sont constitués d'une couche de nœuds contenant une couche d'entrée, une ou plusieurs couches cachées et une couche de sortie. Chaque nœud, ou neurone artificiel, se connecte à un autre et possède un poids et un seuil associés. Si la sortie d'un nœud individuel est supérieure à la valeur seuil spécifiée, ce nœud est activé et envoie des données à la couche suivante du réseau. Sinon, aucune donnée n'est transmise à la couche suivante du réseau.

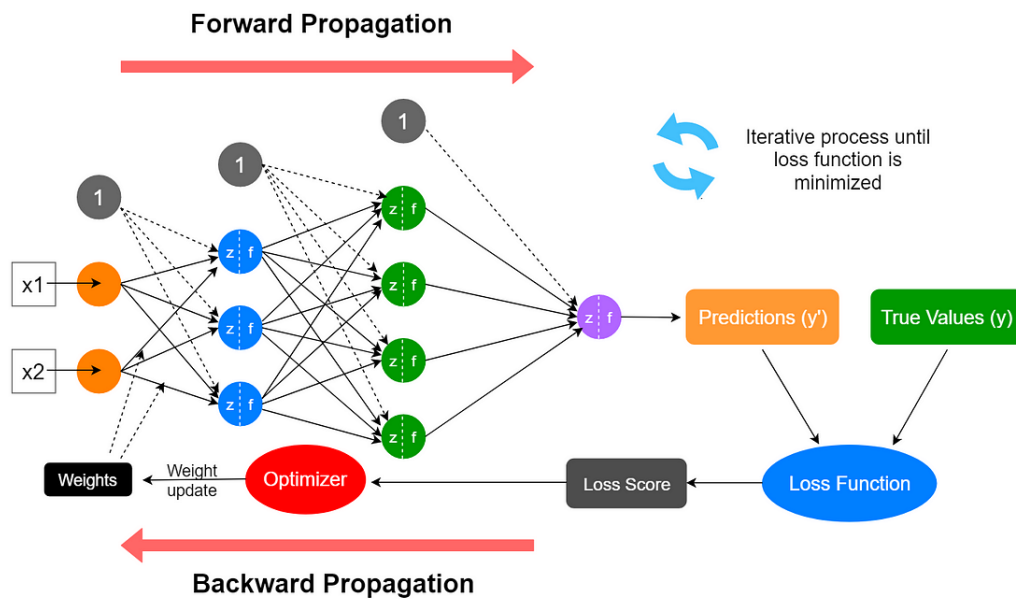


FIGURE 3.17 – Exemple d'un réseau de neurone

Ainsi, pour l'architecture de notre modèle, nous utilisons un réseau de neurones séquentiel avec plusieurs couches de neurones denses. Chaque couche est suivie d'une couche d'abandon, qui désactive certains neurones pendant l'entraînement pour éviter le surapprentissage. Le modèle commence par une petite couche dense, puis passe par des couches qui s'élargissent progressivement avant de se rétrécir à nouveau vers la sortie. Quant à la régularisation, nous appliquons des pénalités aux poids et activations de chaque couche pour éviter que le modèle ne soit surajusté aux données d'entraînement.

```

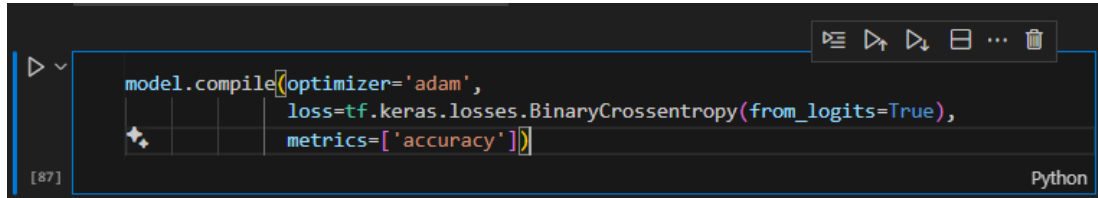
model = Sequential([
    Dense(units=64, activation='relu', input_shape=(x_train.shape[1:])),
    Dropout(0.4),
    Dense(units=128, activation='relu'),
    Dropout(0.4),
    Dense(units=512, activation='relu'),
    Dropout(0.4),
    Dense(units=128, activation='relu'),
    Dropout(0.4),
    Dense(units=1, activation='sigmoid')
])

# Add regularizations : is a technique used in machine learning and deep learning to prevent overfitting,
# which occurs when a model learns to memorize the training data rather than generalize from it
for layer in model.layers:
    if isinstance(layer, Dense):
        layer.kernel_regularizer = regularizers.L1L2(l1=1e-5, l2=1e-4)
        layer.bias_regularizer = regularizers.L2(1e-4)
        layer.activity_regularizer = regularizers.L2(1e-5)

```

FIGURE 3.18 – Définir notre modèle DL

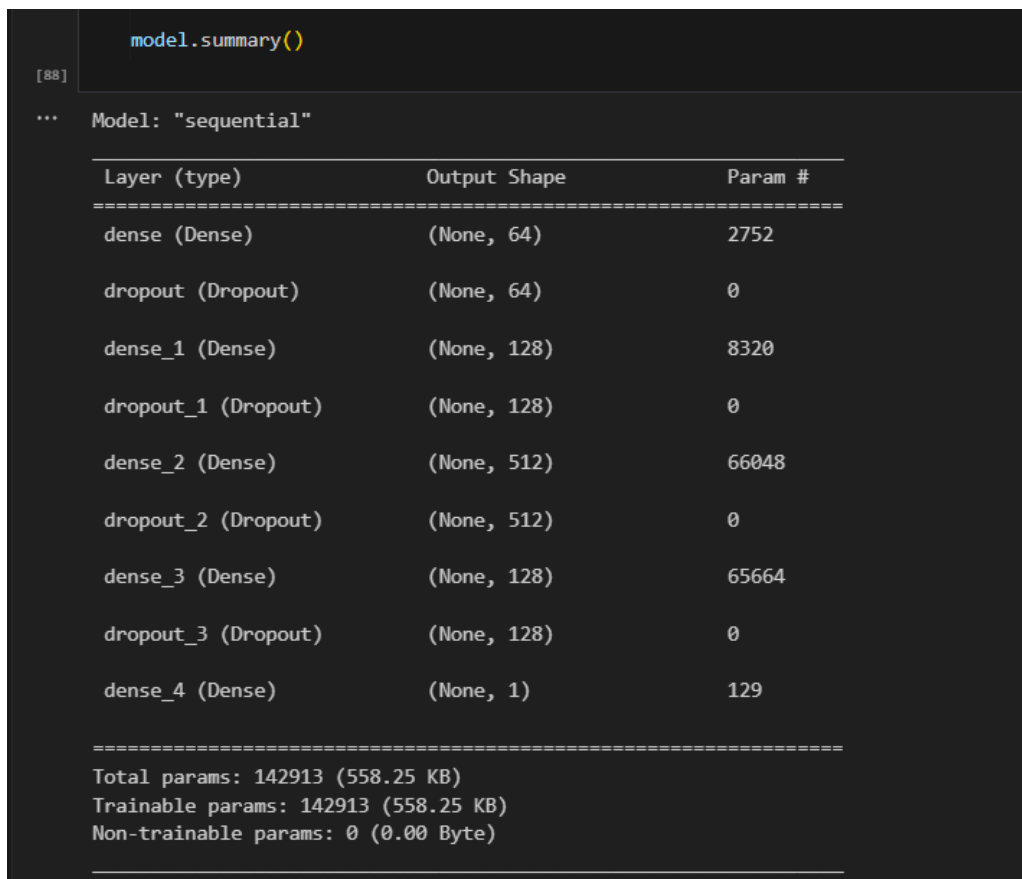
Pour compiler le modèle pour la formation. Il utilise l'optimiseur Adam, un algorithme d'optimisation largement utilisé, ainsi qu'une fonction de perte d'entropie croisée binaire, adaptée aux tâches de classification binaire. De plus, il définit la précision comme mesure permettant d'évaluer les performances du modèle pendant l'entraînement.



```
[87] model.compile(optimizer='adam',
                loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
                metrics=['accuracy'])
```

FIGURE 3.19 – Compile le modèle

pour architecture de modèle



```
[88] model.summary()
```

```
... Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	2752
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 128)	8320
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 512)	66048
dropout_2 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 128)	65664
dropout_3 (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 1)	129

```

Total params: 142913 (558.25 KB)
Trainable params: 142913 (558.25 KB)
Non-trainable params: 0 (0.00 Byte)

```

FIGURE 3.20 – Architecture de modèle

Etape suivante est entraîner le modèle compilé en utilisant les données d'entraînement (x-train, y-train). Il valide également les performances du modèle en utilisant les données de validation (x-test, y-test). Pendant l'entraînement, il s'exécute pendant 20 epochs, chaque epoch itérant sur l'ensemble des données. Le paramètre batch-size définit le nombre d'échantillons traités avant la mise à jour des paramètres internes du modèle. De plus, verbose=1 définit le niveau de

verboosité de l'entraînement pour afficher des barres de progression pendant l'entraînement. La variable `model-train-hist` stocke l'historique de l'entraînement, y compris les métriques telles que la perte et la précision, pour une analyse et une visualisation ultérieures

```

1 model_train_hist = model.fit(x_train, y_train,
2                             validation_data=(x_test, y_test),
3                             epochs=20,
4                             batch_size=32,
5                             verbose=1)

```

FIGURE 3.21 – Entraînement notre DL modèle

Et comme outputs in terminal on a :

```

... Epoch 1/20
3709/3713 [=====>.] - ETA: 0s - loss: 103678536.0000 - accuracy: 0.8876
C:\Users\yassi\AppData\Roaming\Python\Python310\site-packages\keras\src\backend.py:5818: UserWarning: ``binary_crossentropy`` received `fr
output, from_logits = _get_logits(
3713/3713 [=====] - 53s 14ms/step - loss: 103570328.0000 - accuracy: 0.8876 - val_loss: 6514277.5000 - val_accu
Epoch 2/20
3713/3713 [=====] - 49s 13ms/step - loss: 29574164.0000 - accuracy: 0.9235 - val_loss: 1127566.6250 - val_accu
Epoch 3/20
3713/3713 [=====] - 50s 13ms/step - loss: 2048160.0000 - accuracy: 0.9315 - val_loss: 1.0356 - val_accuracy: 0.9
Epoch 4/20
3713/3713 [=====] - 40s 11ms/step - loss: 6.2940 - accuracy: 0.9384 - val_loss: 0.1835 - val_accuracy: 0.9550
Epoch 5/20
3713/3713 [=====] - 43s 11ms/step - loss: 0.2096 - accuracy: 0.9404 - val_loss: 0.1814 - val_accuracy: 0.9446
Epoch 6/20
3713/3713 [=====] - 49s 13ms/step - loss: 0.2114 - accuracy: 0.9373 - val_loss: 0.1493 - val_accuracy: 0.9577
Epoch 7/20
3713/3713 [=====] - 55s 15ms/step - loss: 0.2114 - accuracy: 0.9394 - val_loss: 0.1635 - val_accuracy: 0.9498
Epoch 8/20
3713/3713 [=====] - 55s 15ms/step - loss: 0.2282 - accuracy: 0.9377 - val_loss: 0.1560 - val_accuracy: 0.9518
Epoch 9/20
3713/3713 [=====] - 51s 14ms/step - loss: 0.2146 - accuracy: 0.9371 - val_loss: 0.1634 - val_accuracy: 0.9476
Epoch 10/20
3713/3713 [=====] - 47s 13ms/step - loss: 0.2106 - accuracy: 0.9380 - val_loss: 0.1656 - val_accuracy: 0.9492
Epoch 11/20
3713/3713 [=====] - 48s 13ms/step - loss: 0.2022 - accuracy: 0.9389 - val_loss: 0.1698 - val_accuracy: 0.9511
Epoch 12/20
3713/3713 [=====] - 47s 13ms/step - loss: 0.2088 - accuracy: 0.9408 - val_loss: 0.1553 - val_accuracy: 0.9557
Epoch 13/20
3713/3713 [=====] - 36s 10ms/step - loss: 0.2047 - accuracy: 0.9378 - val_loss: 0.1757 - val_accuracy: 0.9428
...
Epoch 19/20
3713/3713 [=====] - 43s 12ms/step - loss: 0.2400 - accuracy: 0.9423 - val_loss: 0.1546 - val_accuracy: 0.9538
Epoch 20/20
3713/3713 [=====] - 39s 11ms/step - loss: 0.1958 - accuracy: 0.9421 - val_loss: 0.1513 - val_accuracy: 0.9547
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```

FIGURE 3.22 – Epochs notre DL modèle

Pour évaluation premierment nos tracons les courbes de perte (loss) et de validation (valloss) en fonction du nombre d'epochs. Les données sont extraites de l'historique de l'entraînement `model-train-history`. Les valeurs sur l'axe des x représentent les epochs, tandis que les valeurs sur l'axe des y représentent la perte (loss) calculée à l'aide de la fonction SCCE Loss (Sparse Categorical Crossentropy). Enfin, le code ajoute des étiquettes d'axe, une légende pour différencier les courbes, et active la grille pour une meilleure lisibilité du graphique

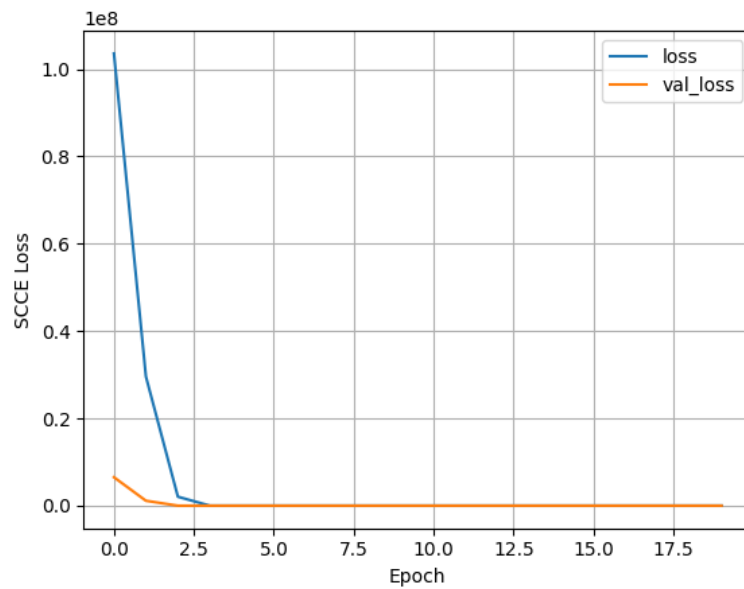


FIGURE 3.23 – Epochs en fonction de SCCE Loss

Et nous traçons une autre courbe décrivant le changement de l'accuracy à chaque epoch.

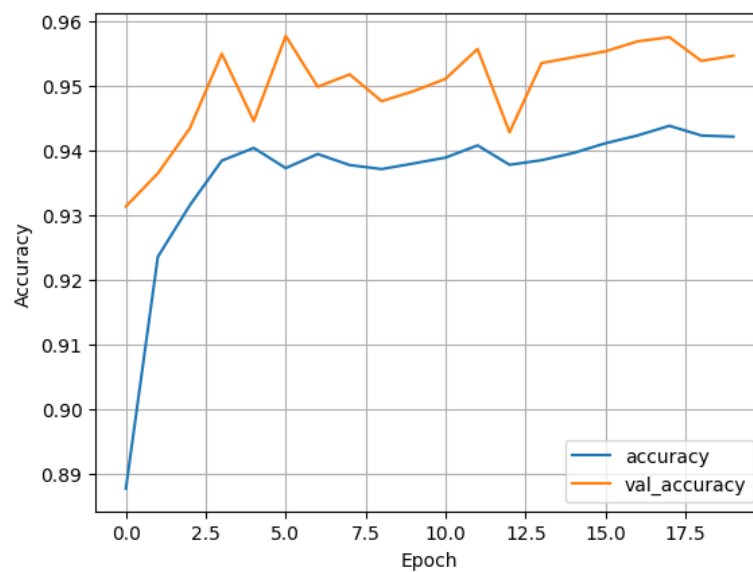


FIGURE 3.24 – Epochs en fonction d'accuracy

À la fin, nous sauvegardons le modèle dans un fichier ".h5".

```
1 model.save('./models/my_model_IDS.h5')
```

FIGURE 3.25 – code de sauvegarde le modèle

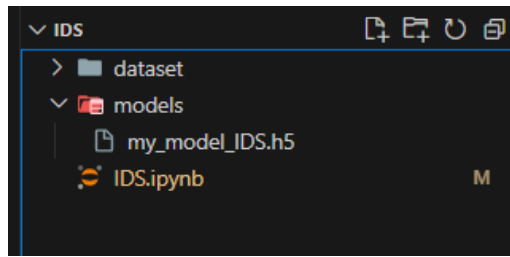


FIGURE 3.26 – path de sauvegarde

3.4 Conclusion

Dans ce chapitre, j'ai décrit tous les aspects possible de notre modèle ainsi que son évaluation.

CONCLUSION

Ce rapport offre un aperçu du travail réalisé tout au long de mon mini-projet. L'objectif principal était de concevoir un modèle pour un système de détection d'intrusion (IDS) capable de prédire les attaques. J'ai utilisé des technologies modernes pour fournir un code de haute qualité afin de mener à bien ces travaux.

Les différentes étapes qui m'ont été confiées pour accomplir ces tâches comprenaient l'analyse, la conception et le développement. Ce mini-projet a été une excellente occasion pour moi de mettre en pratique les connaissances théoriques acquises lors de ma formation en master en mathématiques, cryptographie et cybersécurité (M2C) à l'École Normale Supérieure de Casablanca.

De plus, mon autoformation m'a permis d'acquérir des compétences dans le travail avec de nouvelles technologies très demandées sur le marché de l'emploi. qualité afin de mener à bien ces travaux. Pour source code de projet : Intrusion Detection System Analyze en Github.

BIBLIOGRAPHIE

- [1] Scikit-learn documentation : Cross validation function. https://scikit-learn.org/stable/modules/cross_validation.html.
- [2] Scikit-learn documentation : train test split function. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.
- [3] Scikit-learn documentation : Visualization. <https://scikit-learn.org/stable/visualizations.html>.
- [4] Stefan Axelsson. Intrusion detection systems : A survey and taxonomy. *IEEE Communications Surveys & Tutorials*, 2(4) :1–23, 2000.
- [5] Joseph F Hair, William C Black, Barry J Babin, and Ronald E Anderson. *Multivariate Data Analysis*. Cengage Learning, 8 edition, 2019.
- [6] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied Logistic Regression*. John Wiley & Sons, 2013.