Stage d'Été

Stage ingénieur

Projet : Détection de Fraude Bancaire

Réalisé par : Yasmine Ben Larbi

Encadrante : Madame Ines Haddad Saidani

Société d'accueil : Capgemini

Année académique : 2024/2025

Remerciements

Je tiens à exprimer ma profonde gratitude à l'École Supérieure Privée d'Ingénierie et de Technologies (ESPRIT) pour m'avoir offert l'opportunité de réaliser ce projet dans le cadre de ma formation. Cette expérience a été une étape déterminante dans mon parcours académique et professionnel, me permettant d'approfondir mes connaissances dans le domaine de la science des données et de la détection de fraude bancaire. Grâce à l'enseignement de qualité que j'ai reçu à ESPRIT, j'ai pu aborder ce projet avec confiance et rigueur.

Je souhaite également remercier Capgemini, mon entreprise d'accueil, pour m'avoir offert un cadre stimulant et enrichissant où j'ai eu la chance d'apprendre et de me perfectionner dans un environnement professionnel international. Cette expérience au sein de Capgemini m'a permis de développer des compétences pratiques, de travailler avec des experts et de participer à des projets innovants. L'approche collaborative et les échanges fructueux au sein de l'équipe m'ont permis d'approfondir ma compréhension des technologies et des méthodologies utilisées dans le secteur de l'intelligence artificielle et du machine learning.

Un remerciement particulier à Madame Ines Haddad Saidani, mon encadrante, pour son soutien constant, ses conseils avisés et son dévouement tout au long de ce stage. Grâce à sa patience, son expertise et sa bienveillance, j'ai pu surmonter les défis techniques et méthodologiques rencontrés au cours du projet. Ses orientations précieuses ont été déterminantes pour la réussite de ce travail et pour l'acquisition de nouvelles compétences.

Je tiens enfin à remercier tous ceux qui, de près ou de loin, ont contribué au bon déroulement de ce stage et à la réalisation de ce projet.

Introduction Générale

La détection de fraude bancaire est un enjeu majeur dans le secteur financier, où des milliards de transactions sont effectuées chaque jour à travers le monde. Avec l'essor du commerce électronique et des transactions numériques, les institutions financières sont de plus en plus confrontées à des tentatives de fraude sophistiquées. Ces fraudes peuvent avoir des conséquences financières et juridiques importantes, non seulement pour les institutions, mais également pour leurs clients. Dans ce contexte, l'utilisation de méthodes avancées de détection de fraude, basées sur l'analyse des données et le machine learning, est devenue une priorité.

Le projet que nous avons mené s'inscrit dans cet effort. L'objectif est de développer des modèles d'apprentissage automatique capables d'identifier avec précision les transactions frauduleuses dans un dataset bancaire. Ce projet est particulièrement complexe en raison de la nature déséquilibrée des données, où les transactions frauduleuses représentent une infime minorité par rapport aux transactions normales. Pour surmonter cet obstacle, nous avons mis en œuvre plusieurs techniques, telles que le suréchantillonnage avec SMOTE, l'entraînement de différents modèles de machine learning, et l'évaluation approfondie des performances de ces modèles.

De plus, pour rendre les résultats accessibles et utilisables par des non-experts, nous avons intégré ces modèles dans une application web interactive à l'aide de Streamlit. Cette application permet de visualiser les données, tester différents modèles, et observer en temps réel la prédiction des fraudes. Tenez en compte que le travail présenté ici n'est pas entièrement approfondi en raison de contraintes de temps et de ressources, mais l'application illustre le potentiel d'un outil qui pourrait être utilisé dans un contexte réel pour surveiller les transactions bancaires et prévenir les fraudes. Je vous prie de faire preuve de patience en considérant qu'il s'agit d'une étape préliminaire dans le développement d'une solution complète.

Ainsi, ce projet illustre l'application des techniques de machine learning à une problématique réelle, tout en mettant en lumière les défis liés à la manipulation de données déséquilibrées, aux contraintes de ressources matérielles et à l'intégration d'un modèle dans une interface utilisateur interactive.

I. Présentation de l'entreprise d'accueil

1. Présentation de Capgemini

Capgemini, fondée en 1967, est une entreprise internationale leader dans le secteur du conseil, des services informatiques et de la transformation numérique. Avec plus de 270 000 employés dans le monde entier, Capgemini se positionne comme un partenaire stratégique pour accompagner les entreprises dans leur transformation digitale.

Capgemini Tunisie, quant à elle, fait partie intégrante de cette multinationale en offrant des services adaptés au marché tunisien et international. Avec une équipe expérimentée et des compétences diversifiées, Capgemini Tunisie accompagne ses clients dans le développement de solutions innovantes et dans la mise en œuvre de projets technologiques complexes.

Depuis l'acquisition de Telnet, Capgemini Tunisie s'est consolidée comme un acteur clé du marché local et international. Elle propose des services diversifiés, adaptés aux besoins des entreprises, notamment dans le domaine des nouvelles technologies et des projets technologiques complexes. Cette fusion a permis de conjuguer les forces des deux entités pour offrir des solutions de pointe.

Dans le cadre de cette acquisition, les équipes de Telnet ont été intégrées dans les activités de Capgemini, ce qui explique que les deux noms puissent apparaître conjointement sur des documents officiels comme l'attestation de stage. Cela reflète la transition de Telnet vers Capgemini, tout en préservant les expertises locales et les collaborations déjà établies par Telnet.

Cette synergie illustre parfaitement la stratégie de Capgemini, qui consiste à s'appuyer sur les talents locaux tout en exploitant son réseau mondial pour apporter des solutions innovantes à ses clients.

2. Services :

Capgemini propose une large gamme de services, incluant :

- Conseil en transformation digitale
- Développement de solutions logicielles
- Hébergement et infogérance
- Cloud et cybersécurité

II . Description du projet

1. Description du projet :

La détection de fraude est un enjeu critique pour les institutions financières. L'objectif principal de ce projet est de développer des modèles d'apprentissage automatique capables d'identifier avec précision les transactions frauduleuses dans un ensemble de données déséquilibré. Ce rapport expose en détail chaque étape de ce projet, avec des justifications méthodologiques pour chaque choix effectué. Une attention particulière est portée aux explications des résultats et des visualisations.

2. Objectifs spécifiques du projet :

1. Identifier le modèle le plus performant pour la détection des fraudes.
2. Gérer le déséquilibre important des classes.
3. Optimiser les modèles pour améliorer les performances sans augmenter le temps de calcul.
4. Expliquer pourquoi certaines décisions méthodologiques ont été prises, notamment la non-utilisation de la cross-validation et l'utilisation partielle des données pour accélérer l'exécution.

3. Définition du machine learning

Le machine learning, ou apprentissage automatique, est une branche de l'intelligence artificielle qui permet aux machines d'apprendre à partir de données sans être explicitement programmées. Dans le cadre de ce projet, plusieurs algorithmes de machine learning ont été utilisés pour analyser les transactions financières et détecter les anomalies qui pourraient signaler une fraude.

4. Technologies et logiciels utilisés

Les technologies suivantes ont été utilisées tout au long du projet :

- Anaconda : Environnement de développement pour la gestion des packages Python.
- Python : Langage de programmation principal utilisé pour l'implémentation des modèles de machine learning.
- Jupyter Notebook : Outil interactif pour l'analyse des données.
- Excel : Utilisé pour la manipulation de certaines données.

5. Méthodologie adoptée

Pour ce projet, j'ai adopté la méthodologie CRISP-DM (Cross-Industry Standard Process for Data Mining), qui est un cadre structuré en plusieurs étapes pour le traitement des projets de data science:machine learning. Les étapes suivies dans le cadre de ce projet sont :

a. Compréhension des données
b. Préparation des données
c. Modélisation
d. Évaluation
e. Déploiement

III . Exécution du projet (Jupyter)

1. Exploration et Préparation des Données

1.1. Chargement des Données

```python
# Importation des librairies nécessaires
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from scipy.stats import zscore

# Chargement du dataset
data = pd.read_csv('creditcard.csv')

# Aperçu des premières lignes du dataset
print("Aperçu des premières lignes du dataset :")
print(data.head())

# Informations générales sur les données
print("\nInformations générales sur les données :")
print(data.info())

# Statistiques descriptives
print("\nStatistiques descriptives :")
print(data.describe())
```

```
Aperçu des premières lignes du dataset :
   Time        V1        V2        V3        V4        V5        V6        V7  \
0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

         V8        V9  ...       V21       V22       V23       V24       V25  \
0  0.098698  0.363787  ... -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  ...  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.141267 -0.206010

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0

[5 rows x 31 columns]
```

```
Informations générales sur les données :
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
```

```
None

Statistiques descriptives :
                Time            V1            V2            V3            V4  \
count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean    94813.859575  1.168375e-15  3.416908e-16 -1.379537e-15  2.074095e-15
std     47488.145955  1.958696e+00  1.651309e+00  1.516255e+00  1.415869e+00
min         0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00
25%     54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-01
50%     84692.000000  1.810880e-02  6.548556e-02  1.798463e-01 -1.984653e-02
75%    139320.500000  1.315642e+00  8.037239e-01  1.027196e+00  7.433413e-01
max    172792.000000  2.454930e+00  2.205773e+01  9.382558e+00  1.687534e+01

                 V5            V6            V7            V8            V9  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   9.604066e-16  1.487313e-15 -5.556467e-16  1.213481e-16 -2.406331e-15
std    1.380247e+00  1.332271e+00  1.237094e+00  1.194353e+00  1.098632e+00
min   -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -1.343407e+01
25%   -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -6.430976e-01
50%   -5.433583e-02 -2.741871e-01  4.010308e-02  2.235804e-02 -5.142873e-02
75%    6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01  5.971390e-01
max    3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01  1.559499e+01

       ...           V21           V22           V23           V24  \
count  ...  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   ...  1.654067e-16 -3.568593e-16  2.578648e-16  4.473266e-15
std    ...  7.345240e-01  7.257016e-01  6.244603e-01  6.056471e-01
min    ... -3.483038e+01 -1.093314e+01 -4.480774e+01 -2.836627e+00
25%    ... -2.283949e-01 -5.423504e-01 -1.618463e-01 -3.545861e-01
50%    ... -2.945017e-02  6.781943e-03 -1.119293e-02  4.097606e-02
75%    ...  1.863772e-01  5.285536e-01  1.476421e-01  4.395266e-01
max    ...  2.720284e+01  1.050309e+01  2.252841e+01  4.584549e+00

                V25           V26           V27           V28  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   5.340915e-16  1.683437e-15 -3.660091e-16 -1.227390e-16
std    5.212781e-01  4.822270e-01  4.036325e-01  3.300833e-01
min   -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01
25%   -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02
50%    1.659350e-02 -5.213911e-02  1.342146e-03  1.124383e-02
75%    3.507156e-01  2.409522e-01  9.104512e-02  7.827995e-02
max    7.519589e+00  3.517346e+00  3.161220e+01  3.384781e+01

              Amount  \
count  284807.000000
mean       88.349619
std       250.120109
min         0.000000
25%         5.600000
50%        22.000000
75%        77.165000
max     25691.160000

              Class
count  284807.000000
mean        0.001727
std         0.041527
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000

[8 rows x 31 columns]
```

Le dataset utilisé, creditcard.csv, contient 284 807 transactions bancaires réalisées sur deux jours. Les colonnes comprennent :

- **Time** : temps écoulé (en secondes) depuis la première transaction,
- **Amount** : montant de la transaction,
- **Class** : étiquette cible (0 = non-fraude, 1 = fraude),
- **V1 à V28** : Les variables V1 à V28 sont des combinaisons anonymisées de caractéristiques originales, générées via une Analyse en Composantes Principales (PCA), permettant de préserver la confidentialité tout en conservant les informations statistiques essentielles pour l'analyse.

Aucune valeur manquante n'a été détectée, ce qui facilite le prétraitement.

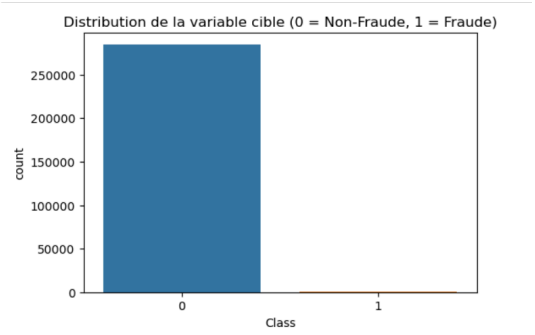
1.2. Analyse Exploratoire des Données (EDA)

L'analyse exploratoire a permis de mieux comprendre les caractéristiques des données

1.2.1 Distribution de la variable cible (Class) :

```python
# Distribution de la variable cible
plt.figure(figsize=(6,4))
sns.countplot(x='Class', data=data)
plt.title("Distribution de la variable cible (0 = Non-Fraude, 1 = Fraude)")
plt.show()

# Calcul de la proportion des classes
fraud_percentage = data['Class'].value_counts(normalize=True) * 100
print("\nProportion des classes :")
print(fraud_percentage)
```




```
Proportion des classes :
Class
0    99.827251
1     0.172749
Name: proportion, dtype: float64
```

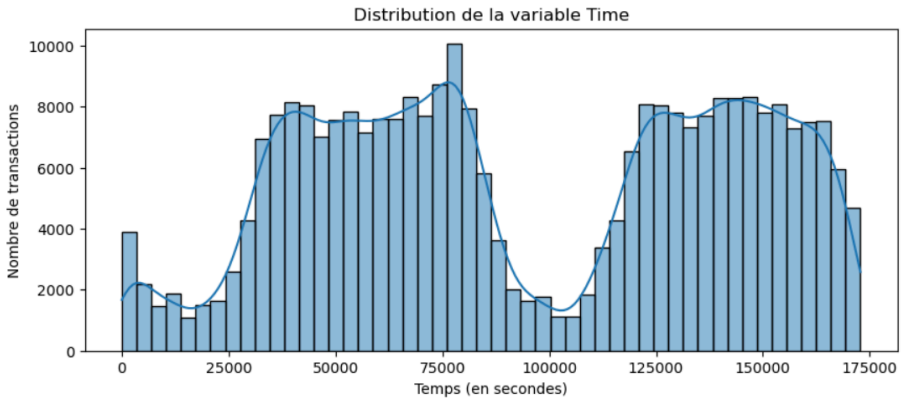
Les transactions frauduleuses représentent seulement **0.17%** des données, ce qui met en évidence un déséquilibre significatif.

Proportion des classes :

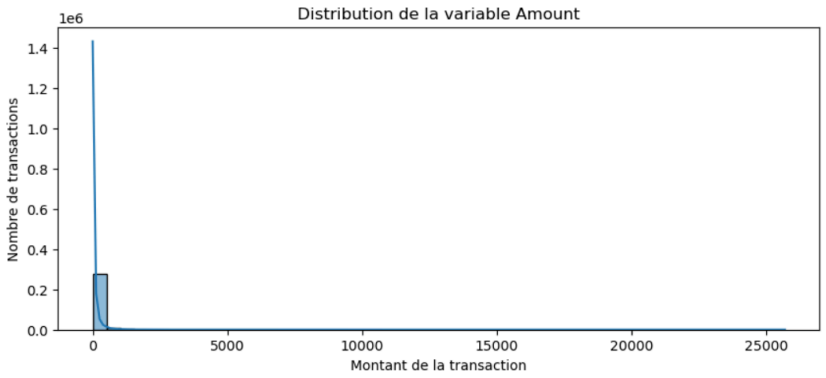
- Non-Fraude : **99.83%**
- Fraude : **0.17%**

1.2.2 Distribution de Time et Amount :

Time : Les transactions frauduleuses sont réparties aléatoirement dans le temps.



Amount : La majorité des montants sont faibles, mais quelques transactions ont des valeurs atypiques.

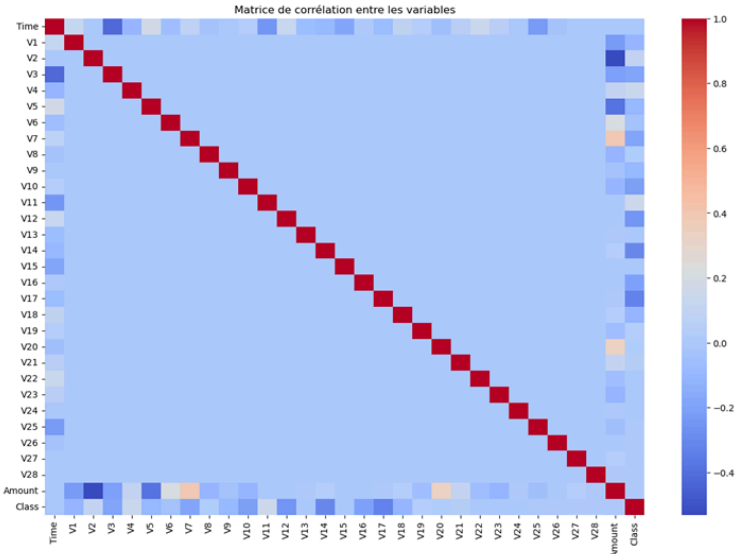


1.2.3 Analyse des Variables PCA (V1 à V28) :

Les distributions des variables PCA sont proches de la normale, mais certaines montrent des queues épaisses ou des outliers, nécessitant une normalisation.

1.2.4 Corrélations :

Les variables issues de la PCA ont des corrélations faibles, comme attendu. Les variables issues de la PCA ont des corrélations faibles car la PCA transforme les données pour que chaque nouvelle variable (V1 à V28) soit indépendante des autres. Cela signifie qu'elles ne partagent pas d'information redondante, chaque variable capturant une partie unique de la variance des données.



1.3. Prétraitement des Données

1.3.1 Normalisation et Standardisation :

Normalisation de Amount avec un Z-score pour réduire l'effet des valeurs extrêmes. La normalisation de **Amount** avec un Z-score est utilisée pour réduire l'impact des valeurs extrêmes (outliers) en ramenant la variable à une échelle standard. Cela permet aux modèles de mieux distinguer les transactions normales des fraudes, indépendamment de l'ampleur des montants.

Vérification des variables PCA pour garantir une échelle uniforme.

Entrée [7]:
```
# Détection et gestion des valeurs aberrantes avec Z-score (exemple avec Amount)
threshold = 3  # seuil de Z-score pour détecter les valeurs aberrantes
data['Amount_Zscore'] = zscore(data['Amount'])
outliers = data[np.abs(data['Amount_Zscore']) > threshold]

# Affichage des transactions avec des montants atypiques
print("\nTransactions avec des valeurs atypiques dans Amount :")
print(outliers[['Amount', 'Class']])
```

```
Transactions avec des valeurs atypiques dans Amount :
          Amount  Class
51       1402.95      0
89       1142.02      0
140       919.60      0
150       937.69      0
164      3828.04      0
...          ...    ...
284249  10199.44      0
284290    897.00      0
284383   1114.00      0
284497   1484.66      0
284528    900.00      0

[4076 rows x 2 columns]
```

1.3.2 Gestion du Déséquilibre des Classes :

Application de **SMOTE (Synthetic Minority Oversampling Technique)** pour équilibrer la classe minoritaire. L'application de SMOTE (Synthetic Minority Oversampling Technique) est essentielle pour équilibrer les données lorsque la classe minoritaire (les fraudes) est sous-représentée. En générant artificiellement de nouvelles instances similaires à celles de la classe minoritaire, SMOTE améliore la capacité des modèles à apprendre les caractéristiques distinctives de cette classe et évite qu'ils soient biaisés en faveur de la classe majoritaire.

```
from imblearn.over_sampling import SMOTE

# Séparation des caractéristiques et de la cible
X = data.drop('Class', axis=1)
y = data['Class']

# Application de SMOTE pour sur-échantillonner la classe minoritaire
smote = SMOTE(random_state=42)
X_smote, y_smote = smote.fit_resample(X, y)

# Création d'un DataFrame équilibré
data_balanced = pd.concat([pd.DataFrame(X_smote, columns=X.columns), pd.DataFrame(y_smote, columns=['Class'])], axis=1)

# Vérification de la distribution après SMOTE
print("Distribution des classes après SMOTE :")
print(data_balanced['Class'].value_counts())
```

```
Distribution des classes après SMOTE :
Class
0    284315
1    284315
Name: count, dtype: int64
```

2. Modélisation et Résultats

2.1. Division des Données

- Données équilibrées après SMOTE :
    - **Non-Fraude** : 284 315 transactions
    - **Fraude** : 284 315 transactions

- Séparation en données d'entraînement (70%) et de test (30%). La séparation en données d'entraînement (70%) et de test (30%) est une étape essentielle pour évaluer les performances d'un modèle de machine learning.

- **Données d'entraînement (70%)** : Elles servent à construire et ajuster le modèle en apprenant les relations entre les caractéristiques et la cible.
- **Données de test (30%)** : Elles permettent d'évaluer le modèle sur des données qu'il n'a jamais vues, afin de mesurer sa capacité à généraliser et à faire des prédictions sur de nouvelles transactions.

Cette séparation garantit une évaluation objective des performances du modèle, évitant un biais lié à un surapprentissage sur l'ensemble des données.

```
from sklearn.model_selection import train_test_split

# Séparation des caractéristiques (X) et de la cible (y)
X = data_balanced.drop('Class', axis=1)
y = data_balanced['Class']

# Division des données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

2.2. Modèles Utilisés

Dans cette étude, nous avons testé quatre modèles de machine learning pour la détection de fraude :

-Régression Logistique (Logistic Regression) :
Ce modèle est simple, rapide à entraîner, et efficace pour les données linéairement séparables. Il est souvent utilisé comme point de départ pour évaluer les performances de classification.

-Forêt d'Arbres Décisionnels (Random Forest) :
Ce modèle d'ensemble repose sur la combinaison de plusieurs arbres de décision pour améliorer la robustesse et réduire le risque de surapprentissage. Il est performant pour gérer des données complexes et non linéaires.

-Gradient Boosting (HistGradientBoostingClassifier) :
Ce modèle optimise les prédictions en ajustant successivement les erreurs des modèles précédents. Il est puissant pour capturer des relations complexes et a montré une performance exceptionnelle dans les tâches de classification.

- Réseau de Neurones (MLP - Multi-Layer Perceptron) :
Ce modèle est basé sur une architecture de couches entièrement connectées, capable de modéliser des relations non linéaires et des interactions complexes entre les variables.

La validation croisée a été utilisée avant l'optimisation pour évaluer de manière fiable les performances initiales des modèles, limiter le surapprentissage, et comparer les approches tout en tenant compte des contraintes de ressources matérielles.

2.2.1 Régression Logistique

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import cross_val_score, train_test_split


# Initialisation du modèle de Régression Logistique
logreg = LogisticRegression(random_state=42, max_iter=200, solver='liblinear')

# Validation croisée
logreg_scores = cross_val_score(logreg, X_train, y_train, cv=5, scoring='roc_auc')
print("ROC-AUC moyen pour la Régression Logistique :", logreg_scores.mean())

# Entraînement final et prédictions
logreg.fit(X_train, y_train)
y_pred_logreg = logreg.predict(X_test)

# Évaluation
print("\nRégression Logistique - Rapport de classification :")
print(classification_report(y_test, y_pred_logreg))
print("Matrice de confusion :\n", confusion_matrix(y_test, y_pred_logreg))
```

```
ROC-AUC moyen pour la Régression Logistique : 0.9896910159340317

Régression Logistique - Rapport de classification :
              precision    recall  f1-score   support

           0       0.96      0.98      0.97     85149
           1       0.98      0.95      0.97     85440

    accuracy                           0.97    170589
   macro avg       0.97      0.97      0.97    170589
weighted avg       0.97      0.97      0.97    170589

Matrice de confusion :
 [[83644  1505]
 [ 3880 81560]]
```

La régression logistique a montré de bonnes performances avec un ROC-AUC moyen de 0,989, indiquant une excellente capacité à différencier les transactions frauduleuses (classe 1) des transactions normales (classe 0). Les métriques de précision (0,96 pour la classe 0 et 0,98 pour la classe 1), de rappel (0,98 pour la classe 0 et 0,95 pour la classe 1) et de F1-score (toutes deux à 0,97) montrent un bon équilibre entre détection des fraudes et limitation des faux positifs. La matrice de confusion indique que, bien que le modèle ait détecté correctement la majorité des cas, il reste quelques erreurs (3880 faux négatifs et 1505 faux positifs).

2.2.2 Random Forest

```python
from sklearn.ensemble import RandomForestClassifier

# Modèle Random Forest
rf = RandomForestClassifier(random_state=42, n_estimators=50, max_depth=7)

# Validation croisée
rf_scores = cross_val_score(rf, X_train, y_train, cv=5, scoring='roc_auc')
print("ROC-AUC moyen pour Random Forest :", rf_scores.mean())

# Entraînement final et prédictions
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)

# Évaluation
print("\nRandom Forest - Rapport de classification :")
print(classification_report(y_test, y_pred_rf))
print("Matrice de confusion :\n", confusion_matrix(y_test, y_pred_rf))
```

```
ROC-AUC moyen pour Random Forest : 0.9980670199200897

Random Forest - Rapport de classification :
              precision    recall  f1-score   support

           0       0.96      1.00      0.98     85149
           1       1.00      0.96      0.98     85440

    accuracy                           0.98    170589
   macro avg       0.98      0.98      0.98    170589
weighted avg       0.98      0.98      0.98    170589

Matrice de confusion :
 [[84845   304]
 [ 3365 82075]]
```

Le modèle Random Forest affiche des performances élevées avec un ROC-AUC moyen de 0,998, démontrant une capacité presque parfaite à distinguer les fraudes des transactions normales. Avec une précision de 0,96 pour la classe 0 et de 1,00 pour la classe 1, le modèle limite efficacement les faux positifs. Le rappel, de 1,00 pour la classe 0 et de 0,96 pour la classe 1, indique une excellente détection des transactions normales mais légèrement moins pour les fraudes. La matrice de confusion révèle 3365 faux négatifs et 304 faux positifs, ce qui est une amélioration par rapport à la régression logistique. Ces résultats en font un modèle robuste et fiable pour la détection de fraudes.

2.2.3 Gradient Boosting

```python
from sklearn.ensemble import HistGradientBoostingClassifier
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler

# Étape 1 : Standardisation des données pour améliorer l'efficacité
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Étape 2 : Réduire la taille des données pour la validation croisée
X_train_cv, _, y_train_cv, _ = train_test_split(X_train_scaled, y_train, test_size=0.8, random_state=42)

# Étape 3 : Initialisation du Gradient Boosting
gb = HistGradientBoostingClassifier(random_state=42)

# Étape 4 : Cross-validation avec cv=3 pour limiter le temps de calcul
gb_scores = cross_val_score(gb, X_train_cv, y_train_cv, cv=3, scoring='roc_auc', n_jobs=-1)
print("ROC-AUC moyen pour Gradient Boosting (Reduced Data, cv=3):", gb_scores.mean())

# Étape 5 : Entraînement final sur les données complètes et prédictions
gb.fit(X_train_scaled, y_train)
y_pred_gb = gb.predict(X_test_scaled)

# Étape 6 : Évaluation des performances
print("\nGradient Boosting - Rapport de classification :")
print(classification_report(y_test, y_pred_gb))
print("Matrice de confusion :\n", confusion_matrix(y_test, y_pred_gb))
```

```
ROC-AUC moyen pour Gradient Boosting (Reduced Data, cv=3): 0.9999376455319086

Gradient Boosting - Rapport de classification :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85149
           1       1.00      1.00      1.00     85440

    accuracy                           1.00    170589
   macro avg       1.00      1.00      1.00    170589
weighted avg       1.00      1.00      1.00    170589

Matrice de confusion :
 [[85083    66]
 [   12 85428]]
```

Le modèle Gradient Boosting atteint une performance quasi parfaite avec un ROC-AUC moyen de 0,9999. Les scores de précision, rappel et F1-score sont tous de 1,00 pour les deux classes, indiquant une capacité optimale à détecter aussi bien les fraudes que les transactions normales. La matrice de confusion montre seulement 12 faux négatifs et 66 faux positifs, ce qui est un excellent résultat. Ces performances exceptionnelles font de Gradient Boosting le modèle le plus performant de cette étude, particulièrement adapté à la détection de fraudes avec un équilibre parfait entre précision et rappel.

2.2.4 Réseau de Neurones (MLP)

```python
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix

# Standardisation des données
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Réduire la taille des données pour la validation croisée
X_train_cv, _, y_train_cv, _ = train_test_split(X_train_scaled, y_train, test_size=0.8, random_state=42)

# Réseau de Neurones avec plus d'itérations
mlp = MLPClassifier(hidden_layer_sizes=(32,), max_iter=200, solver='adam', random_state=42)

# Validation croisée
mlp_scores = cross_val_score(mlp, X_train_cv, y_train_cv, cv=3, scoring='roc_auc', n_jobs=-1)
print("ROC-AUC moyen pour Réseau de Neurones (Reduced Data, cv=3):", mlp_scores.mean())

# Entraînement final sur les données complètes
mlp.fit(X_train_scaled, y_train)

# Prédictions sur les données de test
y_pred_mlp = mlp.predict(X_test_scaled)

# Évaluation des performances
print("\nRéseau de Neurones - Rapport de classification :")
print(classification_report(y_test, y_pred_mlp))
print("Matrice de confusion :\n", confusion_matrix(y_test, y_pred_mlp))
```

```
ROC-AUC moyen pour Réseau de Neurones (Reduced Data, cv=3): 0.9998906380156951

Réseau de Neurones - Rapport de classification :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85149
           1       1.00      1.00      1.00     85440

    accuracy                           1.00    170589
   macro avg       1.00      1.00      1.00    170589
weighted avg       1.00      1.00      1.00    170589

Matrice de confusion :
 [[85068    81]
 [    3 85437]]
```

Le modèle Réseau de Neurones affiche également des performances exceptionnelles avec un ROC-AUC moyen de 0,9999. Tous les scores de précision, rappel et F1-score atteignent 1,00, indiquant une excellente capacité à classifier correctement les fraudes et les transactions normales. La matrice de confusion révèle 81 faux positifs et seulement 3 faux négatifs, montrant une très grande précision dans la détection des fraudes. Ces résultats positionnent ce modèle parmi les meilleurs pour cette étude, avec une fiabilité presque parfaite dans un contexte de déséquilibre des classes.

3. Optimisation des Modèles

Des techniques d'optimisation ont été appliquées :

- **Grid Search** pour affiner les hyperparamètres.

3.1. **Régression Logistique**

```python
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix

# Standardisation des données
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Définition des paramètres pour Grid Search
param_grid_logreg = {
    'C': [0.01, 0.1, 1, 10, 100],
    'solver': ['lbfgs', 'liblinear']
}

# Initialisation du modèle
logreg = LogisticRegression(random_state=42, max_iter=500)

# Grid Search avec validation croisée (cv=5)
grid_search_logreg = GridSearchCV(logreg, param_grid_logreg, cv=5, scoring='roc_auc', n_jobs=-1)
grid_search_logreg.fit(X_train_scaled, y_train)

# Meilleurs paramètres et performance
print("Meilleurs paramètres pour la Régression Logistique :", grid_search_logreg.best_params_)
print("Meilleur score ROC-AUC :", grid_search_logreg.best_score_)

# Entraînement final avec les meilleurs paramètres
best_logreg = grid_search_logreg.best_estimator_
best_logreg.fit(X_train_scaled, y_train)
y_pred_logreg = best_logreg.predict(X_test_scaled)
```

```
Meilleurs paramètres pour la Régression Logistique : {'C': 100, 'solver': 'liblinear'}
Meilleur score ROC-AUC : 0.9974867814990003
```

Après optimisation, le modèle de régression logistique utilise les meilleurs hyperparamètres trouvés (C=100, solver='liblinear'), ce qui améliore son score ROC-AUC à 0.9975, reflétant une excellente capacité à distinguer les transactions frauduleuses des normales.

3.2 Random Forest

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import classification_report, confusion_matrix

# Sous-échantillonnage des données pour GridSearchCV
X_train_sub, _, y_train_sub, _ = train_test_split(X_train, y_train, test_size=0.8, random_state=42)

# Définition des paramètres pour Grid Search
param_grid_rf = {
    'n_estimators': [10, 50],  # Nombre d'arbres
    'max_depth': [5, 10],  # Profondeur maximale des arbres
    'min_samples_split': [2, 5],  # Taille minimale des divisions
    'min_samples_leaf': [1, 2]  # Taille minimale des feuilles
}

# Initialisation du modèle
rf = RandomForestClassifier(random_state=42, n_jobs=-1)

# Grid Search avec paramètres optimisés
grid_search_rf = GridSearchCV(rf, param_grid_rf, cv=3, scoring='roc_auc', n_jobs=-1)
grid_search_rf.fit(X_train_sub, y_train_sub)

# Meilleurs paramètres et performance
print("Meilleurs paramètres pour Random Forest :", grid_search_rf.best_params_)
print("Meilleur score ROC-AUC :", grid_search_rf.best_score_)

# Entraînement final avec les meilleurs paramètres
best_rf = grid_search_rf.best_estimator_
best_rf.fit(X_train, y_train)
y_pred_rf = best_rf.predict(X_test)
```

```
Meilleurs paramètres pour Random Forest : {'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 5
0}
Meilleur score ROC-AUC : 0.9995525561711832
```

Après optimisation, le modèle Random Forest utilise les meilleurs hyperparamètres trouvés (max_depth=10, min_samples_leaf=2, min_samples_split=5, n_estimators=50), atteignant un score ROC-AUC de 0.9996, démontrant une performance exceptionnelle dans la détection des fraudes.

3.3 Gradient Boosting

```python
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import classification_report, confusion_matrix

# Réduction de la taille des données pour accélérer la Grid Search
X_train_sub, _, y_train_sub, _ = train_test_split(X_train, y_train, test_size=0.95, random_state=42)

# Définition des paramètres pour Grid Search
param_grid_gb = {
    'n_estimators': [50, 100, 150],  # Plus d'arbres
    'learning_rate': [0.01, 0.05, 0.1, 0.2],  # Plus de valeurs pour le taux d'apprentissage
    'max_depth': [3, 5, 7],  # Augmentation de la profondeur maximale
    'min_samples_split': [2, 5, 10],  # Taille minimale des divisions
    'min_samples_leaf': [1, 2, 4],  # Taille minimale des feuilles
    'subsample': [0.8, 1.0]  # Fraction d'échantillons utilisés pour chaque arbre
}

# Initialisation du modèle
gb = GradientBoostingClassifier(random_state=42)

# Grid Search sur les données réduites
grid_search_gb = GridSearchCV(gb, param_grid_gb, cv=2, scoring='roc_auc', n_jobs=-1)
grid_search_gb.fit(X_train_sub, y_train_sub)

# Meilleurs paramètres et performance
print("Meilleurs paramètres pour Gradient Boosting :", grid_search_gb.best_params_)
print("Meilleur score ROC-AUC :", grid_search_gb.best_score_)

# Entraînement final avec les meilleurs paramètres sur l'ensemble complet
best_gb = grid_search_gb.best_estimator_
best_gb.fit(X_train, y_train)
y_pred_gb = best_gb.predict(X_test)
```

```
Meilleurs paramètres pour Gradient Boosting : {'learning_rate': 0.2, 'max_depth': 7, 'min_samples_leaf': 4, 'min_samples_spl
it': 2, 'n_estimators': 150, 'subsample': 0.8}
Meilleur score ROC-AUC : 0.9998024639501446
```

Après optimisation, le modèle Gradient Boosting utilise les meilleurs hyperparamètres trouvés (`learning_rate=0.2`, `max_depth=7`, `min_samples_leaf=4`, `min_samples_split=2`, `n_estimators=150`, `subsample=0.8`), atteignant un score ROC-AUC de 0.9998, ce qui indique une précision exceptionnelle dans la détection des transactions frauduleuses.

3.4 Réseau de Neurones

```python
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix

# Standardisation des données
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Réduction de la taille des données pour accélérer la Grid Search
X_train_sub, _, y_train_sub, _ = train_test_split(X_train_scaled, y_train, test_size=0.95, random_state=42)

# Définition des paramètres pour Grid Search
param_grid_mlp = {
    'hidden_layer_sizes': [(32,), (64,), (32, 32), (64, 64)],  # Différentes configurations de couches
    'activation': ['tanh', 'relu'],  # Fonctions d'activation
    'solver': ['adam', 'sgd'],  # Méthodes d'optimisation
    'learning_rate_init': [0.001, 0.01],  # Taux d'apprentissage initial
    'alpha': [0.0001, 0.001]  # Régularisation L2 (contrôle du surapprentissage)
}

# Initialisation du modèle
mlp = MLPClassifier(max_iter=200, random_state=42)

# Grid Search pour l'optimisation des hyperparamètres
grid_search_mlp = GridSearchCV(mlp, param_grid_mlp, cv=2, scoring='roc_auc', n_jobs=-1)
grid_search_mlp.fit(X_train_sub, y_train_sub)

# Meilleurs paramètres et performance
print("Meilleurs paramètres pour Réseau de Neurones :", grid_search_mlp.best_params_)
print("Meilleur score ROC-AUC :", grid_search_mlp.best_score_)
```

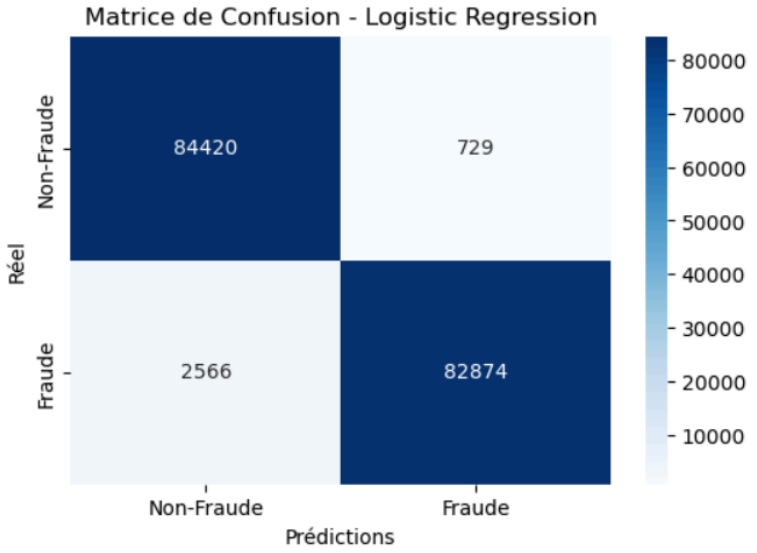
```python
# Entraînement final avec les meilleurs paramètres sur l'ensemble complet
best_mlp = grid_search_mlp.best_estimator_
best_mlp.fit(X_train_scaled, y_train)
y_pred_mlp = best_mlp.predict(X_test_scaled)
```

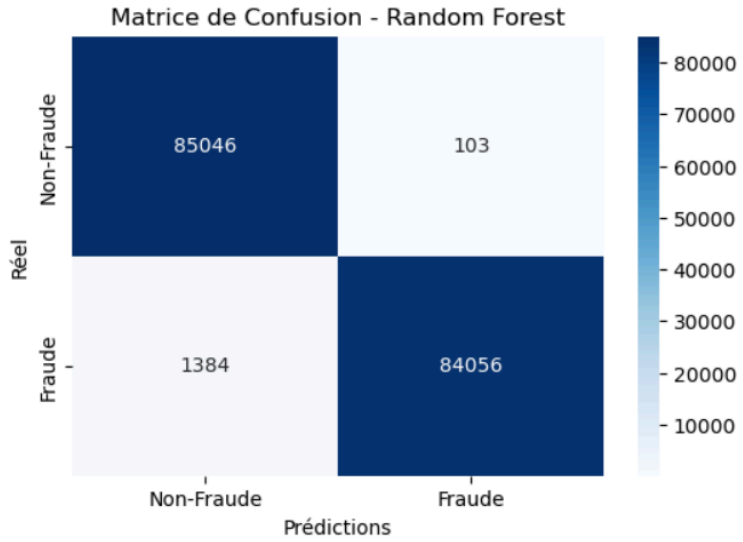
```
Meilleurs paramètres pour Réseau de Neurones : {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (64, 64), 'lear
ning_rate_init': 0.01, 'solver': 'adam'}
Meilleur score ROC-AUC : 0.999867117669705
```

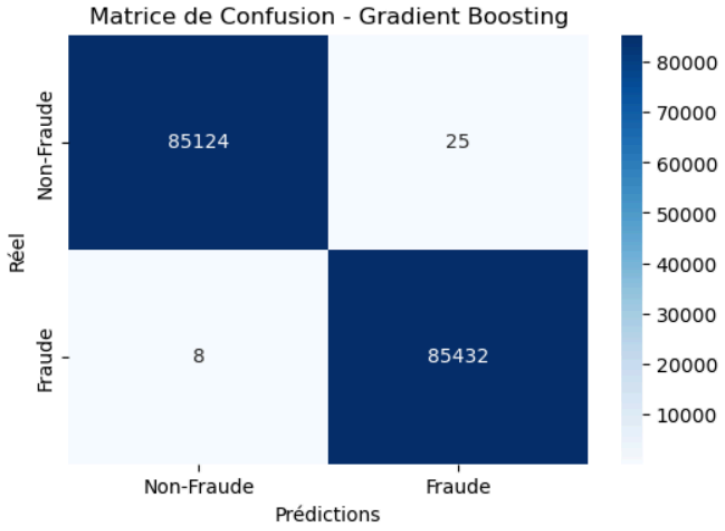
Après optimisation, le modèle Réseau de Neurones utilise les meilleurs hyperparamètres trouvés (`activation='tanh'`, `alpha=0.0001`, `hidden_layer_sizes=(64, 64)`, `learning_rate_init=0.01`, `solver='adam'`), atteignant un score ROC-AUC de 0.9998, ce qui montre son efficacité remarquable pour la détection des fraudes.

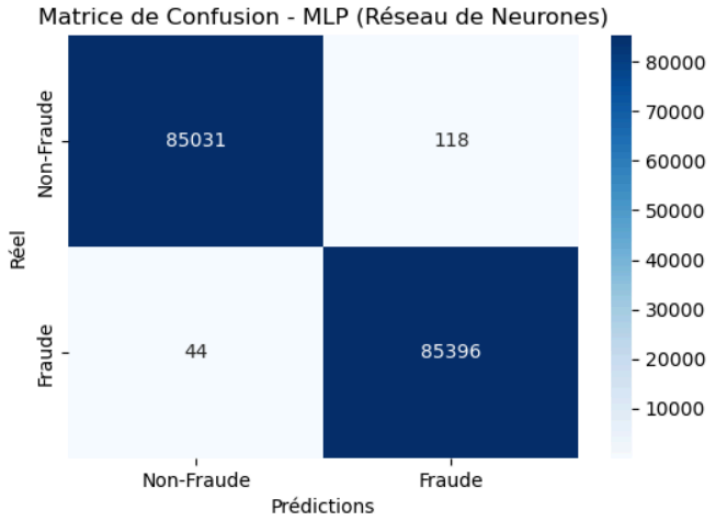
4. Visualisations et comparaison des résultats

4.1 Matrice de Confusion : Les matrices montrent une excellente séparation des classes, avec très peu de faux positifs et négatifs.









4.2 Comparaison des Scores AUC-ROC : Tous les modèles atteignent des performances élevées, mais **Gradient Boosting** et **Réseau de Neurones** se distinguent par des scores presque parfaits.

	Model	AUC-ROC	Accuracy	Precision	Recall	F1-Score
0	Logistic Regression	0.997507	0.980685	0.991280	0.969967	0.980508
1	Random Forest	0.999681	0.991283	0.998776	0.983801	0.991232
2	Gradient Boosting	0.999809	0.999807	0.999707	0.999906	0.999807
3	MLP	0.999972	0.999050	0.998620	0.999485	0.999052

4.3 Résultat final

Parmi les différents modèles évalués, les deux meilleurs sont le Gradient Boosting optimisé et le Réseau de Neurones (MLP) optimisé. Les performances de ces modèles se distinguent par leurs scores AUC-ROC, précision, rappel et F1-Score, ce qui en fait des choix solides pour la détection de fraudes bancaires.

Le Gradient Boosting optimisé a obtenu un AUC-ROC de 0.9998, une accuracy de 99.98%, et des scores de précision, rappel et F1-Score proches de 99.98%, ce qui indique une capacité exceptionnelle à prédire correctement les fraudes et à minimiser les faux positifs et les faux négatifs. Ce modèle est particulièrement efficace dans des scénarios où un équilibre parfait entre précision et rappel est requis.

Le Réseau de Neurones (MLP) optimisé, quant à lui, affiche des performances similaires avec un AUC-ROC légèrement supérieur à 0.9999 et une accuracy de 99.90%, démontrant une excellente capacité à différencier les transactions frauduleuses des transactions normales. Cependant, bien que son score AUC-ROC soit marginalement meilleur, son rappel et sa précision sont légèrement inférieurs à ceux du Gradient Boosting, ce qui pourrait entraîner un nombre légèrement plus élevé de faux négatifs.

4.4 Interprétation

Ces résultats montrent que les deux modèles sont parfaitement adaptés au problème de détection de fraude, mais le Gradient Boosting se démarque comme le modèle le plus équilibré en termes de performances globales. Son efficacité à capturer les fraudes avec un nombre minimal d'erreurs en fait un choix optimal pour des systèmes en production. Le Réseau de Neurones (MLP) reste néanmoins une alternative puissante, notamment dans des cas où des configurations matérielles robustes sont disponibles pour l'entraînement de modèles complexes.

Dans le cadre de ce projet, le Gradient Boosting est recommandé comme modèle principal, avec le Réseau de Neurones en tant que modèle secondaire pour confirmer ou affiner les prédictions.

IV . Exécution du projet (Streamlit)

Dans le cadre de ce projet, une application web interactive a été développée à l'aide de Streamlit pour permettre une utilisation pratique et intuitive des modèles de détection de fraude. Cette application offre trois principales fonctionnalités :

1. Page d'Accueil : Une interface de présentation expliquant les objectifs et les fonctionnalités de l'application.
2. Prédiction de Fraude : Une page permettant à l'utilisateur d'entrer les caractéristiques d'une transaction, comme le montant et le temps écoulé, afin d'obtenir une prédiction de fraude ou non-fraude.
3. Visualisation des Données : Une page permettant d'explorer les transactions et d'analyser leur distribution, notamment par rapport aux montants ou aux classes (fraudes et non-fraudes).

L'application développée avec **Streamlit** a été spécifiquement conçue pour fonctionner avec le dataset `creditcard.csv`, qui contient des transactions bancaires anonymisées et des variables dérivées par Analyse en Composantes Principales (PCA).

1. **Structure et Fonctionnalité du Code**

```python
import streamlit as st
import pandas as pd
import numpy as np
import joblib
import plotly.express as px
from io import BytesIO

# Charger les modèles et le scaler
logreg_model = joblib.load('optimized_logistic_regression.pkl')
rf_model = joblib.load('optimized_random_forest.pkl')
gb_model = joblib.load('optimized_gradient_boosting.pkl')
mlp_model = joblib.load('optimized_mlp.pkl')
scaler = joblib.load('scaler.pkl')

# Colonnes attendues par le modèle
expected_columns = ['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11',
                    'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20', 'V21', 'V22',
                    'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount', 'Amount_Zscore']
```

```python
# Paramètres globaux pour Amount_Zscore
mean_amount = 88.349619  # Moyenne du dataset original
std_amount = 250.120109  # Écart-type du dataset original

# Personnalisation de la mise en page
st.set_page_config(page_title="Détection de Fraude Bancaire", layout="wide")

# Ajouter une sidebar pour le menu
st.sidebar.title("Menu de Navigation")
menu_options = ["Accueil", "Prédiction de Fraude", "Visualisation des Données"]
menu_choice = st.sidebar.selectbox("Choisissez une page", menu_options)

# Page d'accueil
if menu_choice == "Accueil":
    st.title("Bienvenue dans l'application de Détection de Fraude Bancaire")
    st.markdown("""
    ### Fonctionnalités :
    - **Prédiction de Fraude :** Analysez une transaction pour détecter un comportement frauduleux.
    - **Visualisation des Données :** Explorez les caractéristiques des transactions.
    - Application optimisée pour les datasets avec des modèles pré-entraînés.

    ### Technologies utilisées :
    - **Streamlit** pour l'interface utilisateur.
    - **Scikit-learn** pour la création des modèles de machine learning.
    - **Bootstrap** pour un design moderne via Streamlit.

    **Développeur :** *Yasmine Ben Larbi*
    """)
```

```python
# Page de prédiction de fraude
elif menu_choice == "Prédiction de Fraude":
    st.title("Prédiction de Fraude Bancaire")
    st.markdown("""
    ### Entrez les détails de la transaction pour prédire une fraude :
    - Les variables PCA (V1 à V28) représentent des caractéristiques anonymisées par analyse en composantes principales.
    - La variable `Amount` représente le montant de la transaction.
    """)

    # Formulaire pour les entrées utilisateur
    with st.form("fraud_detection_form"):
        # Entrée pour le temps (Time)
        time = st.number_input("Temps (en secondes depuis la première transaction)", min_value=0.0, value=0.0, step=1.0)

        # Entrée pour le montant (Amount)
        amount = st.number_input("Montant de la transaction", min_value=0.0, value=0.0, step=1.0)

        # Bouton pour soumettre le formulaire
        submit_button = st.form_submit_button(label="Prédire")

    # Si le bouton est cliqué
    if submit_button:
        # Créer un DataFrame avec des valeurs par défaut pour les variables PCA
        default_pca_values = {f"V{i}": 0.0 for i in range(1, 29)}
        input_data = {
            "Time": time,
            "Amount": amount,
            **default_pca_values
        }
        input_df = pd.DataFrame([input_data])
```

```python
     # Ajouter Amount_Zscore
     input_df['Amount_Zscore'] = (input_df['Amount'] - mean_amount) / std_amount

     # Réorganiser les colonnes pour correspondre à celles utilisées pendant l'entraînement
     try:
         input_df = input_df[expected_columns]

         # Normaliser les données
         scaled_input = scaler.transform(input_df)

         # Prédictions avec chaque modèle
         predictions = {
             "Régression Logistique": logreg_model.predict(scaled_input)[0],
             "Random Forest": rf_model.predict(scaled_input)[0],
             "Gradient Boosting": gb_model.predict(scaled_input)[0],
             "Réseau de Neurones (MLP)": mlp_model.predict(scaled_input)[0]
         }

         # Afficher les résultats
         st.markdown("### Résultats des Prédictions")
         results = []
         for model_name, pred in predictions.items():
             result = "Fraude" if pred == 1 else "Non-Fraude"
             results.append({"Modèle": model_name, "Prédiction": result})
             st.write(f"- {model_name} : **{result}**")
```

```python
    # Générer un fichier CSV téléchargeable avec les résultats
    results_df = pd.DataFrame(results)
    csv = results_df.to_csv(index=False).encode('utf-8')
    st.download_button(
        label="Télécharger les résultats en CSV",
        data=csv,
        file_name='fraud_predictions.csv',
        mime='text/csv'
    )

except Exception as e:
    st.error(f"Une erreur s'est produite lors de la prédiction : {e}")
```

```python
# Page de visualisation des données
elif menu_choice == "Visualisation des Données":
    st.title("Visualisation des Données")
    st.markdown("### Analyse Exploratoire des Données (EDA)")

    # Charger le dataset réel
    try:
        data = pd.read_csv("creditcard.csv")

        # Sélectionnez le type de transactions à afficher (fraude ou non-fraude)
        st.sidebar.subheader("Filtre des transactions")
        filter_choice = st.sidebar.radio(
            "Afficher uniquement les transactions :",
            options=["Toutes", "Fraude", "Non-Fraude"]
        )

        # Appliquer le filtre
        if filter_choice == "Fraude":
            filtered_data = data[data["Class"] == 1]
        elif filter_choice == "Non-Fraude":
            filtered_data = data[data["Class"] == 0]
        else:
            filtered_data = data

        # Limiter les montants à 10,000 pour le graphique
        filtered_data = filtered_data[filtered_data['Amount'] <= 10000]

        # Histogramme du montant des transactions
        st.subheader("Distribution des montants des transactions")
        fig_amount = px.histogram(filtered_data, x="Amount", nbins=50, title="Montants des Transactions (<= 10,000)")
        st.plotly_chart(fig_amount)
```

```python
        # Analyse des fraudes vs non-fraudes
        st.subheader("Proportion des Fraudes")
        fraud_counts = filtered_data["Class"].value_counts()

        # Vérifiez si toutes les classes sont présentes
        if len(fraud_counts) == 2:
            labels = ["Non-Fraude", "Fraude"]
            fig_fraud = px.pie(
                names=labels,
                values=fraud_counts,
                title="Répartition des Fraudes"
            )
        elif len(fraud_counts) == 1:  # Une seule classe présente
            if 1 in fraud_counts.index:
                labels = ["Fraude"]
            else:
                labels = ["Non-Fraude"]
            fig_fraud = px.pie(
                names=labels,
                values=fraud_counts,
                title="Répartition des Fraudes"
            )
        else:  # Aucun enregistrement
            st.warning("Aucune donnée disponible pour les transactions sélectionnées.")
            fig_fraud = None

        if fig_fraud:
            st.plotly_chart(fig_fraud)
```

```python
    # Afficher les données brutes
    st.subheader("Aperçu des données filtrées")
    st.dataframe(filtered_data)

except FileNotFoundError:
    st.error("Erreur : Le fichier 'creditcard.csv' est introuvable. Veuillez vérifier son emplacement.")
except Exception as e:
    st.error(f"Une erreur s'est produite lors du chargement des données : {e}")
```

1.1 Importation des Librairies

- **Pandas** : Gestion des données (chargement et manipulation du dataset).
- **NumPy** : Calculs mathématiques pour le prétraitement.
- **Joblib** : Chargement des modèles pré-entraînés et du scaler.
- **Plotly Express** : Création de visualisations interactives.
- **Streamlit** : Framework pour créer l'interface utilisateur interactive.

1.2 Chargement des Modèles et Prétraitement

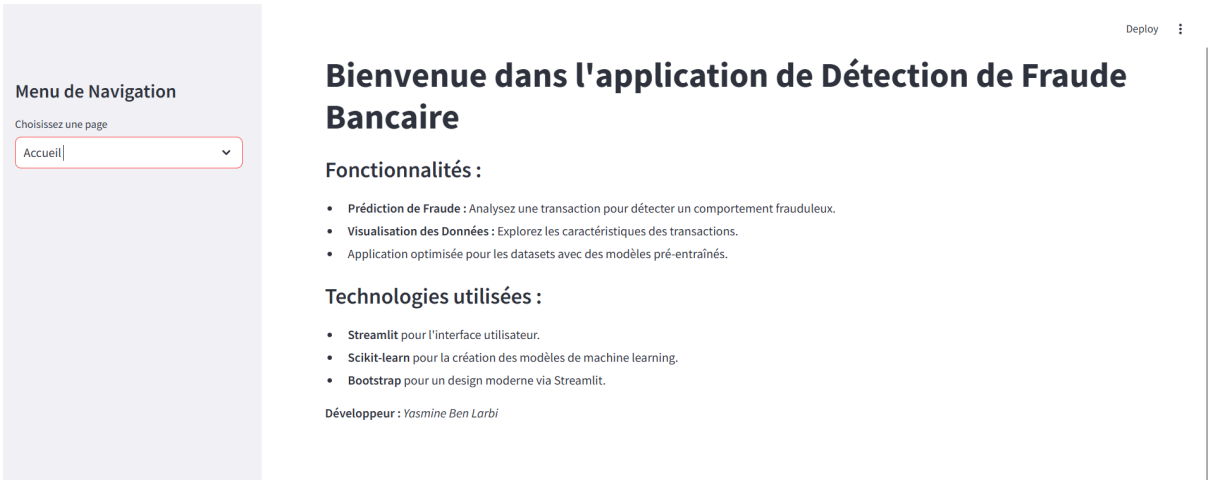
- **Modèles Chargés** :
  - Régression Logistique
  - Random Forest
  - Gradient Boosting
  - Réseau de Neurones (MLP)
- **Scaler** : Normalise les données pour améliorer la performance des modèles.

Les modèles ont été entraînés avec le dataset `creditcard.csv`. Les colonnes attendues (expected_columns) incluent des caractéristiques spécifiques au dataset comme Time, Amount, et les variables PCA (V1 à V28).

2. Interface Utilisateur avec Streamlit

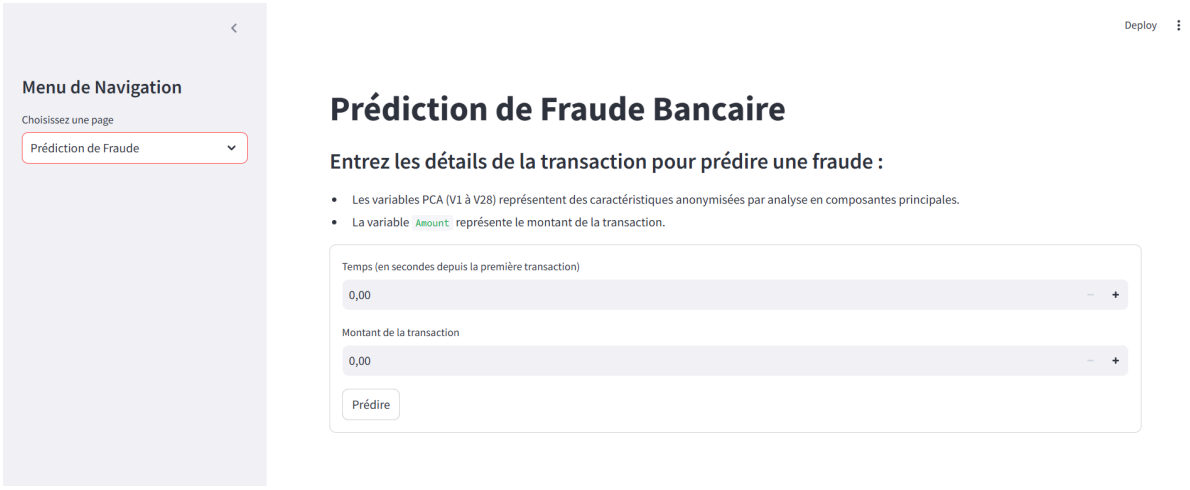
L'interface de l'application est intuitive et se divise en trois sections : une page d'accueil présentant les fonctionnalités, une page pour prédire si une transaction est frauduleuse en saisissant ses caractéristiques, et une section de visualisation pour explorer les données et analyser la répartition des fraudes.

a. Page d'Accueil



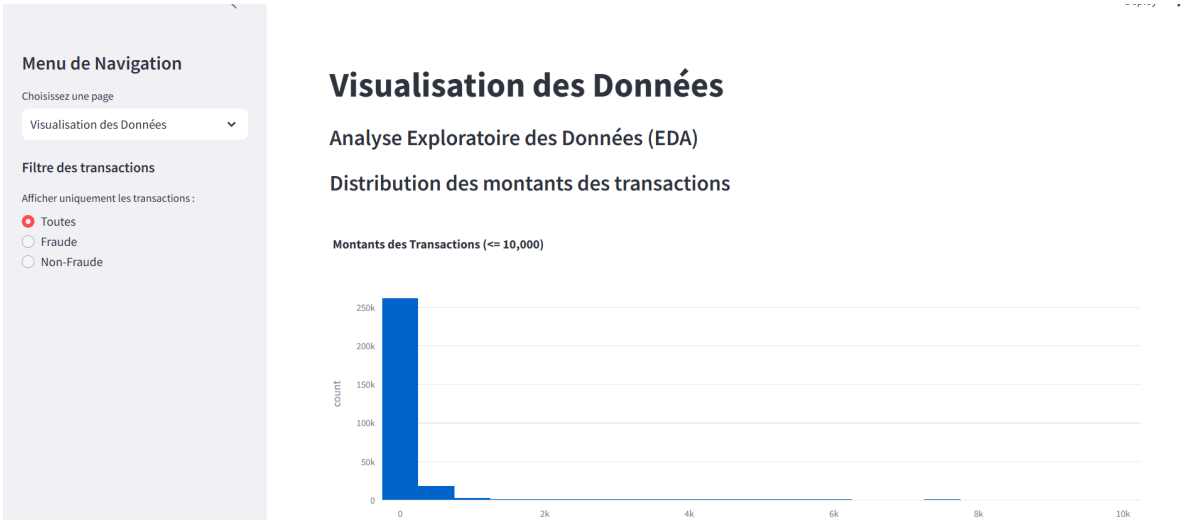
- Objectif : Introduire l'application.
- Contenu : Présentation des fonctionnalités et technologies utilisées (ex. Streamlit, Scikit-learn).

b. Prédiction de Fraude



- Formulaire :
  - L'utilisateur entre les détails d'une transaction, notamment Time (temps écoulé) et Amount (montant).
- Traitement :
  - Les variables PCA (V1 à V28) sont initialisées à 0.0 (par défaut).
- Prédiction :
  - Chaque modèle prédit si la transaction est frauduleuse ou non.
  - Les résultats sont affichés directement et peuvent être téléchargés en format CSV.

c. Visualisation des Données



Deploy :

Proportion des Fraudes

Répartition des Fraudes

Aperçu des données filtrées

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16
0	0	-1.3598	-0.0728	2.5363	1.3782	-0.3383	0.4624	0.2396	0.0987	0.3638	0.0908	-0.5516	-0.6178	-0.9914	-0.3112	1.4682	-0.4704
1	0	1.1919	0.2662	0.1665	0.4482	0.06	-0.0824	-0.0788	0.0851	-0.2554	-0.167	1.6127	1.0652	0.4891	-0.1438	0.6356	0.4639
2	1	-1.3584	-1.3402	1.7732	0.3798	-0.5032	1.8005	0.7915	0.2477	-1.5147	0.2076	0.6245	0.0661	0.7173	-0.1659	2.3459	-2.8901
3	1	-0.9663	-0.1852	1.793	-0.8633	-0.0103	1.2472	0.2376	0.3774	-1.387	-0.055	-0.2265	0.1782	0.5078	-0.2879	-0.6314	-1.0596
4	2	-1.1582	0.8777	1.5487	0.403	-0.4072	0.0959	0.5929	-0.2705	0.8177	0.7531	-0.8228	0.5382	1.3459	-1.1197	0.1751	-0.4514
5	2	-0.426	0.9605	1.1411	-0.1683	0.421	-0.0297	0.4762	0.2603	-0.5687	-0.3714	1.3413	0.3599	-0.3581	-0.1371	0.5176	0.4017
6	4	1.2297	0.141	0.0454	1.2026	0.1919	0.2727	-0.0052	0.0812	0.465	-0.0993	-1.4169	-0.1538	-0.7511	0.1674	0.0501	-0.4436
7	7	-0.6443	1.418	1.0744	-0.4922	0.9489	0.4281	1.1206	-3.8079	0.6154	1.2494	-0.6195	0.2915	1.758	-1.3239	0.6861	-0.0761
8	7	-0.8943	0.2862	-0.1132	-0.2715	2.6696	3.7218	0.3701	0.8511	-0.392	-0.4104	-0.7051	-0.1105	-0.2863	0.0744	-0.3288	-0.2101
9	9	-0.3383	1.1196	1.0444	-0.2222	0.4994	-0.2468	0.6516	0.0695	-0.7367	-0.3668	1.0176	0.8364	1.0068	-0.4435	0.1502	0.7395

- Filtrage :
    - Les transactions peuvent être filtrées par catégorie : "Toutes", "Fraude", ou "Non-Fraude".
- Visualisations :
    - Histogramme des montants des transactions.
    - Diagramme circulaire montrant la répartition des fraudes et non-fraudes.
- Aperçu des Données :
    - Les données filtrées sont affichées dans un tableau interactif.

3. Limitations de l'Application :

Spécificité des Données :

Cette application est conçue exclusivement pour le fichier `creditcard.csv`. Les modèles sont entraînés sur des variables anonymisées (V1 à V28) issues d'une PCA, qui ne sont pas interprétables dans un contexte différent.

Optimisation Limitée :

Bien que tous les modèles soient intégrés, le Gradient Boosting pourrait suffire comme seul modèle utilisé dans l'application, étant donné ses performances supérieures (AUC-ROC proche de 1).

Dépendance au Fichier Source :

L'application n'est pas adaptée à d'autres datasets sans modification préalable, car les variables d'entrée doivent être structurées de manière identique.

Erreurs liées au code :

Des erreurs dans le code actuel peuvent affecter la qualité des prédictions, notamment en raison de problèmes de compatibilité liés aux noms de colonnes ou au format des données utilisées pour la normalisation et les modèles pré-entraînés. Par conséquent, les résultats obtenus ne sont pas complètement fiables et nécessitent des ajustements pour garantir une utilisation correcte et réaliste.

Conclusion Générale

Ce projet de détection de fraude bancaire a permis de développer et d'évaluer plusieurs modèles de machine learning sur un dataset contenant des transactions anonymisées. Les résultats montrent que les modèles tels que le Gradient Boosting et le Réseau de Neurones offrent des performances très élevées en termes de détection de fraude grâce à des scores AUC-ROC proches de 1. Cependant, ces résultats doivent être interprétés avec prudence.

L'utilisation de l'Analyse en Composantes Principales (PCA) pour anonymiser les données a permis de préserver la confidentialité des informations, mais elle a également introduit des limitations importantes. Les variables issues de la PCA (V1 à V28) ne représentent pas des caractéristiques concrètes, ce qui réduit la capacité des modèles à capturer certains aspects contextuels essentiels (par exemple, le type de transaction, la localisation ou le profil du client). Par conséquent, bien que les performances soient élevées sur le dataset fourni, elles pourraient ne pas être aussi fiables dans un environnement réel.

De plus, certaines limitations techniques ont été observées, comme des erreurs de code ou des avertissements liés à la gestion des colonnes, qui pourraient influencer les prédictions finales. Enfin, pour améliorer la précision et l'efficacité de l'application, des données supplémentaires et concrètes, telles que des métadonnées transactionnelles ou des informations sur les clients, seraient nécessaires. Cela permettrait de mieux contextualiser les prédictions et d'augmenter la robustesse du système dans des situations réelles.