

INTRODUCTION

Dans le monde financier, la fraude bancaire doit répondre à de nombreux problèmes. Elle touche autant les banques que les clients. Cela est dû aux arnaques de transaction qui peuvent être complexes à identifier. Par conséquent, il est important de repérer ces fraudes.

Ce projet a pour but de créer un modèle qui peut aider à la détection de ces transactions suspectes, avec des données d'analyse et d'apprentissage automatique.

This slide provides a high-level overview of the project. It features a large title "SOMMAIRE" at the top. Below it is a grid of various data visualizations including heatmaps, histograms, and scatter plots. To the right is a vertical flowchart with four main steps: 1. Nettoyage des données, Compréhension des tendances; 2. Modélisation du jeu de données, test et prédition; 3. Validation de la prédition; and 4. Conclusion résultat. The bottom of the slide includes the year "2024", credit to "Crédit du DataFrame Kaggle", and the author's name "Marvin Hugues Laurac".

LOGICIEL PRIVILÉGIÉ

- Python

J'ai utilisé Python avec ces bibliothèques, comme Pandas pour l'analyse de données, Scikit-Learn pour le machine learning, et Matplotlib pour la visualisation.

CONTEXTE DU SUJET

Ce projet représente les transactions par carte de crédit réalisées en septembre 2013 par des détenteurs européens. Parmi les 284 807 transactions enregistrées sur deux jours, 492 sont identifiées comme frauduleuses, ce qui indique un déséquilibre puisque les fraudes ne représentent que 0,172 % du total. Les données sont principalement constituées de variables numériques issues d'une transformation par **Analyse en Composantes Principales** ACP, à l'exception de « Time » et « Amount ». Les variables V1 à V28 sont le résultat de cette ACP, tandis que « Time » indique les secondes écoulées depuis la première transaction de l'ensemble de données et 'Amount' reflète le montant de chaque transaction. Cet ensemble de données présente un défi en raison de son déséquilibre et des limitations liées à la confidentialité, qui empêchent l'accès aux caractéristiques originales et à des informations plus détaillées. La variable cible, « Class », est binaire, 1 indique une transaction frauduleuse et 0 une transaction légitime.

À noter: Ces informations ont été indiquées par l'auteur. De mon côté, j'ai travaillé sur ce DataFrame en prenant en compte ces indications, en les reformulant et en faisant mes recherches de mon côté

Recommandations de l'auteur du DataFrame

De plus, l'efficacité de la détection de la fraude ne peut être évaluée par la précision seule, étant donné le déséquilibre des classes. Il est donc recommandé d'utiliser l'aire sous la courbe de précision-rappel AUPRC comme métrique principale. Cette approche offre une évaluation plus fiable de la performance du modèle, en particulier dans les cas de classification déséquilibrée, comme c'est le cas avec cet ensemble de données.

DATAFRAME INFO

```
<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
 27  V27    284807 non-null  float64
 28  V28    284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

LICENCE

kaggle (Database Contents License (DbCL) v1.0)

Ce projet a une deadline de 5 jours
pour me challenger, Go!

Marvin Hugues Laurac

```
Entrée [1]: import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.metrics import roc_auc_score, roc_curve

Entrée [2]: #je commence par importer le DataFrame
df = pd.read_csv('/Users/octoberone/Desktop/DataFrame GitHub/DataFrame/Détection de fraude bancaire.csv')

Entrée [3]: #affichage des 5 premières lignes
df.head()

Out[3]:
   Time      V1      V2      V3      V4      V5      V6      V7      V8      V9 ...     V21      V22      V23      V24
0  0.0  -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599  0.098698  0.363787 ... -0.018307  0.277838 -0.110474  0.066928  0.12
1  0.0   1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803  0.085102 -0.255425 ... -0.225775 -0.638672  0.101288 -0.339846  0.16
2  1.0  -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461  0.247676 -1.514654 ...  0.247998  0.771679  0.909412 -0.689281 -0.32
3  1.0  -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609  0.377436 -1.387024 ... -0.108300  0.005274 -0.190321 -1.175575  0.64
4  2.0  -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941 -0.270533  0.817739 ... -0.009431  0.798278 -0.137458  0.141267 -0.20

5 rows × 31 columns

Entrée [4]: #voici le nombre de lignes et de colonnes sans le détail
df.shape

Out[4]: (284807, 31)
```



```
Entrée [5]: #voici les détails des colonnes avec beaucoup plus d'informations (variables, types, le nombre de valeurs nulles)
df.info()

<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
 27  V27    284807 non-null  float64
 28  V28    284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```



```
Entrée [6]: #par souci du détail, j'ai utilisé cette commande pour être sûr qu'il n'y a pas de valeurs manquantes
df.isnull().sum()

#par conséquent, j'en déduis que ce DataFrame est plutôt agréable à utiliser dans un premier temps

Out[6]: Time      0
V1       0
V2       0
V3       0
V4       0
V5       0
V6       0
V7       0
V8       0
V9       0
V10      0
V11      0
V12      0
V13      0
V14      0
V15      0
V16      0
V17      0
V18      0
V19      0
V20      0
V21      0
V22      0
V23      0
V24      0
V25      0
V26      0
V27      0
V28      0
Amount   0
Class    0
dtype: int64
```

DÉTECTION DE FRAUDE PAR CARTE BANCAIRE

Entrée [7]: #Afin d'avoir plus de détails statistiques, j'affiche les colonnes numériques

```
#count%:nombre de valeurs nulles
#mean%:la moyenne des valeurs
#std%:l'écart type (dispersion valeurs)
#min%:le min de la colonne
#25%:1er quartile à moins de 25
#50%:2er quartile à moins de 25
#75%:3er quartile à moins de 25
#max:le maxi de la colonne

print(df.describe())

      Time          V1          V2          V3          V4  \
count  284807.000000  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 -8.983648e-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   9.604966e-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
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-01  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       Amount  \
count 2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  284807.000000
mean 5.340915e-16  1.683437e-15 -3.660091e-16 -1.227390e-16   88.349619
std  5.212781e-01  4.822270e-01  4.036325e-01  3.300833e-01  250.120109
min -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01   0.000000
25% -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02   5.600000
50%  1.659350e-02 -5.213911e-02  1.342146e-03  1.124383e-02  22.000000
75%  3.507156e-01  2.409522e-01  9.104512e-02  7.827995e-02  77.165000
max  7.519589e+00  3.517346e+00  3.161220e+01  3.384781e+01  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]

DÉTECTION DE FRAUDE PAR CARTE BANCAIRE

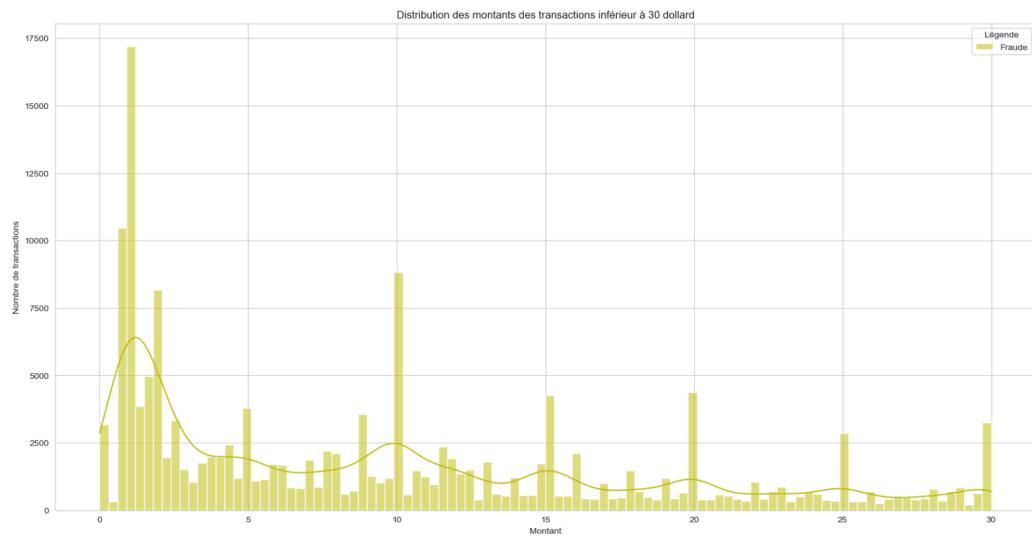
Entrée [22]: #premier affichage d'histogramme afin de visualiser la tendance et son évolution dans le temps
#conclusion : La majorité des transactions sont inférieures à 30 dollars,
avec un nombre important de transactions inférieures à 5 dollars, allant jusqu'à 17 500 transactions

```
plt.figure(figsize=(20, 10))

sns.histplot(df[df["Amount"] <= 30]["Amount"], bins=100, kde=True, color= "y", label="Fraude")

sns.set_style("whitegrid")
plt.title("Distribution des montants des transactions inférieur à 30 dollar")
plt.xlabel("Montant")
plt.ylabel("Nombre de transactions")
plt.legend(title="Légende")

plt.show()
```



Entrée [9]: #dans cette étape, je vérifie la présence des deux classes
#de plus je les affiche, j'en déduis que la classe 1 est celle qui a des transactions frauduleuses

```
class_ = df["Class"].value_counts()

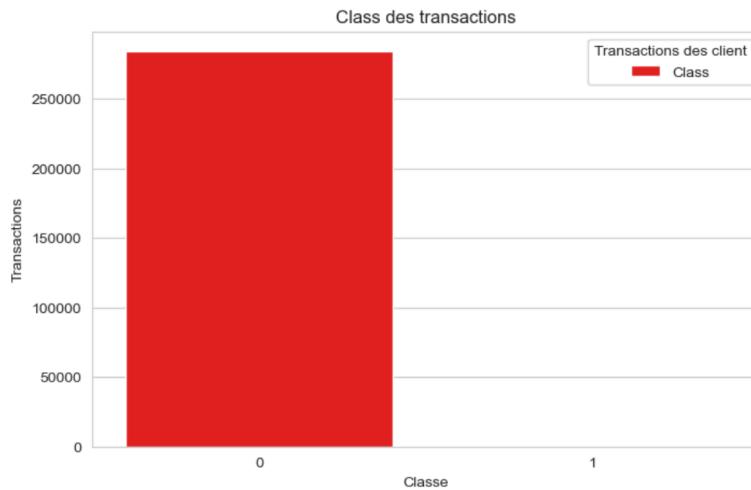
if len(class_) == 2:
    print("il y à deux classe")
    for class_index in class_.index:
        print(f"Class {class_index}: ", class_[class_index])
else:
    print("il y à pas deux classe")

il y à deux classe
Class 0: 284315
Class 1: 492
```

```
Entrée [10]: #Par cela, j'affiche dans un histogramme les classes, dans celui-ci malheureusement la faible présence de transactionne ne permet pas de les apercevoir visuellement
plt.figure(figsize=(8, 5))
sns.countplot(x="Class", color= "red", label="Class", data=df)

sns.set_style("whitegrid")
plt.title("Class des transactions")
plt.xlabel("Classe")
plt.ylabel("Transactions")
plt.legend(title="Transactions des client")

plt.show()
```



```
Entrée [11]: #premièrement, je vais séparer les colonnes dans la variable X sauf "Class" dans Y
#pour isoler les caractéristiques de l'entraînement
X = df.drop("Class", axis=1)
y = df["Class"]
```

```
Entrée [12]: #maintenant je vais fractionner le jeu de données
#pour tester la performance de la prédiction
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
Entrée [13]: #Ensuite une normalisation
#pour améliorer les résultats
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
Entrée [14]: #et enfin un entraînement du modèle
#pour finaliser notre prédiction
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
```

```
Out[14]: RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
Entrée [15]: #après avoir fait cela, il suffit juste de prédire les transactions frauduleuses sur la prédiction établie
y_pred = model.predict(X_test)
```

```
Entrée [16]: #Voici les affichages de ce travail:
#Un accuracy score: pour mesurer la proportion totale (prédiction correcte / nombre total de prédictions)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

Accuracy: 1.00

```
Entrée [17]: #Un classification report: pour mesurer la prédiction avec plus de détail (précision, rappel, score F1, support)
class_report = classification_report(y_test, y_pred)
print("Classification Report:\n", class_report)
```

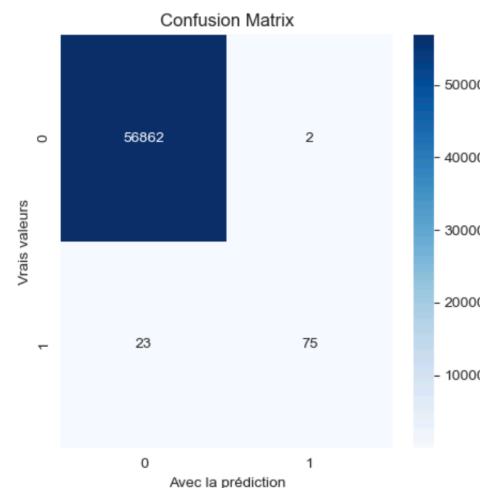
	precision	recall	f1-score	support
0	1.00	1.00	1.00	56864
1	0.97	0.77	0.86	98
accuracy			1.00	56962
macro avg	0.99	0.88	0.93	56962
weighted avg	1.00	1.00	1.00	56962

```
Entrée [18]: #une confusion matrix: pour avoir un visuel sur la prédiction correcte et les erreurs dont :
#56862: sont identifiées correctement comme non frauduleuses (BON)
#2: sont identifiées comme non frauduleuses mais sont frauduleuses (FAUX)
#23:sont identifiées comme frauduleuses mais sont non frauduleuses (FAUX)
#75: sont identifiées correctement comme frauduleuses (BON)
```

```
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", conf_matrix)
```

```
Confusion Matrix:
[[56862 2]
 [ 23 75]]
```

```
Entrée [19]: plt.figure(figsize=(5, 5))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d", cmap="Blues", cbar=True)
plt.xlabel("Avec la prédiction")
plt.ylabel("Vrais valeurs")
plt.title("Confusion Matrix")
plt.show()
```



```
Entrée [20]: #dans ce partie de code, je vais utiliser le model entrainer précédemment
#pour générer la probabilité prédicté du model
#puis le score auc, pour évaluer le model (positif, négatif)
y_probs = modèle.predict_proba(X_test)[:, 1]
auc = roc_auc_score(y_test, y_probs)
```

```
Entrée [21]: #et enfin l'affichage des deux courbe pour savoir le bon et le moins bon de la prédiction
#dans ce graphique nous pouvons apprécier une bonne performance de prédiction
fpr, tpr, _ = roc_curve(y_test, y_probs)
```

```
plt.figure(figsize=(20, 5))
plt.plot(fpr, tpr, color="y", label="ROC")
plt.plot([0, 1], [0, 1], color="r", linestyle="--")
plt.xlabel("Faux positif")
plt.ylabel("Vrai Positif")
plt.title("Courbe ROC")
plt.legend(title="Légende")
plt.show()
```

