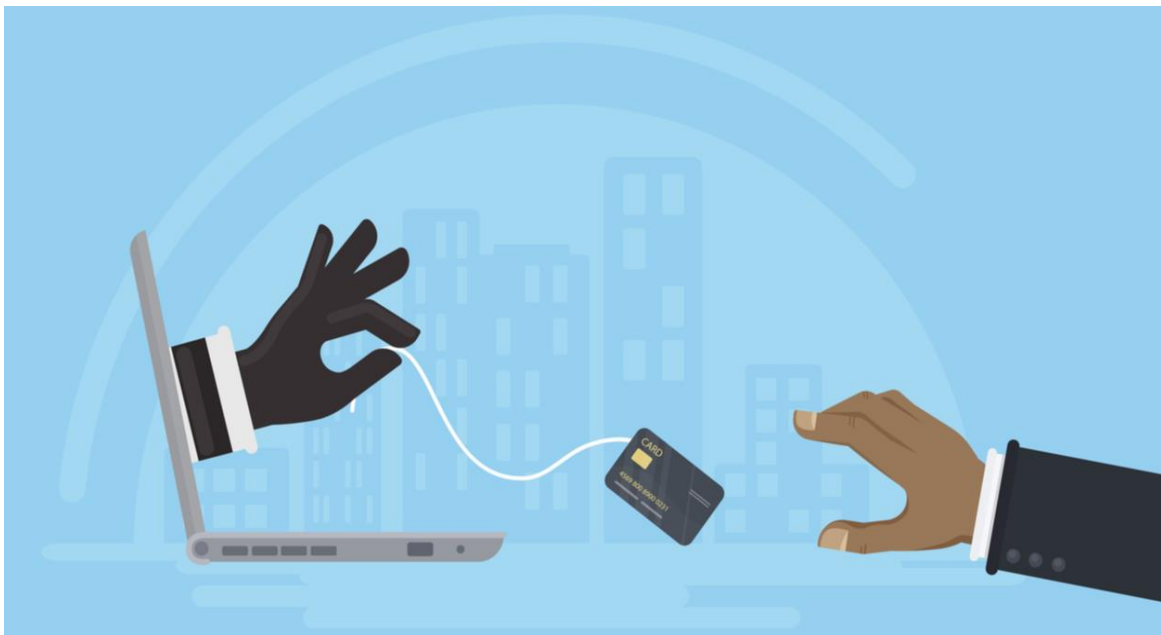


Rapport du projet fin formation : Détection de fraude par carte de crédit



AUTEUR :

- Abdelaziz BELHAJ
- Houssam-Eddine MOUMANIS

ENCADRANT :

- Prof. Abdelghani GHAZDALI

Remerciement

On remercie dieu le tout puissant de nous avoir donné la sante et la volonté d'entamer et de terminer ce projet. Je souhaite aussi adresser mes remerciements les plus sincères aux personnes qui m'ont apporté leur aide pour mener à bien ce projet et à l'élaboration du rapport et de ce projet.

Table des matières

- Introduction : Pourquoi avons-nous besoin de trouver des transactions frauduleuses ?
- Approches de détection de fraude

- Définition : Qu'est-ce que la détection de fraude par carte de crédit ?
- Les Bibliothèques utilisées dans python
- Compréhension et exploration des données de l'ensemble de données de carte de crédit
- Prétraitement des données de carte de crédit
 - Supprimer les colonnes/fonctionnalités non pertinentes
 - Vérification des valeurs nulles ou nan
- Fractionnement de l'ensemble de données
 - Transformation des données
- Présentation de l'algorithme de régression logistique
- Présentation de l'algorithme arbre de décision
- Algorithme régression logistique Implémentation à l'aide de la bibliothèque python sklearn
- Algorithme arbre de décision Implémentation à l'aide de la bibliothèque python sklearn
- Métriques d'évaluation appropriées pour les données.
- Amélioration du modèle à l'aide de techniques d'échantillonnage
- **Oversampling & Undersampling**
- Classification avec régression logistique et Arbre de décision après application des techniques d'échantillonnage
- Conclusion

A. Introduction générale

La fraude est une menace qui doit être prise au sérieux et justifie une action immédiate de la part des agences/entreprises et des organes législatifs ou

directeurs. Les effets qu'elle peut avoir dans le système agences/entreprises peuvent être considérables.

En plus des pertes financières qu'elle occasionne, elle a des effets négatifs sur la réputation de l'organisation où elle se déroule, en mettant en péril sa capacité à entreprendre des programmes avec toute l'efficacité souhaitée, à créer des partenariats et à se développer. Recevoir des cotisations.

C'est pour cette raison que des mécanismes efficaces de prévention, de détection et de répression de la fraude jouent un rôle essentiel dans la protection des intérêts des organisations contre ces effets négatifs.

Les mesures d'exécution jouent également un rôle important dans l'amélioration de l'efficacité du système des agences/entreprises et sa capacité à assumer ses responsabilités, ainsi qu'à encourager l'exercice de fonctions de surveillance appropriées et l'utilisation responsable des ressources.

B. Approches de détection de fraude

Les entreprises commencent donc à détecter automatiquement ces activités frauduleuses en utilisant des technologies intelligentes.

Premièrement, les entreprises embauchent peu de personnes uniquement pour la détection de ce type d'activités ou de transactions. Mais ici, ils doivent être des experts dans ce domaine, et l'équipe doit également avoir une connaissance de la manière dont les fraudes se produisent dans des domaines particuliers. Cela nécessite plus de ressources, telles que les efforts et le temps des personnes.

Deuxièmement, les entreprises ont remplacé les processus manuels par des solutions basées sur des règles. Mais celui-ci échoue aussi la plupart du temps à détecter les fraudes.

Parce que dans le monde réel, la façon de faire des fraudes change radicalement de jour en jour. Ces systèmes basés sur des règles suivent certaines règles et conditions. Si un nouveau processus de fraude est différent des autres, ces systèmes échouent. Cela nécessite d'ajouter cette nouvelle règle pour coder et exécuter.

Aujourd'hui, les entreprises tentent d'adopter des algorithmes d'intelligence artificielle ou d'apprentissage automatique pour détecter les fraudes. Les algorithmes d'apprentissage automatique ont très bien fonctionné pour ce type de problème.

C. Définition de la détection de fraude par carte de crédit

La détection de fraude par carte de crédit est la nécessité d'identifier les activités frauduleuses. Le problème de classification des fraudes par carte de crédit est utilisé pour détecter les transactions frauduleuses ou les activités frauduleuses avant qu'elles ne deviennent un problème majeur pour les sociétés O de cartes de crédit, Il utilise la combinaison de transactions frauduleuses et non frauduleuses à partir des données historiques avec les données de transaction par carte de crédit de différentes personnes pour estimer la fraude ou la non-fraude sur les transactions par carte de crédit.

D. Compréhension et exploration des données de l'ensemble de données de carte de crédit

Pour ce problème de classification de fraude par carte de crédit, nous utilisons l'ensemble de données qui a été téléchargé à partir de la plateforme Kaggle.

Avant de passer à la partie développement du modèle, nous devrions avoir quelques connaissances sur notre ensemble de données.

L'ensemble de données à 284807 lignes et 31 caractéristiques, Nous avons utilisé la fonction `read_csv` de pandas pour lire le fichier et La commande ci-dessous `Head()` ne présente que cinq lignes par défaut, donne 5 échantillons.

```
data = pd.read_csv(r'./creditcard.csv')
df = data.copy() # To keep the data as backup
df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255427
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514651
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387079
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817776

5 rows × 31 columns

V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0

À partir de là, nous savons que `Class` est la variable cible et que les autres sont toutes des caractéristiques de notre ensemble de données.

La variable cible `Class` a des valeurs **0** et **1**. Ici

- 0 pour **les** transactions **non frauduleuses**
- 1 pour **les** transactions **frauduleuses**

Parce que nous visons à trouver des transactions frauduleuses, la valeur cible de l'ensemble de données à une valeur positive pour cela.

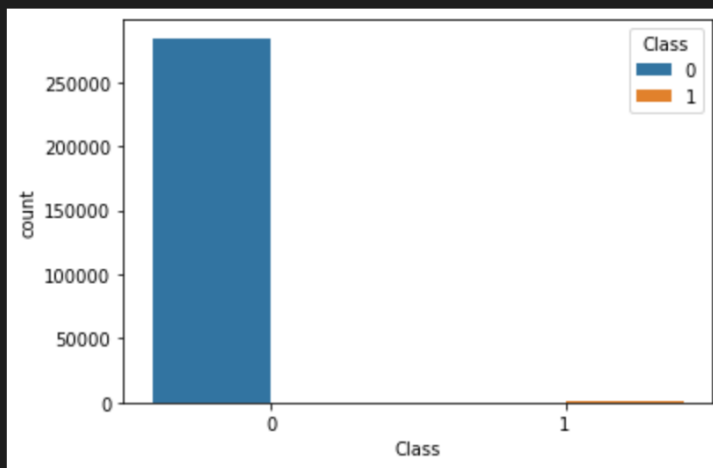
Ensuite on a vérifié le nombre d'échantillons de chaque classe cible et on a trouvé qu'il y a 284315 échantillons de transactions non frauduleuses et 492 échantillons frauduleuses:

```
[8] In [ ]: df.Class.value_counts()

0    284315
1      492
Name: Class, dtype: int64

[9] In [ ]: sns.countplot(x=df.Class, hue=df.Class)

<AxesSubplot:xlabel='Class', ylabel='count'>
```



E. Prétraitement des données de carte de crédit

Le prétraitement est le processus de nettoyage de l'ensemble de données. Dans cette étape, nous appliquerons différentes méthodes pour nettoyer les données brutes afin de fournir des données plus significatives pour la phase de modélisation. Cette méthode comprend la suppression des doublons ou les échantillons non pertinents et la mise à jour les valeurs manquantes avec les valeurs les plus pertinentes.

1. Supprimer les colonnes/fonctionnalités non pertinentes

Dans notre ensemble de données, une seule fonctionnalité non pertinente ou non utile id Time. Donc on l'a supprimé cette fonctionnalité de l'ensemble de données.

```
[48] ▶ ▶ ML
#df.shape
df = df.drop(labels='Time', axis=1)
df
```

	V1	V2	V3	V4	V5	V6	V7
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941
...

2. Vérification des valeurs nulles ou nan

Les valeurs nulles ou nan ne sont rien, mais il n'y a pas de valeur pour cette caractéristique ou cet attribut particulier.


```
[5] df.isnull().sum()

V1      0
V2      0
V3      0
V4      0
V5      0
V6      0
V7      0
V8      0
V9      0
```

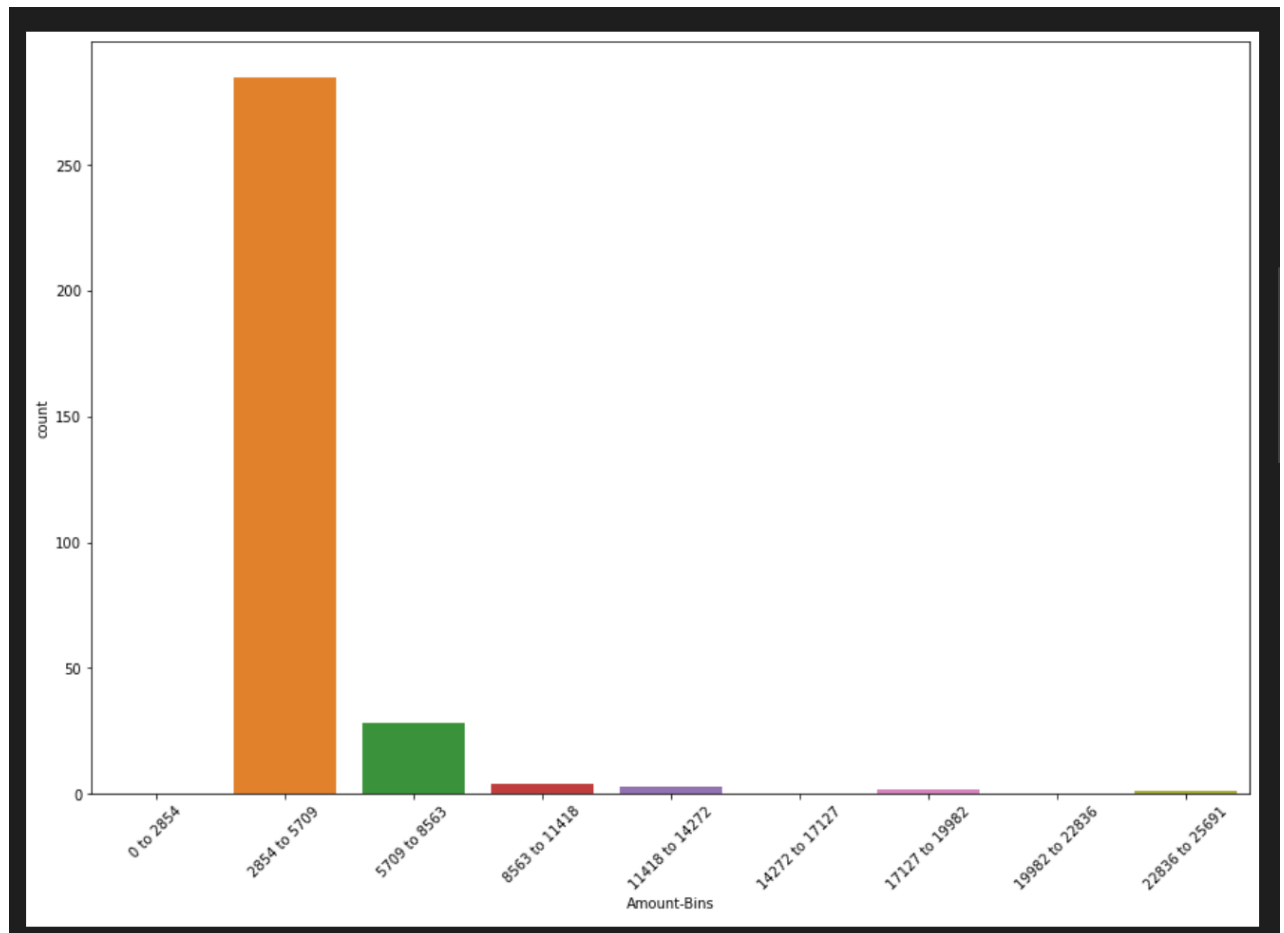
...

```
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
```

3. Génération de nouvelles variables (Features) :

L'ajout des caractéristiques pertinentes aide à améliorer la précision du modèle. Pour cela, on a ajouté de nouvelles colonnes pour compléter les informations sur les transactions.

Les variables qui sont ajoutées sont les classes du montant effectuées lors de chaque transaction, une visualisation sur notre Dataset avec les différentes classes du montant ou on l'a appelée « Amount-bins xxxx-xxxx » sous python :



- Après l'ajout des variables avec la fonction `make_bins ()` notre Dataset ressemble à :

```

▶ ML
df_encoded = pd.get_dummies(data=df, columns=['Amount-Bins'])
df = df_encoded.copy()

▶ ML
df.head()

```

V8	V9	V10	...	Class	Amount-Bins_0 to 2854	Amount-Bins_2854 to 5709	Amount-Bins_5709 to 8563	Amount-Bins_8563 to 11418	Amount-Bins_11418 to 14187
0.098698	0.363787	0.090794	...	0	1	0	0	0	0
0.085102	-0.255425	-0.166974	...	0	1	0	0	0	0
0.247676	-1.514654	0.207643	...	0	1	0	0	0	0
0.377436	-1.387024	-0.054952	...	0	1	0	0	0	0
0.270533	0.817739	0.753074	...	0	1	0	0	0	0

F. Fractionnement de l'ensemble de données

Nous avons pris toutes les colonnes indépendantes (la colonne cible est dépendante et les autres sont toutes des colonnes indépendantes les unes des autres), comme X et la variable cible comme y.

```

▶ ML
X = df.drop(labels='Class', axis=1)
Y = df['Class']
X.shape, Y.shape

((284807, 38), (284807,))

```

Transformation des données

À l'exception de la colonne Montant qui a des valeurs un peu grandes par rapport aux autres, nous avons modifié donc les valeurs des colonnes en une plage de nombres plus petite.

Nous pouvons simplement effectuer ce processus en utilisant StandardScaler de la bibliothèque Sklearn.

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(X)
```

Ensuite, nous avons divisé l'ensemble de données en ensemble de données d'entraînement et de test. Les données d'entraînement sont utilisées au moment de la construction du modèle et un ensemble de données de test est utilisé pour évaluer les modèles entraînés.

En utilisant la méthode train_test_split de la bibliothèque sklearn, nous avons pu effectuer ce processus de division de l'ensemble de données pour former et tester des ensembles.

```
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(
    X, Y, random_state=0, test_size=0.3, shuffle=True)

print(xtrain.shape, ytrain.shape)
print(xtest.shape, ytest.shape)
(199364, 38) (199364,)
(85443, 38) (85443,)
```

Notre jeu de données est maintenant prêt pour la création de modèles. Passons au développement du modèle à l'aide d'algorithmes d'apprentissage automatique tels que régression logistique.

G. Présentation de l'algorithme de régression logistique

La régression logistique est une technique prédictive. Elle vise à construire un modèle permettant de prédire et expliquer les valeurs prises par une variable cible qualitative (le plus souvent binaire, on parle alors de régression logistique binaire, si elle possède plus de 2 modalités, on parle de régression logistique polytomique) à partir d'un ensemble de variables explicatives quantitatives ou qualitatives.

H. Présentation de l'algorithme d'arbre de décision

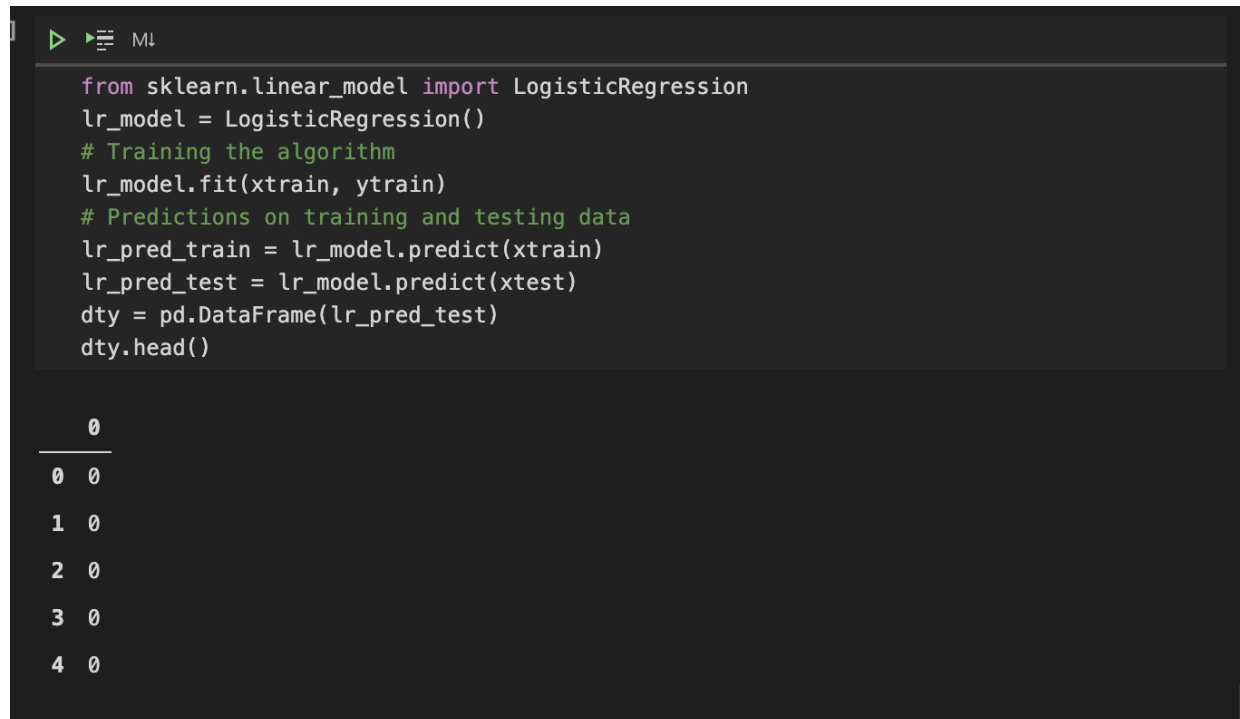
Cet outil d'aide à la décision ou d'exploration de données permet de représenter un ensemble de choix sous la forme graphique d'un arbre. C'est une des méthodes d'apprentissage supervisé les plus populaires pour les problèmes de classification de données.

Concrètement, un arbre de décision modélise une hiérarchie de tests pour prédire un résultat.

Les arbres de classification (Classification Tree) permettent de prédire à quelle classe la variable de sortie appartient (cela permet par exemple de répartir une population d'individus, comme des clients d'une entreprise en différents types de profils).

I. Algorithme régression logistique Implémentation à l'aide de la bibliothèque python sklearn

Nous utiliserons la classe `LogisticRegression` de la bibliothèque `sklearn` pour entraîner et évaluer des modèles. Nous utilisons les données `xtrain` et `ytrain` à des fins de formation. `xtrain` est un jeu de données d'entraînement avec des caractéristiques et `ytrain` est l'étiquette cible.



```
from sklearn.linear_model import LogisticRegression
lr_model = LogisticRegression()
# Training the algorithm
lr_model.fit(xtrain, ytrain)
# Predictions on training and testing data
lr_pred_train = lr_model.predict(xtrain)
lr_pred_test = lr_model.predict(xtest)
dty = pd.DataFrame(lr_pred_test)
dty.head()
```

	0
0	0
1	0
2	0
3	0
4	0

J. Algorithme Arbre Implémentation à l'aide de la bibliothèque python sklearn

```

> ML
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(xtrain,ytrain)
y_pred = classifier.predict(xtest)
dtyy = pd.DataFrame(y_pred)
dtyy.head()

```

	0
0	0
1	0
2	0
3	0
4	0

K. Métriques d'évaluation appropriées pour les données

1 - Matrice de confusion

La matrice de confusion (ou confusion matrix en anglais) est un tableau à double entrée vérifiant la fréquence des prédictions correctes par rapport à la réalité.

		Prédiction		
		reussite	échec	total
Réalité	reussite'	True Positive	False Negative	Réussite'
	échec'	False Positive	True Negative	Échec'
total		Réussite	Échec	

- TP (Vrais Positifs) : un vrai positif est un résultat où le modèle prédit correctement la classe positive.
- TN (Vrais Négatifs) : un vrai négatif est un résultat où le modèle prédit correctement la classe négative.
- FP (Faux Positifs) : un faux positif est un résultat où le modèle prédit incorrectement la classe positive
- FN (Faux Négatifs) : un faux négatif est un résultat où le modèle prédit incorrectement la classe négative La lecture du nombre de prédictions peut se faire en diagonale.

Les prédictions correctes : TP + TN et celles incorrectes : FN + FP.

On a utilisé les métriques mentionnées ci-dessous pour les ensembles de données :

- Recall
- Précision
- F1-score

Sous python pour appliquer les métriques et la matrice de confusion on a appelé la classe Metrics de la bibliothèque sklearn et voilà le résultat :

a) Pour Régression logistique :


```
ML
tn, fp, fn, tp = confusion_matrix(ytest, lr_pred_test).ravel()
conf_matrix = pd.DataFrame(
    {
        'Predicted Fraud': [tp, fp],
        'Predicted Not Fraud': [fn, tn]
    }, index=['Fraud', 'Not Fraud'])
conf_matrix
```

	Predicted Fraud	Predicted Not Fraud
Fraud	91	56
Not Fraud	12	85284

Donc d'après la matrice de confusion on voit que :

- 91 cas étaient en réalité fraude et le modèle les a prédits fraude
- 12 cas en réalité sont non-fraude et le modèle les a prédits fraude
- 56 cas étaient en réalité fraude et le modèle les a prédits non fraude
- 85284 sont des cas de non fraude et le modèle les a prédits non fraude

b) Pour l'arbre de décision :

```

> ▶ Ml
#Decision tree
tnD, fpD, fnD, tpD = confusion_matrix(ytest, y_pred).ravel()
conf_matrix = pd.DataFrame(
    {
        'Predicted Fraud': [tpD, fpD],
        'Predicted Not Fraud': [fnD, tnD]
    }, index=['Fraud', 'Not Fraud'])
conf_matrix

```

	Predicted Fraud	Predicted Not Fraud
Fraud	114	33
Not Fraud	31	85265

Donc d'après la matrice de confusion on voit que :

- 114 cas étaient en réalité fraude et le modèle les a prédits fraude
- 31 cas en réalité sont non-fraude et le modèle les a prédits fraude
- 33 cas étaient en réalité fraude et le modèle les a prédits non fraude
- 85265 sont des cas de non fraude et le modèle les a prédits non fraude

2 - Justesse (Accuracy)

La justesse, score (ou accuracy en anglais) représente la part de prédictions correctes effectuées par un modèle. La formule du score est définie comme suit :

$$Score = \frac{TP + TN}{TP + TN + FP + FN}$$

La justesse du modèle de régression logistique est :

```
ML
lr_accuracy = accuracy_score(ytest, lr_pred_test)
lr_accuracy

0.999204147794436
```

La justesse du classifieur arbre de décision est :

```
ML
accuracy = accuracy_score(ytest, y_pred)
precision = precision_score(ytest, y_pred)
accuracy

0.9992509626300574
```

Les deux modèles de classification ont obtenu des scores de précision de 99% mais il nous reste à vérifier d'autres facteurs.

3 - Précision

La précision est le nombre de vrais positifs divisé par le nombre de vrais positifs et de faux positifs. En d'autres termes, c'est le nombre de prédictions positives divisé par le nombre total de valeurs de classe positives prédites. On l'appelle aussi la valeur prédictive positive (VPP).

La relation est : $TP / (TP + FP)$

La précision peut être considérée comme une mesure de l'exactitude d'un classificateur. Une faible précision peut également indiquer un grand nombre de faux positifs.

4 – Recall (rappel)

Le rappel est le nombre de vrais positifs divisé par le nombre de vrais positifs et le nombre de faux négatifs. En d'autres termes, il s'agit du nombre de prédictions positives divisé par le nombre de valeurs de classe positives dans les données de test. Il est également appelé sensibilité ou taux de vrais positifs.

Le rappel peut être considéré comme une mesure de l'exhaustivité d'un classificateur. Un rappel faible indique de nombreux faux négatifs.

Sa relation est : $TP / (TP + FN)$

5 - F1-score

Score F1 = $2 * P * R / (P + R)$ ici P pour Précision, R pour Rappel

Le score F1 combine la précision et le rappel par rapport à une classe positive spécifique, il peut être interprété comme une moyenne pondérée de la précision et du rappel, où un score F1 atteint sa meilleure valeur à 1 et la pire à 0.

Lorsque nous observons le résultat du rapport de classification des deux classificateurs, le score f1 pour la classe 0 a obtenu 100%, mais pour la classe 1, les scores F1 sont nettement inférieurs.

a) Pour le classifieur de régression logistique :

```
ML
from sklearn.metrics import classification_report
print(classification_report(ytest, lr_pred_test))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	85296
1	0.88	0.62	0.73	147
accuracy			1.00	85443
macro avg	0.94	0.81	0.86	85443
weighted avg	1.00	1.00	1.00	85443

b) Pour le classifieur d'arbre de décision :

```
ML
#Decision Tree
from sklearn.metrics import classification_report
print(classification_report(ytest, y_pred))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	85296
1	0.79	0.78	0.78	147
accuracy			1.00	85443
macro avg	0.89	0.89	0.89	85443
weighted avg	1.00	1.00	1.00	85443

L. Amélioration du modèle à l'aide de techniques d'échantillonnage Oversampling & Undersampling

En raison du nombre d'échantillons pour les classes 0 et 1. Le nombre d'échantillons pour la classe 0 est très élevé par rapport aux échantillons de classe 1.

Donc, ce que nous devons faire, c'est gérer un ensemble de données déséquilibré. Pour cela on a appliqué la notion de sous-échantillonnage et suréchantillonnage.

Dans la technique de suréchantillonnage, les **échantillons sont répétés** et la taille de l'ensemble de données est **plus grande** que l'ensemble de données d'origine.

Dans la technique de sous-échantillonnage, les **échantillons ne sont pas répétés** et la taille de l'ensemble de données est **inférieure** à l'ensemble de données d'origine.

Application des techniques de sous-échantillonnage.

Pour les techniques de sous-échantillonnage, nous vérifions le nombre d'échantillons des deux classes et sélectionnons le plus petit nombre et prélevons des échantillons aléatoires à partir d'échantillons d'autres classes pour créer un nouvel ensemble de données. Le nouvel ensemble de données à un nombre égal d'échantillons pour les deux classes cibles.

C'est tout un processus de sous-échantillonnage, donc nous avons implémenté tout ce processus en utilisant python.

▶ M↓

```
nonfraud_indexies = df[df.Class == 0].index
fraud_indices = np.array(df[df['Class'] == 1].index)
class_val = df['Class'].value_counts()
non_fraud = class_val[0]
fraud = class_val[1]
# take random samples from non fraudulent that are equal to fraudulent samples
random_normal_indexies = np.random.choice(nonfraud_indexies, fraud,
replace=False)
random_normal_indexies = np.array(random_normal_indexies)
# concatenate both indices of fraud and non fraud
under_sample_indices = np.concatenate([fraud_indices, random_normal_indexies])
#extract all features from whole data for under sample indices only
under_sample_data = df.iloc[under_sample_indices, :]

# now we have to divide under sampling data to all features & target
x_undersample_data = under_sample_data.drop(['Class'], axis=1)
y_undersample_data = under_sample_data[['Class']]
# now split dataset to train and test datasets as before
X_train_sample, X_test_sample, y_train_sample, y_test_sample = train_test_split(
x_undersample_data, y_undersample_data, test_size=0.2, random_state=0)
d = pd.DataFrame(y_undersample_data)
d.value_counts()
```

```
Class
0      492
1      492
dtype: int64
```

M. Classification avec régression logistique et Arbre de décision après application des techniques d'échantillonnage

a) Pour le modèle de régression logistique

```
from sklearn.metrics import classification_report
print(classification_report(y_test_sample, pred_test))
```

	precision	recall	f1-score	support
0	0.95	0.98	0.97	106
1	0.98	0.95	0.96	91
accuracy			0.96	197
macro avg	0.97	0.96	0.96	197
weighted avg	0.96	0.96	0.96	197

b) Pour le modèle d'arbre de décision

```
#Decision Tree
print(classification_report(y_test_sample, y_pred_New))
```

	precision	recall	f1-score	support
0	0.85	1.00	0.92	106
1	1.00	0.80	0.89	91
accuracy			0.91	197
macro avg	0.93	0.90	0.91	197
weighted avg	0.92	0.91	0.91	197

Vous voyez, les résultats du score F1 pour les deux valeurs cibles sont équilibrés.

N. Conclusion

Dans notre projet on a utilisé 2 classifieurs différents avec un prétraitement sur l'ensemble de données. On a obtenu des Accuracy qui sont presque les mêmes mais l'ensemble des données étaient déséquilibrée pour cela on a procédé avec une technique d'équilibrage qui s'appelle sous-échantillonnage (undersampling), cela nous a donné un f1-score homogène entre les classes 0, 1 et une Accuracy du classifieur régression logistique meilleure que l'arbre de décision.

Nous pouvons améliorer les résultats du modèle en ajoutant plus d'arbres ou en appliquant des techniques de prétraitement de données supplémentaires, pas seulement les arbres de décision ou régression logistique adaptés à ce problème. Nous pouvons essayer avec d'autres algorithmes de classification d'apprentissage automatique tels que les machines à vecteurs de support (SVM), les k plus proches voisins, etc. pour vérifier comment différents algorithmes sont exécutés pour classer les activités frauduleuses.