Master 01 informatique
UE Techniques d'Apprentissage Artificiel

Prédiction de prêt
Problème de classification

Fait par :

- Chaima Nemir douaouda
- Chahinez Kebbi

Supervisé par :

- Mme, Rakia JAZIRI

Année : 2022/2023

Objectif de projet :

Notre travail port sur le concept du prêt entre particuliers se présente comme une alternative au financement bancaire classique. Les investisseurs attribuent un taux d'intérêt qui varie en fonction d'une cote de crédit, plus la cote est élevée (emprunt plus fiable et moins risqués) plus le taux d'intérêt est bas.

Pour les investisseurs, les prêts à taux d'intérêt plus élevés sont les plus intéressants car ils offrent un retour sur investissement plus élevé , mais d'un autre coté , ils présentent des risques de ne pas être remboursés du tout .

Lorsque un investisseur ou une entreprise de prêt reçoit une demande de prêt, elle doit prendre une décision d'approbation de prêt en fonction du profil du demandeur. Deux types de risques sont associés à la décision de la société :

- Si le demandeur est susceptible derembourser le prêt, le refus d'approuvement entraîne une perte d'affaires pour l'entreprise.
- Si le demandeur n'est pas susceptible de rembourser le prêt, c'est-à-dire qu'il risque de ne pas le rembourser, l'approbation du prêt peut entraîner une perte financière pour l'entreprise.

L'objectif est donc de se mettre à la place d'un data scientist d'une société de prêt et construire un modèle de prédiction basé sur la Machine Learning , qui pourrait prévoir quels prêts sont les susceptibles d'être remboursés , en particulier pour les taux d'intérêt élevés, apporterait un rendement plus important pour les investisseurs , en minimisant les risques associés , ce qui peut être utilisé pour prendre des mesures telles que le refus du prêt, la réduction du montant du prêt, augmenter le taux d'intérêt du prêt si le demandeur a été jugé comme '' demandeur à risque '' etc.

Description du dataset :

Les données fournies proviennent d'une entreprise de prêt basée en Californie USA , le dataset contient des informations sur les anciens demandeurs de prêts et indique aussi s'ils ont «défailli» ou non.

Lorsqu'une personne demande un prêt, l'entreprise peut prendre deux types dedécisions :

1- Prêt accepté: Si l'entreprise approuve le prêt, il y a 3 scénarios possibles décrits ci-dessous:

- Fully paid (entièrement remboursé): le demandeur a remboursé intégralement le prêt (principalet taux d'intérêt).
- Current (actuel) : Le demandeur est en train de payer les versements, c'est-à-dire que la durée du prêt n'est pas encore terminée. Ces candidats ne sont pas étiquetés comme « default ».
- Charged-off (imputation): le demandeur n'a pas payé les versements en temps voulu pendant une longue période, c'est-à-dire qu'il n'apas remboursé le prêt.
- On retrouve aussi le statu ‘ Late 1 et Late 2 ‘ qui représentent les demandeurs qui sont pas à jours au payement .

2- Prêt refusé : L'entreprise avait refusé le prêt (car le candidat ne répond pas à ses exigences, etc.). Comme le prêt a été refusé, il n'y a pas d'historique des transactions entre les demandeurs et l'entreprise et, par conséquent, ces données ne sont pas disponibles auprès de l'entreprise (et donc dans ce dataset).

Pour la description des colonnes du dataset(on va joindre un pdf qui contient la description de chaque featur)

Étapes de travail :

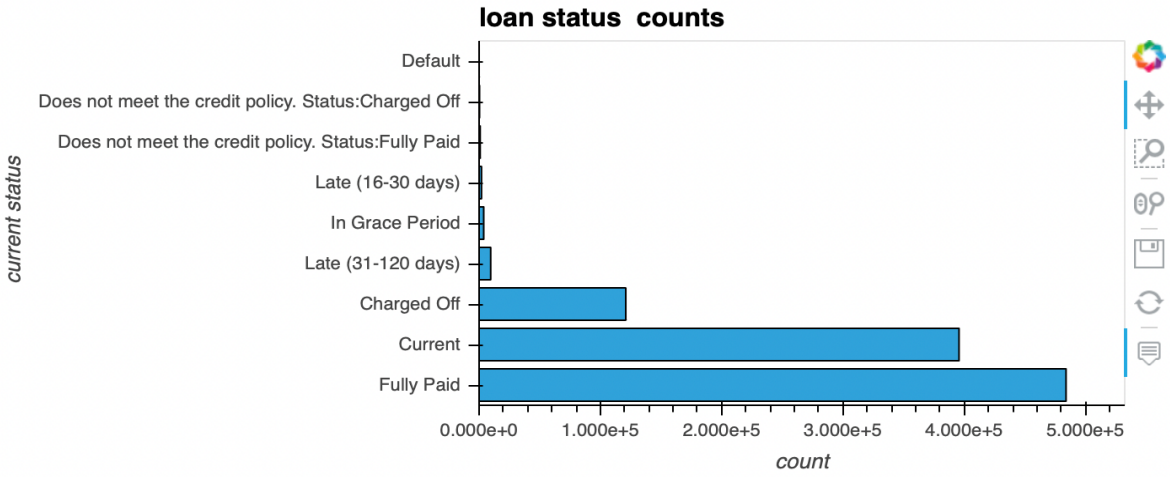
1- Chargement et exploration des données :

Pour charger les données on utilise une fonction de module Pandas de python, Comme le fichier est très grand on va pas être capable de l'analyser avec nos machines, il faudra donc réduire la taille en prenant un échantillon du dataset : (cela réduit le nombre de lignes de 2260701 à 1017315):

```
df = pd.read_csv("/Users/chaimanemir/Desktop/projet ia/accepted_2007_to_2018Q4.csv",sep=',')
df2 =df.sample(frac = 0.45, axis = 0, random_state = 0).reset_index(drop = True)
print(df2.info())
print(df2.describe())
```

On analyse aussi la proportion des différentes situations du prêt en explorant la colonne ‘target’ ‘loan_status’

```
proportion =(pd.value_counts(df2['loan_status']))/len(df2['loan_status'])*100
pl=df2['loan_status'].value_counts().hvplot.barh(title=" loan status  counts",xlabel=" current status",ylabel='count')
hvplot.show(pl)
```



2- **Preprocessing et Featurs selection :**

D’abord on commence par traiter les valeurs manquantes, pour cela on analyse le taux de valeurs manquantes dans chaque ligne et on supprime celles dont le taux des V.M dépasse 30% , On supprime également les lignes contenants des NAN:

```
nul = df2.isnull().mean().sort_values()
nul = nul[nul>0.3]
nul_col = nul.sort_values(ascending = False).index
data = df2.drop(nul_col, axis = 1)
data = data.dropna(axis = 0).reset_index(drop = True)
```

On procède ensuite à la suppression des colonnes qui n’apportent pas d’informations sur l’approbation du prêt , autrement dit , les colonnes dont la valeurs est différentes pour chaque ligne :

```
categorie = [feature for feature in data.columns if data[feature].dtype == "O"]
for value in categorie:
    print(value, " : ", data[value].nunique())
data = data.drop(['id', 'url','sub_grade','title', 'zip_code','emp_title'], axis = 1)
```

Après avoir fait ça , on se retrouve avec 87 colonnes , par intuition on analyse ces colonnes et à l’aide de la description on peut juger quelles featurs ont in impact sur la décision d’approbation de prêt, On supprime aussi les colonnes

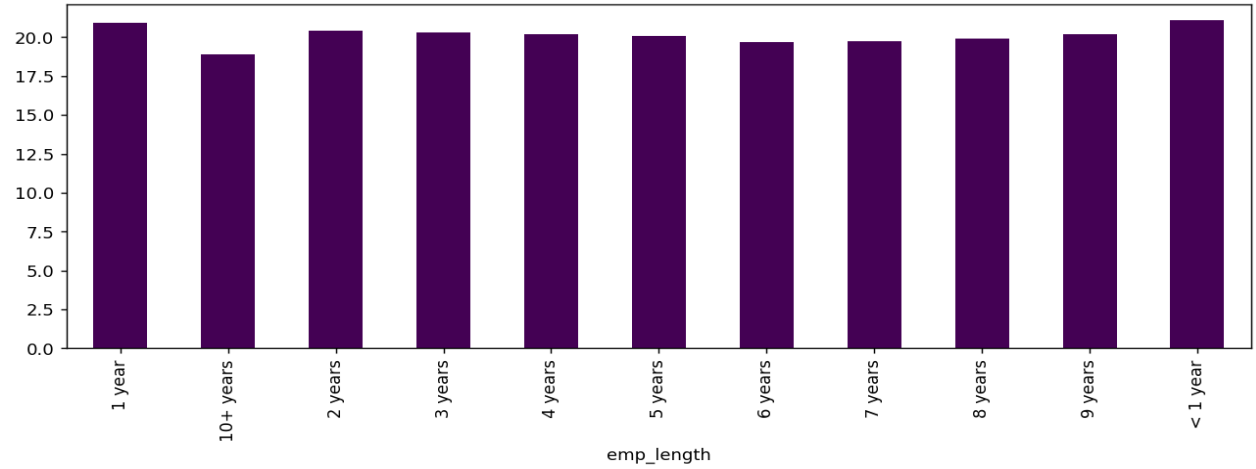
apportant des informations sur la période après que le prêt soit approuvé, ça ne va pas nous être utile pour la prise de décision :

```
keep_list = ['charged_off','funded_amnt','addr_state', 'annual_inc', 'application_type',
             'dti', 'earliest_cr_line', 'emp_length',  'fico_range_high', 'fico_range_low',
             'grade', 'home_ownership',  'initial_list_status', 'installment', 'int_rate',
             'loan_amnt', 'loan_status', 'mort_acc', 'open_acc', 'pub_rec', 'pub_rec_bankruptcies',
             'purpose', 'revol_bal', 'revol_util',  'term','total_acc', 'verification_status','last_pymnt_amnt',
             'num_actv_rev_tl', 'mo_sin_rcnt_rev_tl_op','mo_sin_old_rev_tl_op',"bc_util","bc_open_to_buy",
             "avg_cur_bal","acc_open_past_24mths" ]
drop_list = [col for col in data.columns if col not in keep_list]
data=data.drop(labels=drop_list, axis=1)
```

Pour les colonnes restantes on va analyser chaque colonnes pour vérifier si l'information apporté par cet featur a un impact sur le statut de prêt , si oui on garde la colonnes bien évidemment , sinon on supprime la colonnes :

Pour 'emp_legnth'(la durée de travail) par exemple :

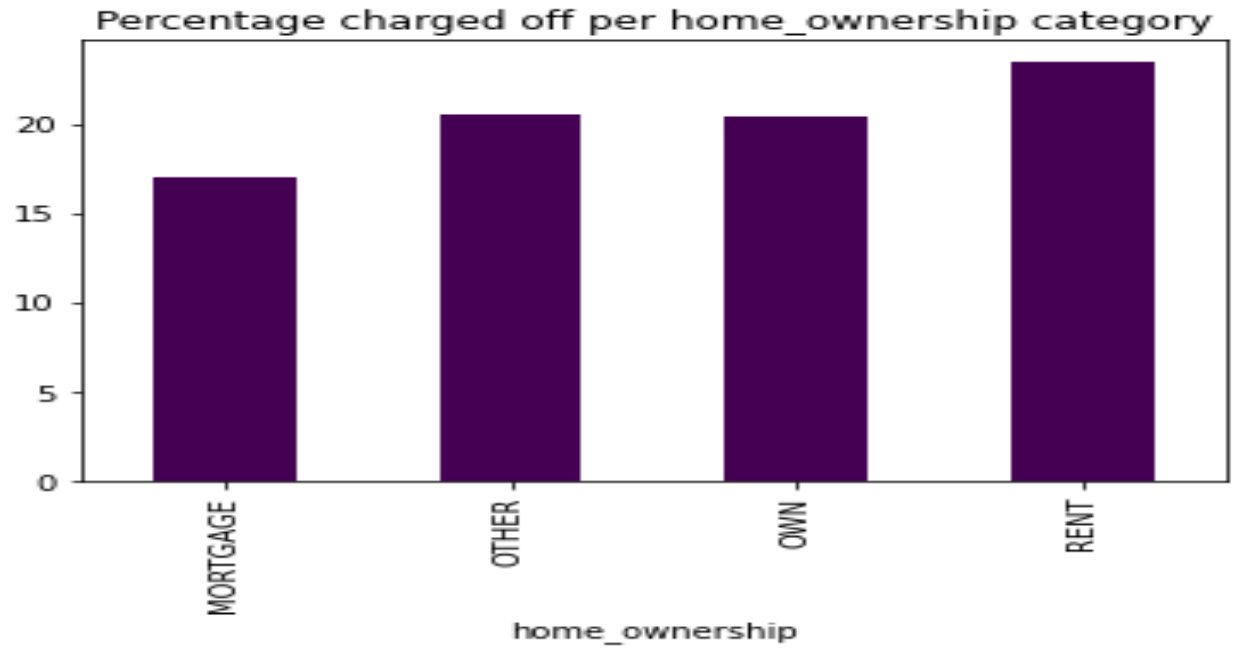
```
emp_charged_off = data[data['loan_status']=="Charged Off"].groupby("emp_length").count()['loan_status']
emp_fully_paid = data[data['loan_status']=="Fully Paid"].groupby("emp_length").count()['loan_status']
percentage_charged_off = (emp_charged_off * 100)/(emp_charged_off + emp_fully_paid)
plt.figure(figsize=(12,4), dpi=130)
percentage_charged_off.plot(kind='bar', cmap='viridis')
```



On remarque qu'il n'y a pas vraiment de différence d'impact sur le statut de prêt entre les demandeurs qui travailles depuis des années et ceux qui viennent de commencer … donc on supprime la colonne.

Pour 'home_ownership' (la possession de la propriété) :

```
data['home_ownership'] = data['home_ownership'].replace(['NONE', 'ANY'], 'OTHER')
charged_off = data[data['loan_status']=="Charged Off"].groupby("home_ownership").count()['loan_status']
fully_paid = data[data['loan_status']=="Fully Paid"].groupby("home_ownership").count()['loan_status']
percentage_charged_off = (charged_off * 100)/(charged_off + fully_paid)
percentage_charged_off.plot(kind='bar', cmap='viridis')
plt.title("Percentage charged off per home_ownership category")

dummies_home_ownership = pd.get_dummies(data['home_ownership'])
data = pd.concat([data.drop('home_ownership', axis=1), dummies_home_ownership], axis=1)
```



On remarque clairement une différence d'impact entre la home ownership et le statut du prêt donc on garde la colonne mais en la décortiquant en 4 colonnes (une colonnes pour chaque categorie : rent , own, mortgage, other) en utilisant get_dummies de pandas .
On fait la même chose pour les colonnes restantes, et au final on garde 49 colonnes.

Finalement, on passe à la colonne target que l'on cherche à prédire, on observe qu'il y'a 3 situations :
Situation A : regroupent ceux qui ont remboursé le prêt fully_paid et current (les demandeurs qui sont à jour par rapport au payement)
Situation B : ceux qui n'ont pas remboursé le prêt charged_off et default.
Situation C : regroupent les autres situation ou on peut pas mettre une hypothèse par rapport au remboursement du prêt (late1, lat2 ..) , on les supprime car on peut rien conclure par rapport à eux.
Et on recode les deux autres situations en affectant un 1 à la situation A et un 0 à la situation B.

```
a_supp_index= data[data['loan_status']=='Late (16-30 days)'].index
data = data.drop(labels=a_supp_index).reset_index(drop=True)
a_supp_index= data[data['loan_status']=='Late (31-120 days)'].index
data = data.drop(labels=a_supp_index).reset_index(drop=True)
a_supp_index= data[data['loan_status']=='In Grace Period'].index
data = data.drop(labels=a_supp_index).reset_index(drop=True)

data['target']=0
data['target'][(data['loan_status']=='Fully Paid')]= 1
data['target'][(data['loan_status']=='Current')]= 1
data=data.drop(labels=['loan_status'], axis=1)
```

3- Normalisation et découpage de données :

Avant de commencer l'entrainement du modèle il faut d'abord le couper en deux, des données pour l'entrainement du modèle et des données pour le test de validation pour cela on utilise la méthode prédéfinie de model_selection de Sklearn , une fois le découpage fait , on fait une normalisation de données en utilisant le MinMax scaler du preprocessing de Sklearn :

```
#----------------------- split  ---------------------------
X=data.drop(labels=['target'], axis=1)
Y=  data['target']
#
X_train,X_test, Y_train, Y_test= train_test_split(X,Y,test_size=0.15)

#---------------------------scalling ------------------------------
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

4- Construction de Modèle & prédiction :

On est face à un problème très répondu en Machine Learning, à savoir le problème de classification, il existe plusieurs modèles dédiés à ce genre de probléme , on a choisi d'en utiliser 4 :

- RandomForestClassifier , logisticRegression de Sklearn
- eXGBClassifier de xgboost
- ANNs un reseaux de neurone artificiel de Keras.Tensorflow

Ensuit on va comparer la performance de chaque modèle en utilisant les accuracy report et les matrices de confusions.

I. logisticRegression:

```
from sklearn.metrics import accuracy_score,plot_confusion_matrix,classification_report
from sklearn.metrics import ConfusionMatrixDisplay,confusion_matrix
#LogisticRegression
from sklearn.linear_model import LogisticRegression
lr =LogisticRegression()
lr.fit(X_train,Y_train)
preds = lr.predict(X_test)
accuracy_report_LinReg = classification_report(Y_test,preds)
print(accuracy_report_LinReg)
plot_confusion_matrix(lr,X_test,Y_test)
```

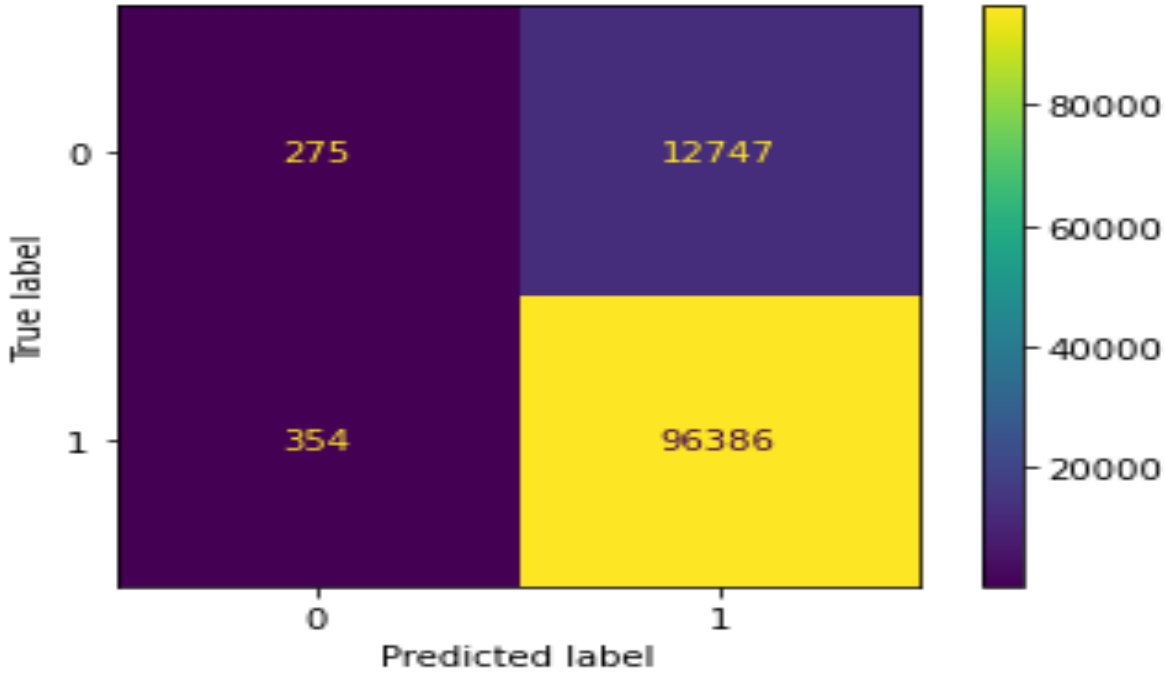
Le test de performance donne :

Le recall : appelé aussi ‘sensitivity’ ,’hit rate’, il corresponds au taux de prédictions pertinentes détectées par le modèles

La precision : appelé aussi ‘positive value prediction’, elle représente le taux de prédiction correctes parmi les prédictions faites.

F1-score : représente la moyenne harmonique de la précision et du recall

	precision	recall	f1-score	support
0	0.44	0.02	0.04	13022
1	0.88	1.00	0.94	96740
accuracy			0.88	109762
macro avg	0.66	0.51	0.49	109762
weighted avg	0.83	0.88	0.83	109762

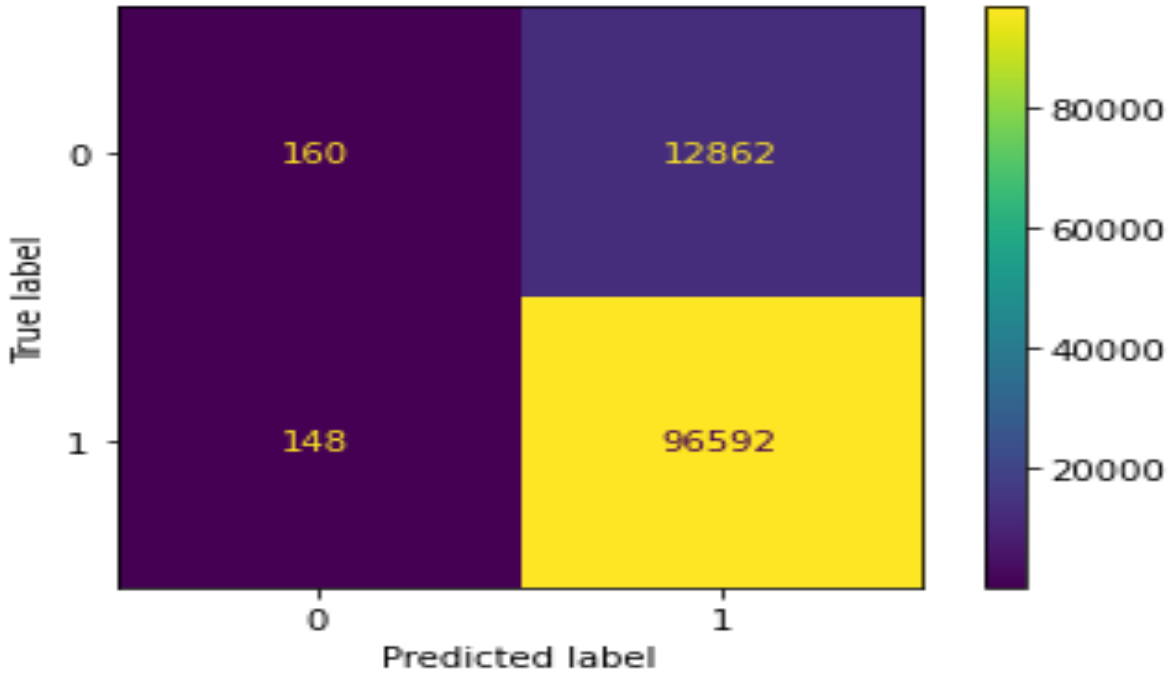


II. RandomForestClassifier :

```
#random_forest
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100)
rf.fit(X_train,Y_train)
preds = rf.predict(X_test)
accuracy_report_RandomForest = classification_report(Y_test,preds)
print(accuracy_report_RandomForest)
plot_confusion_matrix(rf,X_test,Y_test)
```

Le test de performance donne :

```
              precision    recall  f1-score   support

           0       0.52      0.01      0.02     13022
           1       0.88      1.00      0.94     96740

    accuracy                           0.88    109762
   macro avg       0.70      0.51      0.48    109762
weighted avg       0.84      0.88      0.83    109762
```

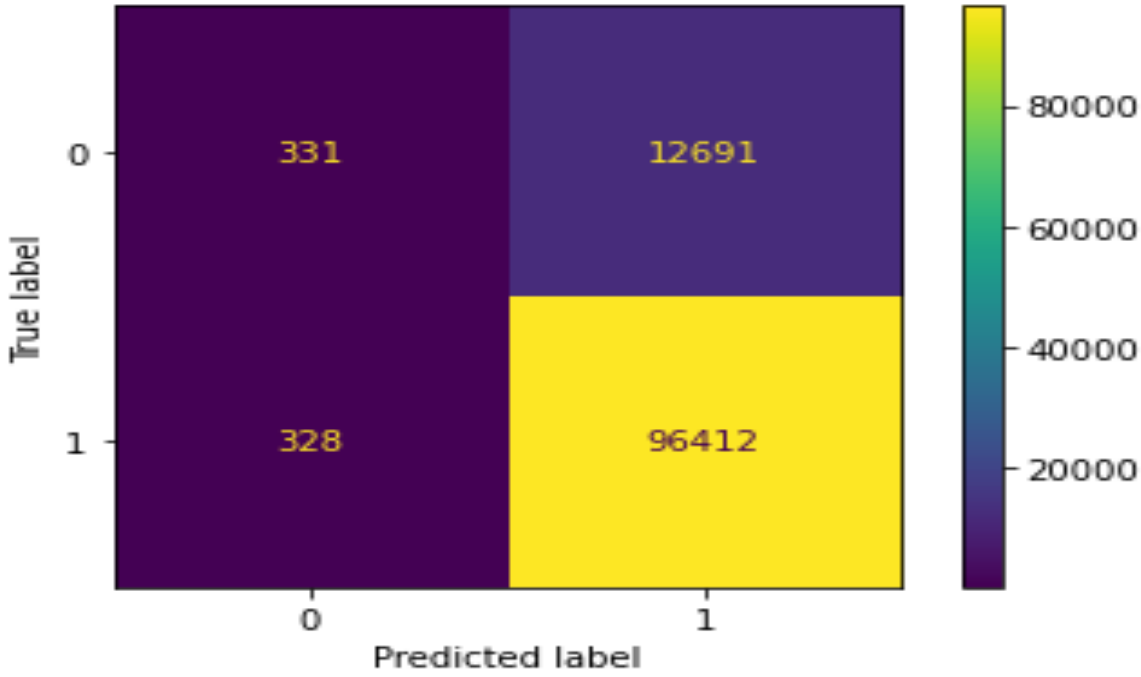


III. XGBClassifier :

```
#XGBoost
from xgboost import XGBClassifier
model = XGBClassifier()
model.fit(X_train, Y_train)
preds = model.predict(X_test)
accuracy_report_XGBoost=classification_report(Y_test,preds)
print(accuracy_report_XGBoost)
plot_confusion_matrix(model,X_test,Y_test)
```

Le test de performance donne :

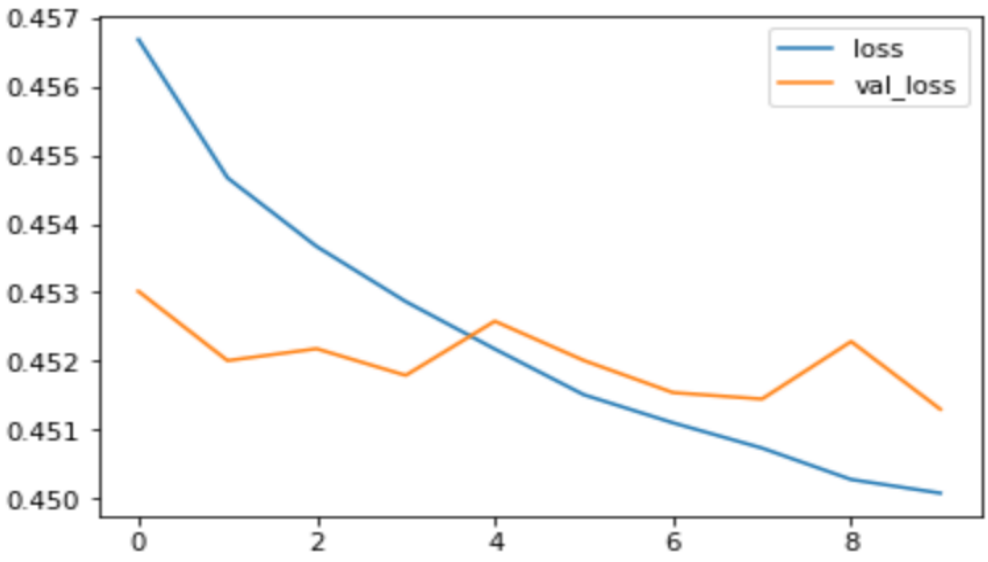
	precision	recall	f1-score	support
0	0.50	0.03	0.05	13022
1	0.88	1.00	0.94	96740
accuracy			0.88	109762
macro avg	0.69	0.51	0.49	109762
weighted avg	0.84	0.88	0.83	109762



IV. ANNs :

Pour ANNs on a dû réduire la taille encore parce que ça prenait beaucoup d temps pour s'exécuter. (aussi on a utilisé Jupyter notebook pour ce modèle et non pas Spyder)

```
#--------- Reseaux de neurones ----------------
#ANN
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense, Activation,Dropout
from tensorflow.keras.constraints import max_norm
from tensorflow import keras

model = keras.Sequential()
# input couche
model.add(Dense(119,  activation='relu'))
model.add(Dropout(0.1))

#  couches cachées
model.add(Dense(78,  activation='relu'))
model.add(Dropout(0.1))

model.add(Dense(39, activation='relu'))
model.add(Dropout(0.1))

model.add(Dense(19, activation='relu'))
model.add(Dropout(0.1))

# output couche
model.add(Dense(units=1,activation='sigmoid'))

#Compile model
model.compile(loss='binary_crossentropy', optimizer='adam')
model.fit(x=X_train, y=Y_train, epochs=10,batch_size=256,validation_data=(X_test, Y_test),)

losses = pd.DataFrame(model.history.history)
losses[['loss','val_loss']].plot()
plt.show()
```



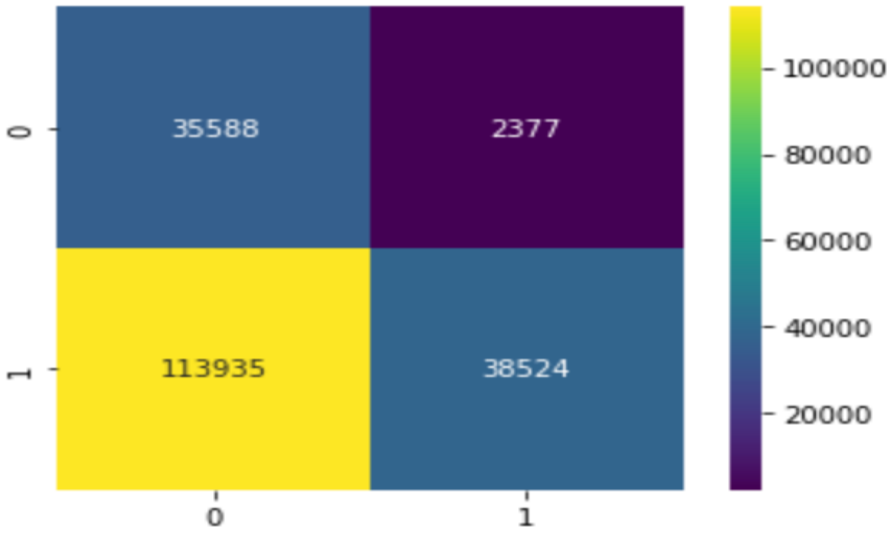
(Pour l'affichage de matrice de confusion on a utilisé Seaborn au lieu de plot_confusion_matrix qui ne marchait pas directement avec ANNs)

Avec 0.5 :

```
predictions = (model.predict(X_test) > 0.5).astype("int32")
accuracy_report_ANN=classification_report(Y_test,predictions)
print(accuracy_report_ANN)

cm = confusion_matrix(Y_test,predictions)
f = sns.heatmap(cm, annot=True, fmt='d', cmap='viridis', square=True)
```

Le test de performance donne :

	precision	recall	f1-score	support
0	0.56	0.05	0.09	37965
1	0.81	0.99	0.89	152459
accuracy			0.80	190424
macro avg	0.68	0.52	0.49	190424
weighted avg	0.76	0.80	0.73	190424

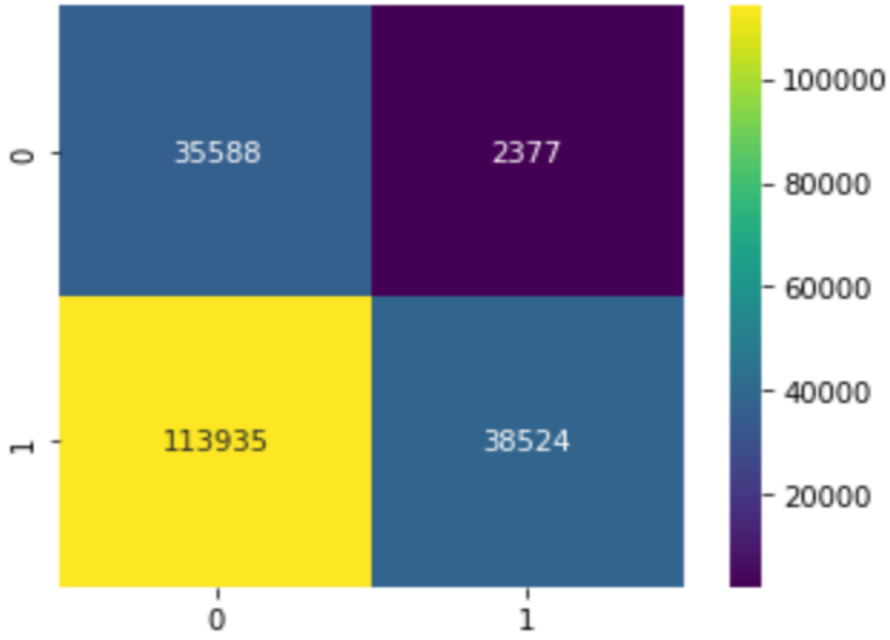


Avec 0.7 :

```
predictions = (model.predict(X_test) > 0.7).astype("int32")
accuracy_report_ANN=classification_report(Y_test,predictions)
print(accuracy_report_ANN)

cm = confusion_matrix(Y_test,predictions)
f = sns.heatmap(cm, annot=True, fmt='d', cmap='viridis', square=True)
```

Le test de performance donne :

	precision	recall	f1-score	support
0	0.40	0.39	0.39	37965
1	0.85	0.85	0.85	152459
accuracy			0.76	190424
macro avg	0.62	0.62	0.62	190424
weighted avg	0.76	0.76	0.76	190424



Conclusion :

En terme de performances les 4 modèles ont été assez performants , le RandomForestClassifier était le modèle qui a eu la meilleur accuracy et F1-score, suivis du XGBoost , logisticRegression et en fin le ANNs .

En terme du temps, le RandomForestClassifier et le ANNs étaient les modèles qui a pris plus du temps pour donner des résultat comparant ay LogisitcRegression et le XGBoost.

La baisse de performance de ANNs est peut-être dû à la réduction que l'on a fait au dataset , ou bien mal ajustement des paramètres .