Home Credit

on a établi un modèle pour prédire dans quelle mesure chaque client est capable de rembourser un prêt.

Concept

```
Dans ce problème, les données sont déséquilibrées. Nous ne pouvons
donc pas utiliser la précision comme mesure d'erreur. On va utiliser
:
   . Log loss, F1-score et AUC
   . La courbe Roc (visualiser les performances des classificateur
binaire)
   . La Matrice de confusion (Obtenir un aperçu des prévisions)
```

```
Les 7 sources de données (Datasets)
```

Application_train/test: les principales base de données pour chaque demande de crédit et chaque prêt identifié par (SK_ID_CURR). TARGET indiquant 1 si le prêt n'a pas été remboursé ou 0 si le preta été rembourses . Dans le cas de notre modèle on utilise seulement les données d'entraînement (**Application_train**)

bureau: cet ensemble de données comprends les crédits précédent auprés d'autres institutions financières (client’s previous credits).

bureau_balance: se compose de données mensuelles sur les crédits précédents, Chaque ligne correspond à un mois d'un crédit précédent et un seul depond de chaque mois de la durée du crédit .

previous_application : Les données des demandes précédentes de prêts au crédit immobilier des clients. Chaque application précédente a une ligne et est identifiée par la fonction (SK_ID_PREV).

POS_CASH_BALANCE : se compose de données mensuelles sur les points de vente précédents ou les prêts de trésorerie que les clients ont obtenus avec les crédit immobilier.

credit_card_balance: Les données mensuelles sur les anciennes cartes de crédit que les clients avaient avec Home Credit.

installments_payment:l'historique de paiement des prêts anciens chez Home Credit.

les étapes des datasets :

```
    1. Verifier combien de colonnes et de lignes
    2. La jointure des Datasets :
         Previous Application data et Application Bureau data
         POS_CASH_balance data et application_bureau_prev_data
         Installments Payments data et application_bureau_prev_dat
    a
    3. Repartition des datasets en train, test et valid
    4. Normalisée notre dataset
    5. La sélection des features
```

```
Modèles de Machine Learning
```

```
  1.LightGBM : est un cadre de gradient rapide, distribué et haute
performance basé sur un algorithme d'arbre de décision, utilisé pour
le classement, la classification et de nombreuses autres tâches
d'apprentissage automatique.
  2.Logistic Regression :
  3.Random Forest Classifier
```

Table des matières

1. Importer les librairies

Entrée [49]:
```
#Importer les librairies
import pandas as pd
import sklearn
import numpy as np
import matplotlib.pyplot as plt
import os
import warnings
import seaborn as sns
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.linear_model import SGDClassifier
import plotly.offline as py
import plotly.graph_objs as go
from plotly.offline import init_notebook_mode, iplot
from sklearn.model_selection import train_test_split
init_notebook_mode(connected=True)
import cufflinks as cf
cf.go_offline()
import pickle
import gc
import lightgbm as lgb
from lightgbm import LGBMClassifier
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
%matplotlib inline
#Installation de cufflinks
#pip install cufflinks
#pip install lightgbm
```

2. Lire le Dataset

Entrée [2]:
```
#Lire le dataset Train
application= pd.read_csv("/Users/macbook/Downloads/application_train.
print('done!!!')
print('The shape of data:',application.shape)
print('First 5 rows of data:')
application.head()
```

```
done!!!
The shape of data: (307511, 122)
First 5 rows of data:
```

Out[2]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_
0	100002	1	Cash loans	M	N	
1	100003	0	Cash loans	F	N	
2	100004	0	Revolving loans	M	Y	
3	100006	0	Cash loans	F	N	
4	100007	0	Cash loans	M	N	

5 rows × 122 columns

Entrée [3]:
```
#Calculer les valeurs manquantes avec des pourcentages
count = application.isnull().sum().sort_values(ascending=False)
percentage = ((application.isnull().sum()/len(application)*100)).sort
missing_application = pd.concat([count, percentage], axis=1, keys=['C
print('Count and percentage of missing values for top 20 columns:')
missing_application.head(20)
```

Count and percentage of missing values for top 20 columns:

Out[3]:

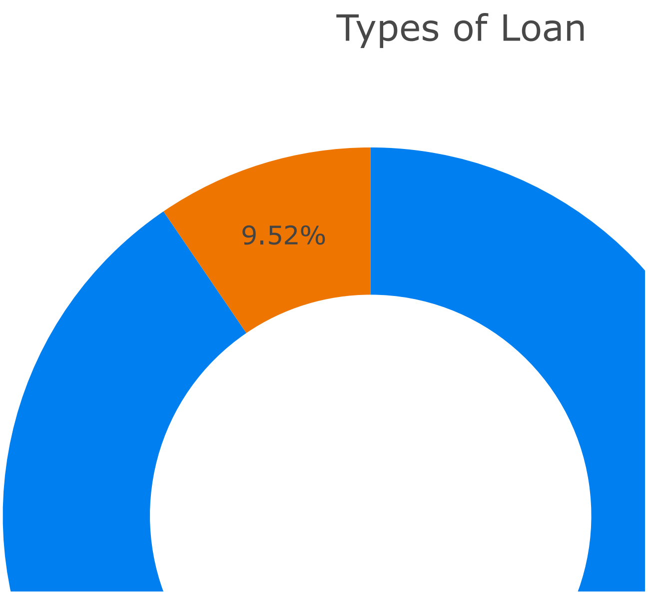
	Count	Percentage
COMMONAREA_MEDI	214865	69.872297
COMMONAREA_AVG	214865	69.872297
COMMONAREA_MODE	214865	69.872297
NONLIVINGAPARTMENTS_MODE	213514	69.432963
NONLIVINGAPARTMENTS_MEDI	213514	69.432963
NONLIVINGAPARTMENTS_AVG	213514	69.432963
FONDKAPREMONT_MODE	210295	68.386172
LIVINGAPARTMENTS_MEDI	210199	68.354953
LIVINGAPARTMENTS_MODE	210199	68.354953
LIVINGAPARTMENTS_AVG	210199	68.354953
FLOORSMIN_MEDI	208642	67.848630
FLOORSMIN_MODE	208642	67.848630
FLOORSMIN_AVG	208642	67.848630
YEARS_BUILD_MEDI	204488	66.497784
YEARS_BUILD_AVG	204488	66.497784
YEARS_BUILD_MODE	204488	66.497784
OWN_CAR_AGE	202929	65.990810
LANDAREA_MODE	182590	59.376738
LANDAREA_AVG	182590	59.376738
LANDAREA_MEDI	182590	59.376738

Entrée [4]:
```
columns_without_id = [col for col in application.columns if col!='SK_
#Vérifier les duplicates dans notre base
application[application.duplicated(subset = columns_without_id, keep=
print('The no of duplicates in the data:',application[application.dup
      .shape[0])
```

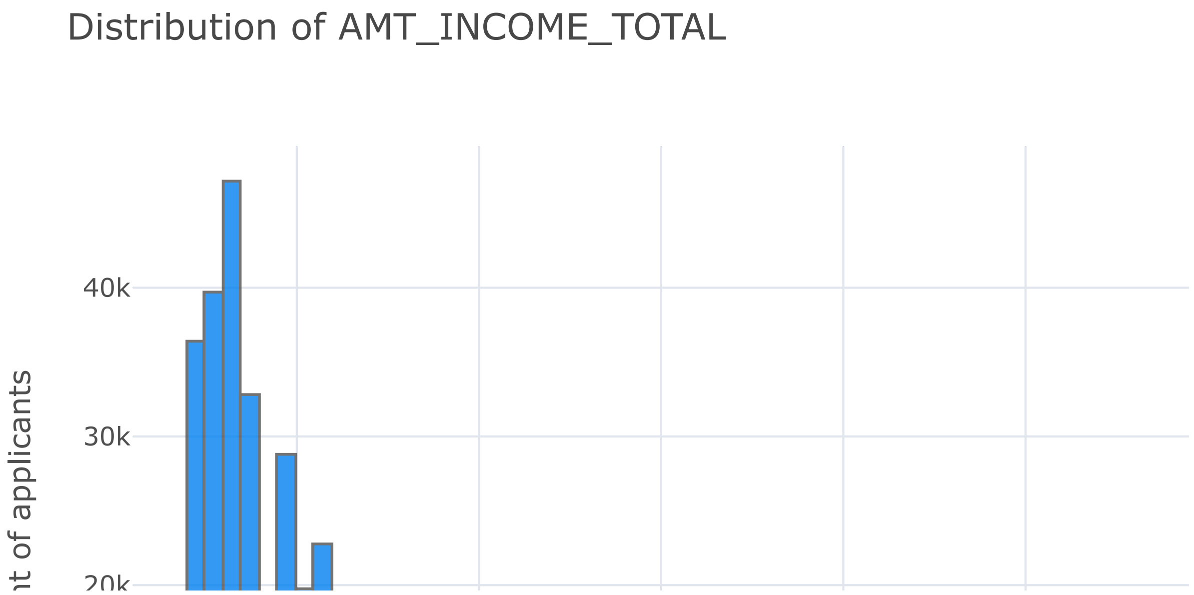
The no of duplicates in the data: 0

3. Visualization (EDA)

Entrée [5]:
```
#Le type de credit
cf.set_config_file(theme='polar')
contract_val = application['NAME_CONTRACT_TYPE'].value_counts()
contract_df = pd.DataFrame({'labels': contract_val.index,
                   'values': contract_val.values
                  })
contract_df.iplot(kind='pie',labels='labels',values='values', title='
```



Entrée [6]:
```
#La distribution des revenues
application[application['AMT_INCOME_TOTAL'] < 2000000]['AMT_INCOME_TOT
    xTitle = 'Total Income', yTitle ='Count of applicants',
             title='Distribution of AMT_INCOME_TOTAL')
```



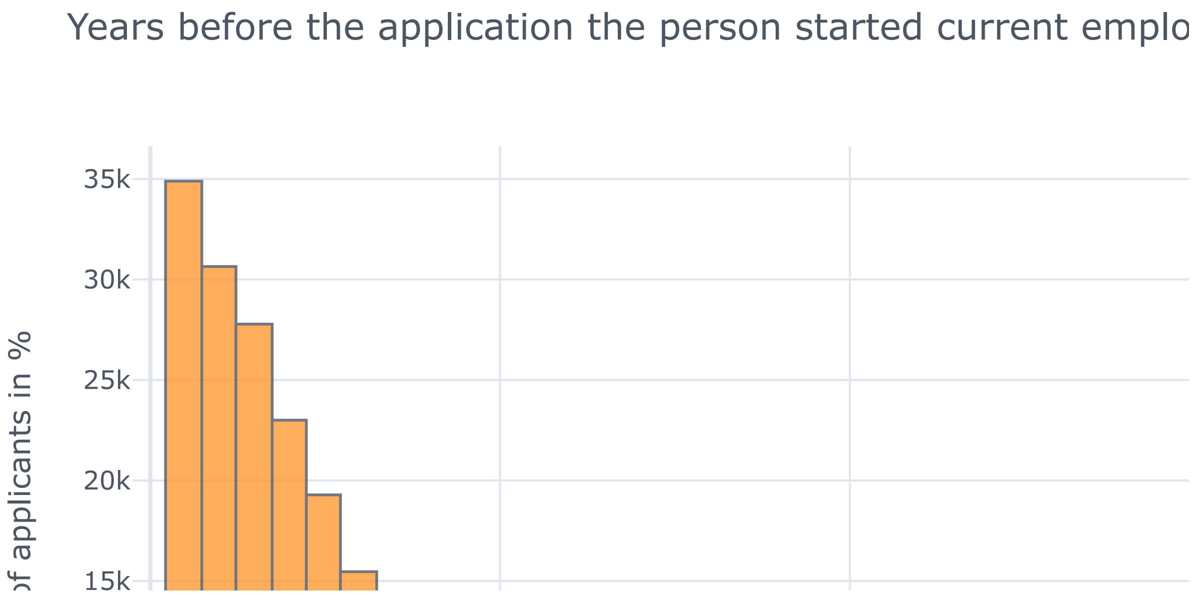
Entrée [7]:
```
#Calculer le nombre des erreurs pour Target et Days Employed
error = application[application['DAYS_EMPLOYED'] == 365243]
print('The no of errors are :', len(error))
(error['TARGET'].value_counts()/len(error))*100
```

```
The no of errors are : 55374
```

Out[7]:
```
0    94.600354
1     5.399646
Name: TARGET, dtype: float64
```

Entrée [8]:
```
# Crée une colonne
application['DAYS_EMPLOYED_ERROR'] = application["DAYS_EMPLOYED"] ==
# Replace the error values with nan
application['DAYS_EMPLOYED'].replace({365243: np.nan}, inplace = True
```

Entrée [9]:
```
cf.set_config_file(theme='pearl')
(application['DAYS_EMPLOYED']/(-365)).iplot(kind='histogram', xTitle
             yTitle='Count of applicants in %',
             title='Years before the application the person started c
```



Entrée [10]:
```
application[application['DAYS_EMPLOYED']>(-365*2)]['TARGET'].value_c
```

Out[10]:
```
0    0.887924
1    0.112076
Name: TARGET, dtype: float64
```

Preparation de notre data :

Entrée [11]:

```
 Représentation de Total income quand c'est supérieur a Credit
pplication['INCOME_GT_CREDIT_FLAG'] = application['AMT_INCOME_TOTAL']
 Colonne de pourcentage de Credit Income Percent
pplication['CREDIT_INCOME_PERCENT'] = application['AMT_CREDIT'] / app
 Column to represent Annuity Income percent
pplication['ANNUITY_INCOME_PERCENT'] = application['AMT_ANNUITY'] / a
 colonne qui représente les Credit Term
pplication['CREDIT_TERM'] = application['AMT_CREDIT'] / application['
 Colonne qui représente les  Days Employed pourcentage
pplication['DAYS_EMPLOYED_PERCENT'] = application['DAYS_EMPLOYED'] /
 Shape of Application data
rint('The shape of application data:',application.shape)
```

```
The shape of application data: (307511, 128)
```

Entrée [12]:

```
print('Reading the data....', end='')
bureau = pd.read_csv("/Users/macbook/Downloads/bureau.csv")
print('done!!!')
print('The shape of data:',bureau.shape)
print('First 5 rows of data:')
bureau.head()
```

```
Reading the data....done!!!
The shape of data: (1716428, 17)
First 5 rows of data:
```

Out[12]:

	SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	CREDIT_CURRENCY	DAYS_CREDIT	CRED
0	215354	5714462	Closed	currency 1	-497	
1	215354	5714463	Active	currency 1	-208	
2	215354	5714464	Active	currency 1	-203	
3	215354	5714465	Active	currency 1	-203	
4	215354	5714466	Active	currency 1	-629	

Entrée [13]:
```
# Combiner les variables numériques (features)
grp = bureau.drop(['SK_ID_BUREAU'], axis = 1).groupby(by=['SK_ID_CURR
grp.columns = ['BUREAU_'+column if column !='SK_ID_CURR' else column
application_bureau = application.merge(grp, on='SK_ID_CURR', how='lef
application_bureau.update(application_bureau[grp.columns].fillna(0))

# Combiner les variable catégoriques
bureau_categorical = pd.get_dummies(bureau.select_dtypes('object'))
bureau_categorical['SK_ID_CURR'] = bureau['SK_ID_CURR']
grp = bureau_categorical.groupby(by = ['SK_ID_CURR']).mean().reset_in
grp.columns = ['BUREAU_'+column if column !='SK_ID_CURR' else column
application_bureau = application_bureau.merge(grp, on='SK_ID_CURR', h
application_bureau.update(application_bureau[grp.columns].fillna(0))

# Shape de application et bureau data combiner
print('The shape application and bureau data combined:',application_b
```

```
The shape application and bureau data combined: (307511, 163)
```

Entrée [14]:
```
# Numéro des anciens crédits par client
grp = bureau.groupby(by = ['SK_ID_CURR'])['SK_ID_BUREAU'].count().re
application_bureau = application_bureau.merge(grp, on='SK_ID_CURR',
application_bureau['BUREAU_LOAN_COUNT'] = application_bureau['BUREAU
# Numéro de type des anciens crédits par client
grp = bureau[['SK_ID_CURR', 'CREDIT_TYPE']].groupby(by = ['SK_ID_CUR
application_bureau = application_bureau.merge(grp, on='SK_ID_CURR',
application_bureau['BUREAU_LOAN_TYPES'] = application_bureau['BUREAU
# Ratio dette/ crédit
bureau['AMT_CREDIT_SUM'] = bureau['AMT_CREDIT_SUM'].fillna(0)
bureau['AMT_CREDIT_SUM_DEBT'] = bureau['AMT_CREDIT_SUM_DEBT'].fillna
grp1 = bureau[['SK_ID_CURR','AMT_CREDIT_SUM']].groupby(by=['SK_ID_CU
grp2 = bureau[['SK_ID_CURR','AMT_CREDIT_SUM_DEBT']].groupby(by=['SK_
grp1['DEBT_CREDIT_RATIO'] = grp2['TOTAL_CREDIT_SUM_DEBT']/grp1['TOTA
del grp1['TOTAL_CREDIT_SUM']
application_bureau = application_bureau.merge(grp1, on='SK_ID_CURR',
application_bureau['DEBT_CREDIT_RATIO'] = application_bureau['DEBT_C
application_bureau['DEBT_CREDIT_RATIO'] = application_bureau.replace
application_bureau['DEBT_CREDIT_RATIO'] = pd.to_numeric(application_
# Le ratio d'endettement
bureau['AMT_CREDIT_SUM_OVERDUE'] = bureau['AMT_CREDIT_SUM_OVERDUE'].
bureau['AMT_CREDIT_SUM_DEBT'] = bureau['AMT_CREDIT_SUM_DEBT'].fillna
grp1 = bureau[['SK_ID_CURR','AMT_CREDIT_SUM_OVERDUE']].groupby(by=['
grp2 = bureau[['SK_ID_CURR','AMT_CREDIT_SUM_DEBT']].groupby(by=['SK_
grp1['OVERDUE_DEBT_RATIO'] = grp1['TOTAL_CUSTOMER_OVERDUE']/grp2['TC
del grp1['TOTAL_CUSTOMER_OVERDUE']
application_bureau = application_bureau.merge(grp1, on='SK_ID_CURR',
application_bureau['OVERDUE_DEBT_RATIO'] = application_bureau['OVERD
application_bureau['OVERDUE_DEBT_RATIO'] = application_bureau.replac
application_bureau['OVERDUE_DEBT_RATIO'] = pd.to_numeric(application
```

Entrée [15]:

```
print('Reading the data....', end='')
previous_applicaton = pd.read_csv("/Users/macbook/Downloads/previous
print('The shape of data:',previous_applicaton.shape)
print('First 5 rows of data:')
previous_applicaton.head()
```

```
Reading the data....The shape of data: (1670214, 37)
First 5 rows of data:
```

Out[15]:

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	A
0	2030495	271877	Consumer loans	1730.430	17145.0	
1	2802425	108129	Cash loans	25188.615	607500.0	
2	2523466	122040	Cash loans	15060.735	112500.0	
3	2819243	176158	Cash loans	47041.335	450000.0	
4	1784265	202054	Cash loans	31924.395	337500.0	

5 rows × 37 columns

Entrée [16]:

```
vious applications per custome
applicaton[['SK_ID_CURR','SK_ID_PREV']].groupby(by=['SK_ID_CURR'])['
eau_prev = application_bureau.merge(grp, on =['SK_ID_CURR'], how = '
eau_prev['PREV_APP_COUNT'] = application_bureau_prev['PREV_APP_COUNT
variables numériques (features)
applicaton.drop('SK_ID_PREV', axis =1).groupby(by=['SK_ID_CURR']).mea
['PREV_'+column if column != 'SK_ID_CURR' else column for column in g
rev_columns
eau_prev = application_bureau_prev.merge(grp, on =['SK_ID_CURR'], hov
eau_prev.update(application_bureau_prev[grp.columns].fillna(0))
variable catégoriques
l = pd.get_dummies(previous_applicaton.select_dtypes('object'))
l['SK_ID_CURR'] = previous_applicaton['SK_ID_CURR']
l.head()
gorical.groupby('SK_ID_CURR').mean().reset_index()
'PREV_'+column if column != 'SK_ID_CURR' else column for column in gı
eau_prev = application_bureau_prev.merge(grp, on=['SK_ID_CURR'], how=
eau_prev.update(application_bureau_prev[grp.columns].fillna(0))
```

Entrée [17]:

```
print('Reading the data....', end='')
pos_cash = pd.read_csv("/Users/macbook/Downloads/POS_CASH_balance.cs
print('The shape of data:',pos_cash.shape)
print('First 5 rows of data:')
pos_cash.head()
```

```
Reading the data....The shape of data: (10001358, 8)
First 5 rows of data:
```

Out[17]:

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	CNT_INSTALMENT	CNT_INSTALMENT_FUT
0	1803195	182943	-31	48.0	
1	1715348	367990	-33	36.0	
2	1784872	397406	-32	12.0	
3	1903291	269225	-35	48.0	
4	2341044	334279	-35	36.0	

Entrée [18]:

```
)iner les variable numériques (features)
 pos_cash.drop('SK_ID_PREV', axis =1).groupby(by=['SK_ID_CURR']).mean
olumns = ['POS_'+column if column != 'SK_ID_CURR' else column for co
lumns = prev_columns
ation_bureau_prev = application_bureau_prev.merge(grp, on =['SK_ID_C
ation_bureau_prev.update(application_bureau_prev[grp.columns].fillna
)iner les variable catégoriques
sh_categorical = pd.get_dummies(pos_cash.select_dtypes('object'))
sh_categorical['SK_ID_CURR'] = pos_cash['SK_ID_CURR']
 pos_cash_categorical.groupby('SK_ID_CURR').mean().reset_index()
lumns = ['POS_'+column if column != 'SK_ID_CURR' else column for col
ation_bureau_prev = application_bureau_prev.merge(grp, on=['SK_ID_CU
ation_bureau_prev.update(application_bureau_prev[grp.columns].fillna
```

Entrée [19]:
```
print('Reading the data....', end='')
insta_payments = pd.read_csv("/Users/macbook/Downloads/installments_
print('The shape of data:',insta_payments.shape)
print('First 5 rows of data:')
insta_payments.head()
```

```
Reading the data....The shape of data: (13605401, 8)
First 5 rows of data:
```

Out[19]:

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	DA
0	1054186	161674	1.0	6	
1	1330831	151639	0.0	34	
2	2085231	193053	2.0	1	
3	2452527	199697	1.0	3	
4	2714724	167756	1.0	2	

Entrée [20]:
```
Combiner les variables numériques et pas catégoriques
rp = insta_payments.drop('SK_ID_PREV', axis =1).groupby(by=['SK_ID_CU
rev_columns = ['INSTA_'+column if column != 'SK_ID_CURR' else column
rp.columns = prev_columns
pplication_bureau_prev = application_bureau_prev.merge(grp, on =['SK_
pplication_bureau_prev.update(application_bureau_prev[grp.columns].fi
```

Entrée [21]:
```
print('Reading the data....', end='')
credit_card = pd.read_csv("/Users/macbook/Downloads/credit_card_bala
print('The shape of data:',credit_card.shape)
print('First 5 rows of data:')
credit_card.head()
```

```
Reading the data....The shape of data: (3840312, 23)
First 5 rows of data:
```

Out[21]:

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_ACTU
0	2562384	378907	-6	56.970	135
1	2582071	363914	-1	63975.555	45
2	1740877	371185	-7	31815.225	450
3	1389973	337855	-4	236572.110	225
4	1891521	126868	-1	453919.455	450

5 rows × 23 columns

Entrée [22]:
```
# Combiner les variables numériques (features)
grp = credit_card.drop('SK_ID_PREV', axis =1).groupby(by=['SK_ID_CURI
prev_columns = ['CREDIT_'+column if column != 'SK_ID_CURR' else colur
grp.columns = prev_columns
application_bureau_prev = application_bureau_prev.merge(grp, on =['SI
application_bureau_prev.update(application_bureau_prev[grp.columns].:
# Combiner les variables catégoriques
credit_categorical = pd.get_dummies(credit_card.select_dtypes('object
credit_categorical['SK_ID_CURR'] = credit_card['SK_ID_CURR']
grp = credit_categorical.groupby('SK_ID_CURR').mean().reset_index()
grp.columns = ['CREDIT_'+column if column != 'SK_ID_CURR' else columi
application_bureau_prev = application_bureau_prev.merge(grp, on=['SK_
application_bureau_prev.update(application_bureau_prev[grp.columns].:
```

Entrée [23]:
```
 = application_bureau_prev.pop('TARGET').values
_train, X_temp, y_train, y_temp = train_test_split(application_bureau
_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, strati
rint('Shape of X_train:',X_train.shape)
rint('Shape of X_val:',X_val.shape)
rint('Shape of X_test:',X_test.shape)
```

```
Shape of X_train: (215257, 375)
Shape of X_val: (46127, 375)
Shape of X_test: (46127, 375)
```

Entrée [24]:
```
# Séparation des colonnes numériques et catégoriques
types = np.array([dt for dt in X_train.dtypes])
all_columns = X_train.columns.values
is_num = types != 'object'
num_cols = all_columns[is_num]
cat_cols = all_columns[~is_num]
```

Entrée [25]:
```
# Caractérisation des données numériques
imputer_num = SimpleImputer(strategy='median')
X_train_num = imputer_num.fit_transform(X_train[num_cols])
X_val_num = imputer_num.transform(X_val[num_cols])
X_test_num = imputer_num.transform(X_test[num_cols])
scaler_num = StandardScaler()
X_train_num1 = scaler_num.fit_transform(X_train_num)
X_val_num1 = scaler_num.transform(X_val_num)
X_test_num1 = scaler_num.transform(X_test_num)
X_train_num_final = pd.DataFrame(X_train_num1, columns=num_cols)
X_val_num_final = pd.DataFrame(X_val_num1, columns=num_cols)
X_test_num_final = pd.DataFrame(X_test_num1, columns=num_cols)

# Caractérisation des données catégoriques
imputer_cat = SimpleImputer(strategy='constant', fill_value='MISSING
X_train_cat = imputer_cat.fit_transform(X_train[cat_cols])
X_val_cat = imputer_cat.transform(X_val[cat_cols])
X_test_cat = imputer_cat.transform(X_test[cat_cols])
X_train_cat1= pd.DataFrame(X_train_cat, columns=cat_cols)
X_val_cat1= pd.DataFrame(X_val_cat, columns=cat_cols)
X_test_cat1= pd.DataFrame(X_test_cat, columns=cat_cols)
ohe = OneHotEncoder(sparse=False,handle_unknown='ignore')
X_train_cat2 = ohe.fit_transform(X_train_cat1)
X_val_cat2 = ohe.transform(X_val_cat1)
X_test_cat2 = ohe.transform(X_test_cat1)
cat_cols_ohe = list(ohe.get_feature_names(input_features=cat_cols))
X_train_cat_final = pd.DataFrame(X_train_cat2, columns = cat_cols_oh
X_val_cat_final = pd.DataFrame(X_val_cat2, columns = cat_cols_ohe)
X_test_cat_final = pd.DataFrame(X_test_cat2, columns = cat_cols_ohe)
# Données finales
X_train_final = pd.concat([X_train_num_final,X_train_cat_final], axi
X_val_final = pd.concat([X_val_num_final,X_val_cat_final], axis = 1)
X_test_final = pd.concat([X_test_num_final,X_test_cat_final], axis =
print(X_train_final.shape)
print(X_val_final.shape)
print(X_test_final.shape)
```

```
(215257, 505)
(46127, 505)
(46127, 505)
```

LGBMClassifier

Entrée [26]:
```
model_sk = lgb.LGBMClassifier(boosting_type='gbdt', max_depth=7, lea
                     class_weight='balanced', subsample=0.9, colsample_b
train_features, valid_features, train_y, valid_y = train_test_split(
model_sk.fit(train_features, train_y, early_stopping_rounds=100, eva
```

```
Training until validation scores don't improve for 100 rounds
[200]	valid_0's auc: 0.75423	valid_0's binary_logloss: 0.592408
[400]	valid_0's auc: 0.768815	valid_0's binary_logloss: 0.566125
[600]	valid_0's auc: 0.774772	valid_0's binary_logloss: 0.551609
[800]	valid_0's auc: 0.777189	valid_0's binary_logloss: 0.541956
[1000]	valid_0's auc: 0.778678	valid_0's binary_logloss: 0.534552
[1200]	valid_0's auc: 0.77957	valid_0's binary_logloss: 0.52803
[1400]	valid_0's auc: 0.779734	valid_0's binary_logloss: 0.522452
Early stopping, best iteration is:
[1332]	valid_0's auc: 0.779798	valid_0's binary_logloss: 0.524251
```

Out[26]:
```
LGBMClassifier(boosting_type='gbdt', class_weight='balanced',
               colsample_bytree=0.8, importance_type='split',
               learning_rate=0.01, max_depth=7, min_child_samples=20
,
               min_child_weight=0.001, min_split_gain=0.0, n_estimat
ors=2000,
               n_jobs=-1, num_leaves=31, objective=None, random_stat
e=None,
               reg_alpha=0.0, reg_lambda=0.0, silent=True, subsample
=0.9,
               subsample_for_bin=200000, subsample_freq=0)
```

Entrée [32]:
```
eature_imp = pd.DataFrame(sorted(zip(model_sk.feature_importances_, X
eatures_df = feature_imp.sort_values(by="Value", ascending=False)
elected_features = list(features_df[features_df['Value']>=50]['Featur
 Enregistrement des fonctionnalités sélectionnées dans un fichier pic
ith open('select_features.txt','wb') as fp:
   pickle.dump(selected_features, fp)
rint('The no. of features selected:',len(selected_features))
```

```
ERROR! Session/line number was not unique in database. History loggi
ng moved to new session 1392
The no. of features selected: 179
```

Regression Logistique

Entrée [33]:
```
def plot_confusion_matrix(test_y, predicted_y):
    # Confusion matrix
    C = confusion_matrix(test_y, predicted_y)

    # Recall matrix
    A = (((C.T)/(C.sum(axis=1))).T)

    # Precision matrix
    B = (C/C.sum(axis=0))

    plt.figure(figsize=(20,4))

    labels = ['Re-paid(0)','Not Re-paid(1)']
    cmap=sns.light_palette("purple")
    plt.subplot(1,3,1)
    sns.heatmap(C, annot=True, cmap=cmap,fmt="d", xticklabels = labe
    plt.xlabel('Predicted Class')
    plt.ylabel('Orignal Class')
    plt.title('Confusion matrix')

    plt.subplot(1,3,2)
    sns.heatmap(A, annot=True, cmap=cmap, xticklabels = labels, ytic
    plt.xlabel('Predicted Class')
    plt.ylabel('Orignal Class')
    plt.title('Recall matrix')

    plt.subplot(1,3,3)
    sns.heatmap(B, annot=True, cmap=cmap, xticklabels = labels, ytic
    plt.xlabel('Predicted Class')
    plt.ylabel('Orignal Class')
    plt.title('Precision matrix')

    plt.show()
def cv_plot(alpha, cv_auc):

    fig, ax = plt.subplots()
    ax.plot(np.log10(alpha), cv_auc,c='g')
    for i, txt in enumerate(np.round(cv_auc,3)):
        ax.annotate((alpha[i],str(txt)), (np.log10(alpha[i]),cv_auc[
    plt.grid()
    plt.xticks(np.log10(alpha))
    plt.title("Cross Validation Error for each alpha")
    plt.xlabel("Alpha i's")
    plt.ylabel("Error measure")
    plt.show()
```

Entrée [42]:
```
alpha = np.logspace(-4,4,9)
cv_auc_score = []
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l1',class_weight = 'balanc
```

Entrée [35]:
```
clf.fit(X_train_final[selected_features], y_train)
```

Out[35]:
```
SGDClassifier(alpha=10000.0, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_inter
cept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='log', ma
x_iter=1000,
              n_iter_no_change=5, n_jobs=None, penalty='l1', power_t
=0.5,
              random_state=28, shuffle=True, tol=0.001, validation_f
raction=0.1,
              verbose=0, warm_start=False)
```

Entrée [36]:
```
sig_clf = CalibratedClassifierCV(clf, method='sigmoid')
sig_clf.fit(X_train_final[selected_features], y_train)
```

Out[36]:
```
CalibratedClassifierCV(base_estimator=SGDClassifier(alpha=10000.0,
                                                    average=False,
                                                    class_weight='ba
lanced',
                                                    early_stopping=F
alse,
                                                    epsilon=0.1, eta
0=0.0,
                                                    fit_intercept=Tr
ue,
                                                    l1_ratio=0.15,
                                                    learning_rate='o
ptimal',
                                                    loss='log', max_
iter=1000,
                                                    n_iter_no_change
=5,
                                                    n_jobs=None, pen
alty='l1',
                                                    power_t=0.5,
                                                    random_state=28,
                                                    shuffle=True, to
l=0.001,
                                                    validation_fract
ion=0.1,
                                                    verbose=0,
                                                    warm_start=False
),
                       cv=None, method='sigmoid')
```

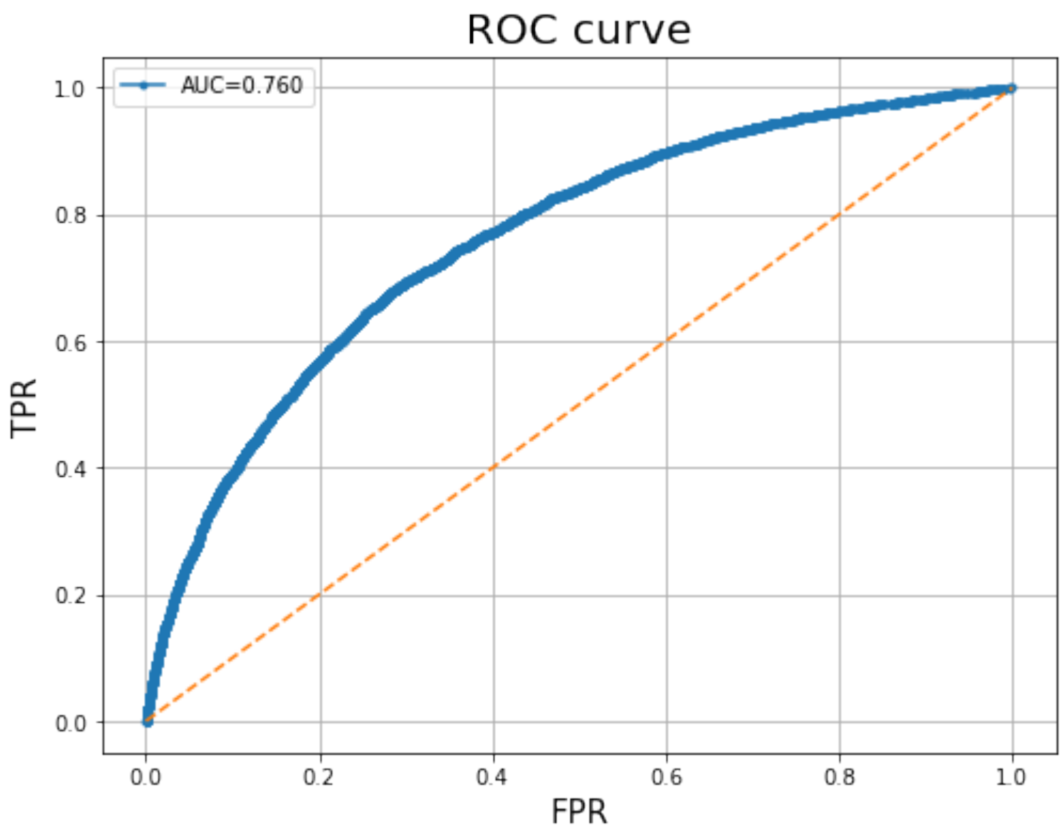
Entrée [37]:
```
y_pred_prob = []
sig_clf.predict_proba(X_val_final[selected_features])[:,1]
```

Out[37]:
```
array([0.08072677, 0.08072677, 0.08072677, ..., 0.08072677, 0.080726
77,
       0.08072677])
```

Entrée [40]:
```
best_alpha = alpha[np.argmax(cv_auc_score)]
logreg = SGDClassifier(alpha = best_alpha, class_weight = 'balanced'
logreg.fit(X_train_final[selected_features], y_train)
logreg_sig_clf = CalibratedClassifierCV(logreg, method='sigmoid')
logreg_sig_clf.fit(X_train_final[selected_features], y_train)
y_pred_prob = logreg_sig_clf.predict_proba(X_train_final[selected_fe
print('For best alpha {0}, The Train AUC score is {1}'.format(best_a
y_pred_prob = logreg_sig_clf.predict_proba(X_val_final[selected_feat
print('For best alpha {0}, The Cross validated AUC score is {1}'.for
y_pred_prob = logreg_sig_clf.predict_proba(X_test_final[selected_fea
print('For best alpha {0}, The Test AUC score is {1}'.format(best_al
y_pred = logreg.predict(X_test_final[selected_features])
print('The test AUC score is :', roc_auc_score(y_test,y_pred_prob))
print('The percentage of misclassified points {:05.2f}% :'.format((1
plot_confusion_matrix(y_test, y_pred)
```

```
For best alpha 0.0001, The Train AUC score is 0.7650162581393501
For best alpha 0.0001, The Cross validated AUC score is 0.7569473024
589161
For best alpha 0.0001, The Test AUC score is 0.7604168247220617
The test AUC score is : 0.7604168247220617
The percentage of misclassified points 32.31% :
```

Entrée [41]:
```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
auc = roc_auc_score(y_test,y_pred_prob)
plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, marker='.')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.title('ROC curve', fontsize = 20)
plt.xlabel('FPR', fontsize=15)
plt.ylabel('TPR', fontsize=15)
plt.grid()
plt.legend(["AUC=%.3f"%auc])
plt.show()
```



RandomForestClassifier

Entrée [ ]:
```
lpha = [200,500,1000,2000]
ax_depth = [7, 10]
v_auc_score = []
or i in alpha:
    for j in max_depth:
        clf = RandomForestClassifier(n_estimators=i, criterion='gini',
                                     random_state=42, n_jobs=-1)
        clf.fit(X_train_final[selected_features], y_train)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(X_train_final[selected_features], y_train)
        y_pred_prob = sig_clf.predict_proba(X_val_final[selected_featur
        cv_auc_score.append(roc_auc_score(y_val,y_pred_prob))
        print('For n_estimators {0}, max_depth {1} cross validation AUC
              format(i,j,roc_auc_score(y_val,y_pred_prob)))
```

```
For n_estimators 200, max_depth 7 cross validation AUC score 0.74594
69826033478
For n_estimators 200, max_depth 10 cross validation AUC score 0.7509
055165092412
For n_estimators 500, max_depth 7 cross validation AUC score 0.74624
43061744536
For n_estimators 500, max_depth 10 cross validation AUC score 0.7511
577697532852
For n_estimators 1000, max_depth 7 cross validation AUC score 0.7462
187597785892
For n_estimators 1000, max_depth 10 cross validation AUC score 0.751
1696626961294
For n_estimators 2000, max_depth 7 cross validation AUC score 0.7462
969758260167
```