OPENCLASSROOMS

Construisez un modèle de scoring

Sarah Bitan

Sommaire

SUJETS TRAITÉS

Présentation du projet

JE VAIS VOUS PRÉSENTER NOTRE PROJET D'ÉLABORATION D'UN ALGORITHME DE SCORING DE CRÉDIT, UN OUTIL CRUCIAL POUR ÉVALUER LA PROBABILITÉ DE REMBOURSEMENT D'UN CLIENT ET PRENDRE DES DÉCISIONS ÉCLAIRÉES SUR L'OCTROI DE PRÊTS À LA CONSOMMATION.
CONTEXTE DU PROJET :

- OBJECTIF PRINCIPAL : DÉVELOPPER UN MODÈLE DE CLASSIFICATION POUR DÉTERMINER SI UN PRÊT DOIT ÊTRE ACCORDÉ À UN CLIENT.
- UTILISATEURS CIBLES : CHARGÉS DE RELATION CLIENT.
- CONTRAINTES : MODÈLE INTERPRÉTABLE ET MESURES DE L'IMPORTANCE DES VARIABLES.
-

RESSOURCES ET DIRECTIVES :

- PRÉPARATION DES DONNÉES : UTILISATION D'UN KERNEL KAGGLE ADAPTÉ AVEC LA CRÉATION DE TROIS NOUVELLES VARIABLES POUR AMÉLIORER LA PRÉDICTION.
- INTERPRÉTABILITÉ : MÉTHODE D'IMPORTANCE DES VARIABLES EXPLIQUÉE EN DÉTAIL, AVEC UNE ANALYSE GLOBALE ET LOCALE.

Import du Kernel de Kaggle

ON IMPORTE LE KERNEL DE KAGGLE ET ON L'ADAPTE

Importation du Kernle Kaggle

```
[ ] #on enregistre le temps nécessaire à l'execution
    @contextmanager
    def timer(title):
        t0 = time.time()
        yield
        print("{} - done in {:.0f}s".format(title, time.time() - t0))

    #encodage one-hot pour les colonnes qualitatives
    def one_hot_encoder(df, nan_as_category = True):
        original_columns = list(df.columns)
        categorical_columns = [col for col in df.columns if df[col].dtype == 'object']
        df = pd.get_dummies(df, columns= categorical_columns, dummy_na= nan_as_category)
        new_columns = [c for c in df.columns if c not in original_columns]
        return df, new_columns

    # Preprocess application_train.csv and application_test.csv
    def application_train_test(num_rows = None, nan_as_category = False):
        # Fusion des fichiers
        df = pd.read_csv('/content/drive/MyDrive/openclass/application_train.csv', nrows= num_rows)
        test_df = pd.read_csv('/content/drive/MyDrive/openclass/application_test.csv', nrows= num_rows)
        print("Train samples: {}, test samples: {}".format(len(df), len(test_df)))
        df = df.append(test_df).reset_index()
        # Optional: On supprime 4 applications with XNA CODE_GENDER (train set)
        df = df[df['CODE_GENDER'] != 'XNA']
            # Categorical features with Binary encode (0 or 1; two categories)
        for bin_feature in ['CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY']:
            df[bin_feature], uniques = pd.factorize(df[bin_feature])
        # Categorical features with One-Hot encode
        df, cat_cols = one_hot_encoder(df, nan_as_category)

        # NaN values for DAYS_EMPLOYED: 365.243 -> nan
        df['DAYS_EMPLOYED'].replace(365243, np.nan, inplace= True)
        # Some simple new features (percentages)
        df['DAYS_EMPLOYED_PERC'] = df['DAYS_EMPLOYED'] / df['DAYS_BIRTH']
        df['INCOME_CREDIT_PERC'] = df['AMT_INCOME_TOTAL'] / df['AMT_CREDIT']
```

Analyse exploratoire

ON COMMENCE AVEC L'EXPLORATION DE NOS DONNÉES: TOUT D'ABORD LE JEU DE DONNÉES D'ENTRAINEMENT

```
[ ] # Jeu d'entrainement
    data_train = pd.read_csv('/content/drive/MyDrive/openclass/application_train.csv')
    data_train
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	...	FLAG_DOCUMENT_18	FLAG_DOCUMENT_19	FLAG_DOCUMENT_20	FLAG_D
0	100002	1	Cash loans	M	N	Y	0	202500.0	406597.5	24700.5	...	0.0	0.0	0.0	
1	100003	0	Cash loans	F	N	N	0	270000.0	1293502.5	35698.5	...	0.0	0.0	0.0	
2	100004	0	Revolving loans	M	Y	Y	0	67500.0	135000.0	6750.0	...	0.0	0.0	0.0	
3	100006	0	Cash loans	F	N	Y	0	135000.0	312682.5	29686.5	...	0.0	0.0	0.0	
4	100007	0	Cash loans	M	N	Y	0	121500.0	513000.0	21865.5	...	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
73764	185540	0	Cash loans	F	N	N	0	157500.0	139113.0	15696.0	...	0.0	0.0	0.0	
73765	185541	0	Cash loans	F	Y	Y	0	171000.0	1120500.0	41652.0	...	0.0	0.0	0.0	
73766	185542	0	Cash loans	F	N	Y	1	90000.0	157500.0	17091.0	...	0.0	0.0	0.0	
73767	185543	0	Revolving loans	M	Y	Y	0	225000.0	450000.0	22500.0	...	0.0	0.0	0.0	
73768	185544	0	Cash loans	M	Y	Y	0	202500.0	370629.0	21406.5	...	NaN	NaN	NaN	

73769 rows × 122 columns

ON CONTINUE AVEC NOTRE JEU DE DONNÉES TEST

```
[ ] # Jeu test
    data_test = pd.read_csv('/content/drive/MyDrive/openclass/application_test.csv')
    data_test
```

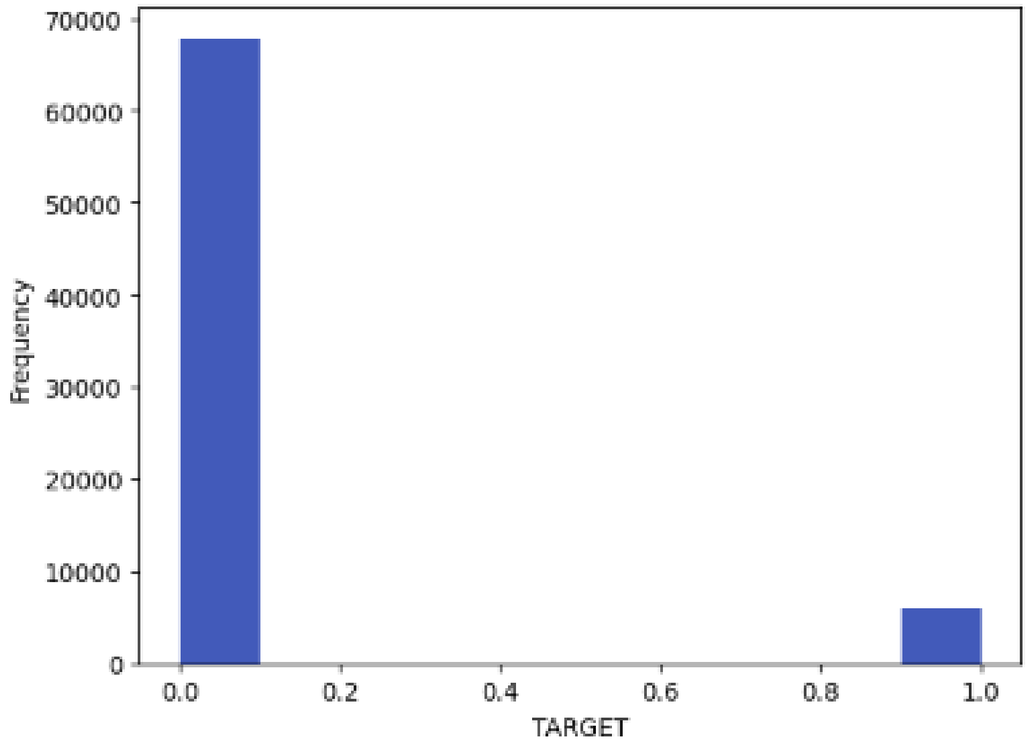
	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	...	FLAG_DOCUMENT_18	FLAG_DOCUMENT_19	FLAG_DOCUMENT_20	FLAG
0	100001	Cash loans	F	N	Y	0	135000.0	568800.0	20560.5	450000.0	...	0	0	0	
1	100005	Cash loans	M	N	Y	0	99000.0	222768.0	17370.0	180000.0	...	0	0	0	
2	100013	Cash loans	M	Y	Y	0	202500.0	663264.0	69777.0	630000.0	...	0	0	0	
3	100028	Cash loans	F	N	Y	2	315000.0	1575000.0	49018.5	1575000.0	...	0	0	0	
4	100038	Cash loans	M	Y	N	1	180000.0	625500.0	32067.0	625500.0	...	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
48739	456221	Cash loans	F	N	Y	0	121500.0	412560.0	17473.5	270000.0	...	0	0	0	
48740	456222	Cash loans	F	N	N	2	157500.0	622413.0	31909.5	495000.0	...	0	0	0	
48741	456223	Cash loans	F	Y	Y	1	202500.0	315000.0	33205.5	315000.0	...	0	0	0	
48742	456224	Cash loans	M	N	N	0	225000.0	450000.0	25128.0	450000.0	...	0	0	0	
48743	456250	Cash loans	F	Y	N	0	135000.0	312768.0	24709.5	270000.0	...	0	0	0	

48744 rows × 121 columns

On s'intéresse à la colonne TARGET

La colonne target n'est pas présente dans le jeu test, on va donc chercher à quoi elle correspond.

```
[ ]  data_train['TARGET'].astype(int).plot.hist();
     plt.xlabel('TARGET');
```



0 correspond aux prets qui ont été remboursés et 1 aux prets qui ont eu des problèmes de remboursements.

On commence l'analyse univariée

ON S'INTÉRESSE AUX DIFFÉRENTS TYPES DE DONNÉES

```
# Nombre de types de chaque colonne
data_train.dtypes.value_counts()
```

```
float64    79
int64      27
object     16
dtype: int64
```

On s'intéresse d'abord aux variables quantitaitves

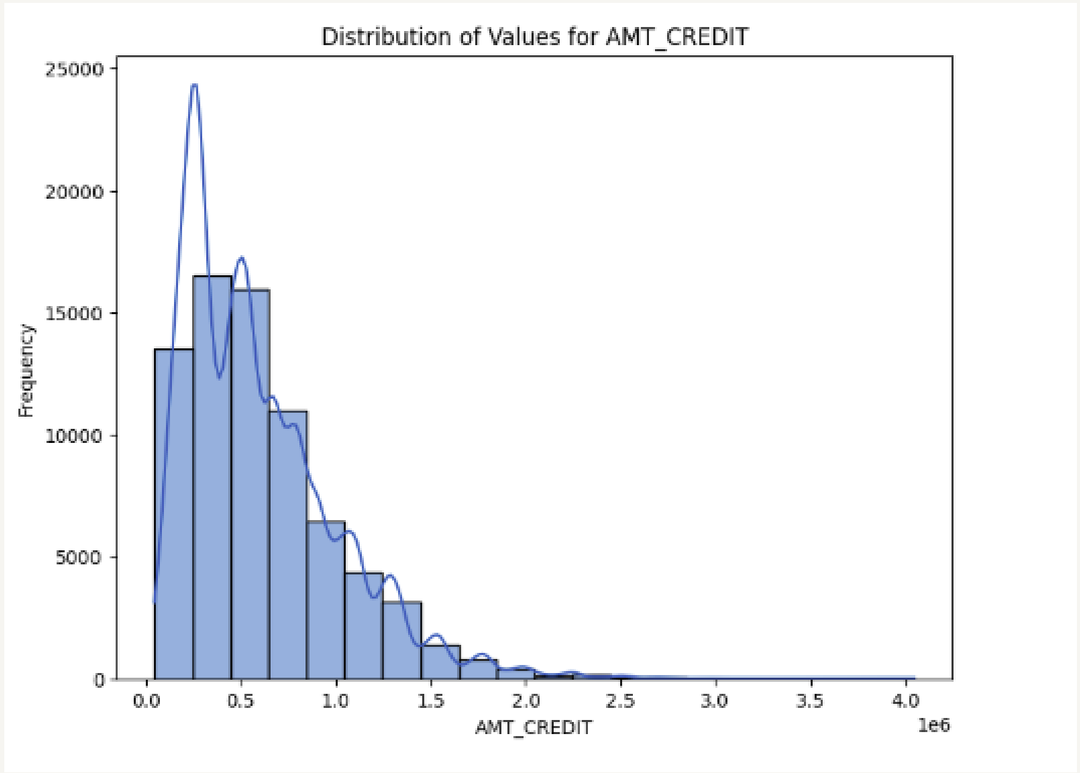
On s'interesse aux variables quantitatives

```
[ ] # Nombre de valeurs uniques pour chaque colonne
    data_train.select_dtypes('float').apply(pd.Series.nunique, axis = 0)
```

```
AMT_INCOME_TOTAL              1014
AMT_CREDIT                    3819
AMT_ANNUITY                   9895
AMT_GOODS_PRICE                567
REGION_POPULATION_RELATIVE      80
                               ...
AMT_REQ_CREDIT_BUREAU_DAY        7
AMT_REQ_CREDIT_BUREAU_WEEK       7
AMT_REQ_CREDIT_BUREAU_MON       20
AMT_REQ_CREDIT_BUREAU_QRT        9
AMT_REQ_CREDIT_BUREAU_YEAR      19
Length: 79, dtype: int64
```

```
[ ] # Nombre de valeurs uniques pour chaque colonne
    data_train.select_dtypes('int64').apply(pd.Series.nunique, axis = 0)
```

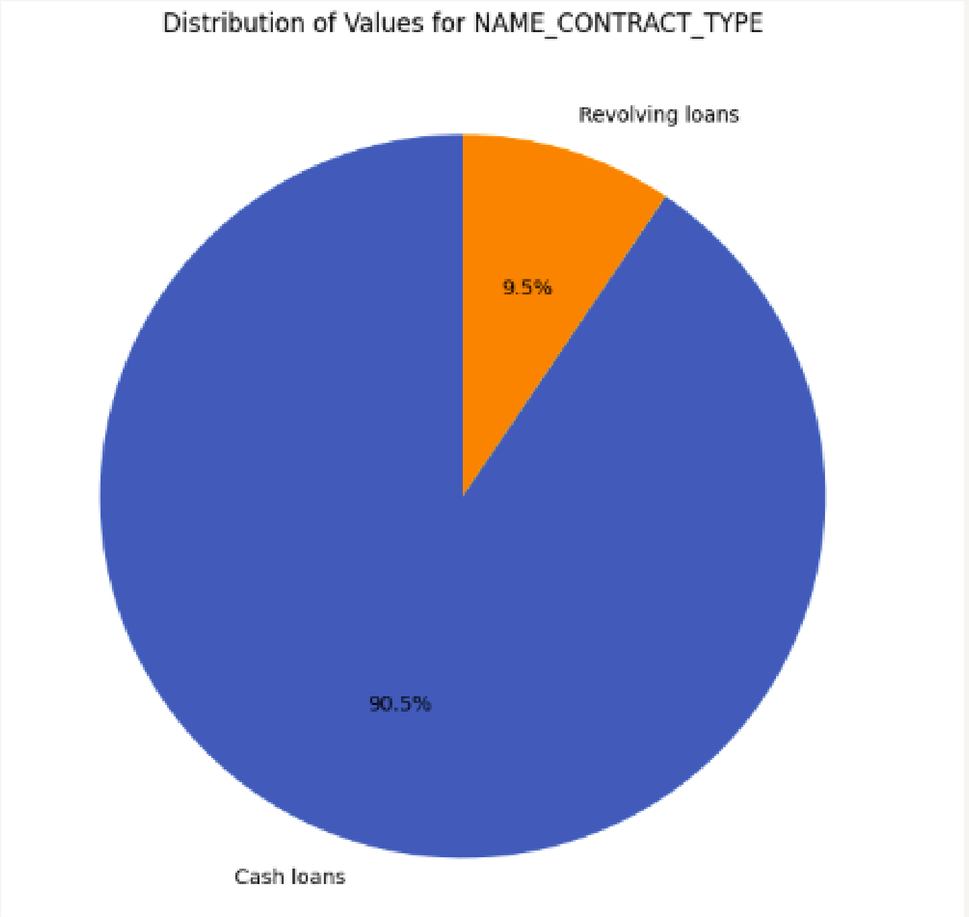
```
SK_ID_CURR                   73769
TARGET                           2
CNT_CHILDREN                    11
DAYS_BIRTH                   16635
DAYS_EMPLOYED                 9086
DAYS_ID_PUBLISH               5884
FLAG_MOBIL                       2
FLAG_EMP_PHONE                   2
FLAG_WORK_PHONE                  2
FLAG_CONT_MOBILE                 2
FLAG_PHONE                       2
FLAG_EMAIL                       2
REGION_RATING_CLIENT             3
REGION_RATING_CLIENT_W_CITY      3
HOUR_APPR_PROCESS_START         24
REG_REGION_NOT_LIVE_REGION       2
REG_REGION_NOT_WORK_REGION       2
LIVE_REGION_NOT_WORK_REGION      2
REG_CITY_NOT_LIVE_CITY           2
REG_CITY_NOT_WORK_CITY           2
LIVE_CITY_NOT_WORK_CITY          2
FLAG_DOCUMENT_2                  2
FLAG_DOCUMENT_3                  2
FLAG_DOCUMENT_4                  2
```



Puis aux variables qualitatives

```
[ ] # Nombre de valeurs uniques pour chaque colonne
    data_train.select_dtypes('object').apply(pd.Series.nunique, axis = 0)
```

```
NAME_CONTRACT_TYPE             2
CODE_GENDER                    3
FLAG_OWN_CAR                   2
FLAG_OWN_REALTY                2
NAME_TYPE_SUITE                7
NAME_INCOME_TYPE               8
NAME_EDUCATION_TYPE            5
NAME_FAMILY_STATUS             6
NAME_HOUSING_TYPE              6
OCCUPATION_TYPE               18
WEEKDAY_APPR_PROCESS_START     7
ORGANIZATION_TYPE             58
FONDKAPREMONT_MODE             4
HOUSETYPE_MODE                 3
WALLSMATERIAL_MODE             7
EMERGENCYSTATE_MODE            2
dtype: int64
```



Jointure des fichiers

```
[ ] #On va merger ensemble les fichiers application_train.csv et application_test.csv, pour y effectuer un nettoyage
    # Jointure sur la colonne commune
    jointure = data_test.append(data_train).reset_index()
    # Affichage du résultat
    jointure.head()
```

	index	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	...	FLAG_DOCUMENT_19
0	0	100001	Cash loans	F	N	Y	0	135000.0	568800.0	20560.5	...	0.0
1	1	100005	Cash loans	M	N	Y	0	99000.0	222768.0	17370.0	...	0.0
2	2	100013	Cash loans	M	Y	Y	0	202500.0	663264.0	69777.0	...	0.0
3	3	100028	Cash loans	F	N	Y	2	315000.0	1575000.0	49018.5	...	0.0
4	4	100038	Cash loans	M	Y	N	1	180000.0	625500.0	32067.0	...	0.0

5 rows × 123 columns

```
[ ] # On verifie si il y'a des doublons basés sur l'id
    duplicates = jointure[jointure.duplicated(subset='SK_ID_CURR')]
    print(duplicates)
```

```
Empty DataFrame
Columns: [index, SK_ID_CURR, NAME_CONTRACT_TYPE, CODE_GENDER, FLAG_OWN_CAR, FLAG_OWN_REALTY, CNT_CHILDREN, AMT_INCOME_TOTAL, AMT_CREDIT, AMT_ANNUITY, AMT_GOODS_PR
Index: []

[0 rows x 123 columns]
```

Valeurs manquantes

ON LES AFFICHE

```python
# Calcul du pourcentage de valeurs manquantes pour chaque variable
missing_percentage = (jointure.isnull().mean() * 100).sort_values(ascending=False)

# Création d'un DataFrame pour affichage
missing_data = pd.DataFrame({'Variable': missing_percentage.index, 'Missing Percentage': missing_percentage.values})

# Affichage du DataFrame des valeurs manquantes
print(missing_data)
# Filtrage pour afficher uniquement les variables avec des valeurs manquantes
variables_with_missing = missing_data[missing_data['Missing Percentage'] > 50]

# Affichage des variables avec des valeurs manquantes
print(variables_with_missing)
```

```
                    Variable  Missing Percentage
0               COMMONAREA_AVG           69.383657
1              COMMONAREA_MEDI           69.383657
2              COMMONAREA_MODE           69.383657
3     NONLIVINGAPARTMENTS_AVG           68.988597
4    NONLIVINGAPARTMENTS_MEDI           68.988597
..                        ...                 ...
118          NAME_INCOME_TYPE            0.000000
119                AMT_CREDIT            0.000000
120          AMT_INCOME_TOTAL            0.000000
121           FLAG_OWN_REALTY            0.000000
122                     index            0.000000

[123 rows x 2 columns]
                    Variable  Missing Percentage
0               COMMONAREA_AVG           69.383657
1              COMMONAREA_MEDI           69.383657
2              COMMONAREA_MODE           69.383657
3     NONLIVINGAPARTMENTS_AVG           68.988597
4    NONLIVINGAPARTMENTS_MEDI           68.988597
..                        ...                 ...
77          FLAG_DOCUMENT_11            0.000816
78          FLAG_DOCUMENT_13            0.000816
79          FLAG_DOCUMENT_14            0.000816
80          FLAG_DOCUMENT_15            0.000816
81          FLAG_DOCUMENT_17            0.000816

[82 rows x 2 columns]
```

ON SUPPRIME LES VALEURS AVEC PLUS DE 50% DE VALEURS MANQUANTES

```
[ ]
    # Filtrage des variables avec plus de 50% de valeurs manquantes
    variables_to_drop = missing_percentage[missing_percentage > 50].index

    # Suppression des variables sélectionnées du DataFrame
    jointure = jointure.drop(columns=variables_to_drop)

    # Affichage des variables supprimées
    print("Variables supprimées:")
    print(variables_to_drop)
```

```
Variables supprimées:
Index(['COMMONAREA_AVG', 'COMMONAREA_MEDI', 'COMMONAREA_MODE',
       'NONLIVINGAPARTMENTS_AVG', 'NONLIVINGAPARTMENTS_MEDI',
       'NONLIVINGAPARTMENTS_MODE', 'LIVINGAPARTMENTS_AVG',
       'LIVINGAPARTMENTS_MEDI', 'LIVINGAPARTMENTS_MODE', 'FONDKAPREMONT_MODE',
       'FLOORSMIN_AVG', 'FLOORSMIN_MODE', 'FLOORSMIN_MEDI', 'OWN_CAR_AGE',
       'YEARS_BUILD_AVG', 'YEARS_BUILD_MEDI', 'YEARS_BUILD_MODE',
       'LANDAREA_AVG', 'LANDAREA_MEDI', 'LANDAREA_MODE', 'BASEMENTAREA_MODE',
       'BASEMENTAREA_MEDI', 'BASEMENTAREA_AVG', 'NONLIVINGAREA_MODE',
       'NONLIVINGAREA_AVG', 'NONLIVINGAREA_MEDI', 'ELEVATORS_MODE',
       'ELEVATORS_MEDI', 'ELEVATORS_AVG', 'EXT_SOURCE_1', 'WALLSMATERIAL_MODE',
       'APARTMENTS_MEDI', 'APARTMENTS_AVG', 'APARTMENTS_MODE'],
      dtype='object')
```

Suppression des variables fortement corrélées →

Création de nouvelles variables

```
[ ] #proportion de la vie professionnelle par rapport à la durée totale de la vie
    jointure['DAYS_EMPLOYED_PERC'] = jointure['DAYS_EMPLOYED'] / jointure['DAYS_BIRTH']
    #capacité d'une personne à rembourser un prêt en fonction de son revenu
    jointure['INCOME_CREDIT_PERC'] = jointure['AMT_INCOME_TOTAL'] / jointure['AMT_CREDIT']
    #Cette variable représente le revenu par personne, calculé en divisant le revenu total (AMT_INCOME_TOTAL) par le nombre de membres de la famille (CNT_FAM_MEMBERS).
    jointure['INCOME_PER_PERSON'] = jointure['AMT_INCOME_TOTAL'] / jointure['CNT_FAM_MEMBERS']
    jointure
```

	index	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	...	AMT_REQ_CREDIT_BUREA
0	0	100001	Cash loans	F	N	Y	0	135000.0	568800.0	20560.5	...	
1	1	100005	Cash loans	M	N	Y	0	99000.0	222768.0	17370.0	...	
2	2	100013	Cash loans	M	Y	Y	0	202500.0	663264.0	69777.0	...	
3	3	100028	Cash loans	F	N	Y	2	315000.0	1575000.0	49018.5	...	
4	4	100038	Cash loans	M	Y	N	1	180000.0	625500.0	32067.0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
122508	73764	185540	Cash loans	F	N	N	0	157500.0	139113.0	15696.0	...	
122509	73765	185541	Cash loans	F	Y	Y	0	171000.0	1120500.0	41652.0	...	
122510	73766	185542	Cash loans	F	N	Y	1	90000.0	157500.0	17091.0	...	
122511	73767	185543	Revolving loans	M	Y	Y	0	225000.0	450000.0	22500.0	...	
122512	73768	185544	Cash loans	M	Y	Y	0	202500.0	370629.0	21406.5	...	

122513 rows × 79 columns

Imputation des valeurs manquantes pour les variables qualitatives

```
[ ]  # Imputation pour les variables qualitatives
     qual_imputer = SimpleImputer(strategy='most_frequent')
     jointure[categorical_columns] = qual_imputer.fit_transform(jointure[categorical_columns])

     # Vérifier le pourcentage de valeurs manquantes après l'imputation
     print(jointure.isnull().mean() * 100)
```

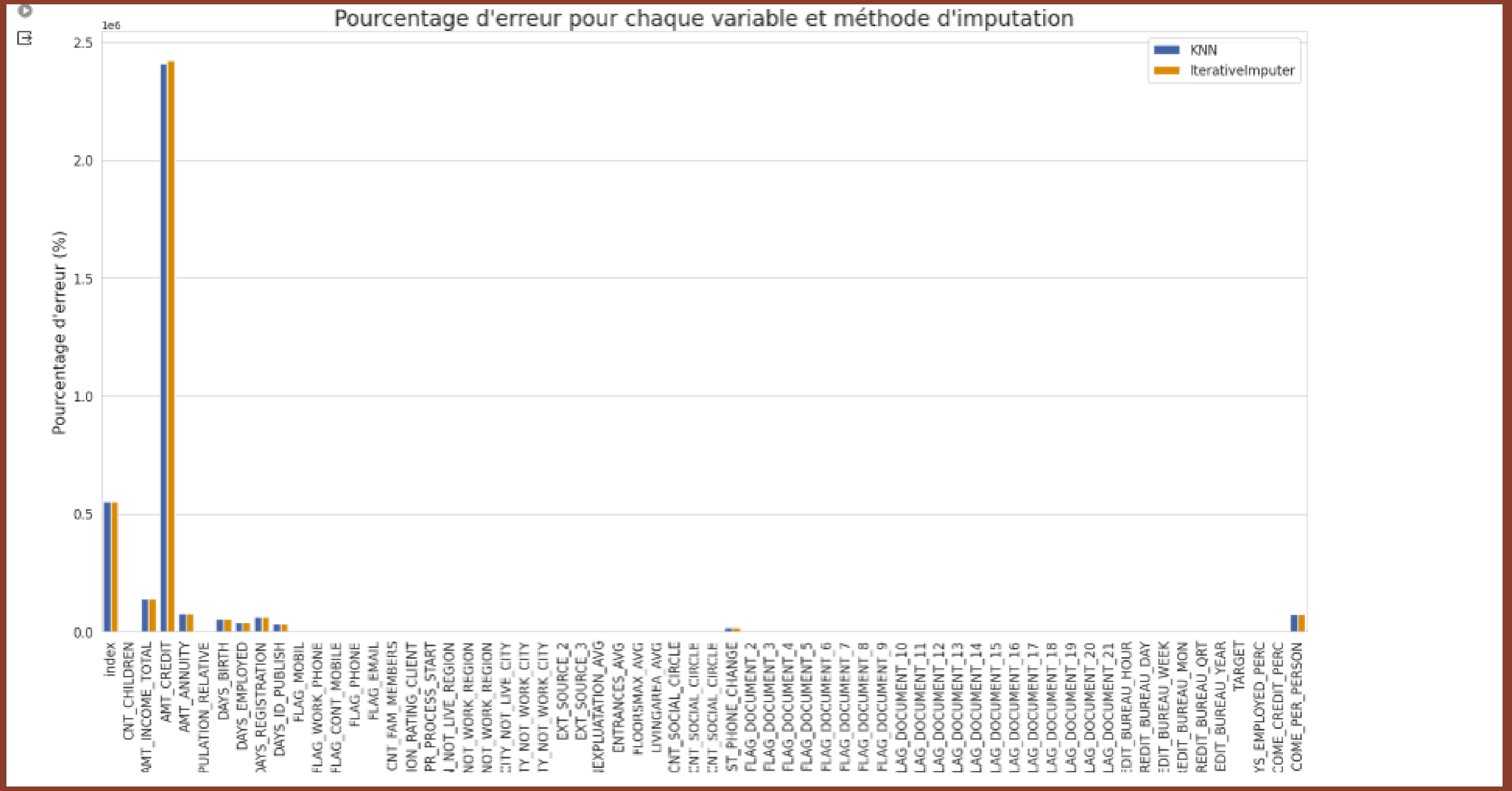
```
index                          0.000000
SK_ID_CURR                     0.000000
NAME_CONTRACT_TYPE             0.000000
CODE_GENDER                    0.000000
FLAG_OWN_CAR                   0.000000
                                  ...
AMT_REQ_CREDIT_BUREAU_YEAR    12.987193
TARGET                        39.786798
DAYS_EMPLOYED_PERC             0.000000
INCOME_CREDIT_PERC             0.000000
INCOME_PER_PERSON              0.000816
Length: 79, dtype: float64
```

Imputation des valeurs manquantes pour les variables quantitatives

```
[ ]  # Supprimer la variable SK_ID_CURR des colonnes quantitatives
     quantitative_columns = quantitative_columns.drop('SK_ID_CURR', errors='ignore')

     # Maintenant, quantitative_columns contient les noms des colonnes quantitatives sans SK_ID_CURR
     print(quantitative_columns)
```

```
Index(['SK_ID_CURR', 'TARGET', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL',
       'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE',
       'REGION_POPULATION_RELATIVE', 'DAYS_BIRTH', 'DAYS_EMPLOYED',
       ...
       'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20',
       'FLAG_DOCUMENT_21', 'AMT_REQ_CREDIT_BUREAU_HOUR',
       'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK',
       'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT',
       'AMT_REQ_CREDIT_BUREAU_YEAR'],
      dtype='object', length=106)
```

```
[ ]  variables=quantitative_columns
     # Sélection dun sous-ensemble de 5% des données
     data_subset = jointure.sample(frac=0.05, random_state=42)

     # Diviser les données en ensemble d'entrainement et de test
     train, test = train_test_split(data_subset, test_size=0.2, random_state=42) # Création d'un DataFrame pour stocker les résultats
     results = pd.DataFrame(columns=['Variable', 'Méthode', 'Erreur', 'Pourcentage d\'erreur'])  # Calcul de l'erreur de prédiction pour chaque méthode d'imputation
```


Pourcentage d'erreur pour chaque variable et méthode d'imputation
1e6
2.5
2.0
1.5
1.0
0.5
0.0
Pourcentage d'erreur (%)
KNN
IterativeImputer
index
CNT_CHILDREN
AMT_INCOME_TOTAL
AMT_CREDIT
AMT_ANNUITY
DAYS_BIRTH
DAYS_EMPLOYED
DAYS_ID_PUBLISH
FLAG_MOBIL
FLAG_EMAIL
FLAG_PHONE
REGION_RATING_CLIENT
REG_CITY_NOT_LIVE_CITY
REG_CITY_NOT_WORK_CITY
EXT_SOURCE_2
EXT_SOURCE_3
FLAG_DOCUMENT_2
FLAG_DOCUMENT_3
FLAG_DOCUMENT_4
FLAG_DOCUMENT_5
FLAG_DOCUMENT_6
FLAG_DOCUMENT_7
FLAG_DOCUMENT_8
FLAG_DOCUMENT_9
FLAG_DOCUMENT_10
FLAG_DOCUMENT_11
FLAG_DOCUMENT_12
FLAG_DOCUMENT_13
FLAG_DOCUMENT_14
FLAG_DOCUMENT_15
FLAG_DOCUMENT_16
FLAG_DOCUMENT_17
FLAG_DOCUMENT_18
FLAG_DOCUMENT_19
FLAG_DOCUMENT_20
FLAG_DOCUMENT_21
TARGET

On impute pour chaque variable avec la méthode qui lui est la plus efficace

```
[ ] # Instanciation de l'IterativeImputer
    imputer = IterativeImputer(random_state=0)

    # Liste des colonnes à imputer avec IterativeImputer
    columns_to_impute = IterativeImputer_variables
```

```
[ ] from sklearn.impute import IterativeImputer
    import pandas as pd

    # Create an IterativeImputer instance
    imputer = IterativeImputer()

    # Imputation of missing values for each selected column
    for column in columns_to_impute:
        if column not in subset_products_data.columns:
            print(f"Warning: Column '{column}' not found in the DataFrame.")
            continue

        # Extract the column as a 2D array (required by IterativeImputer)
        column_data = subset_products_data[[column]].values

        # Impute missing values using fit_transform
        imputed_column = imputer.fit_transform(column_data)

        # Assign the imputed values back to the DataFrame
        subset_products_data[column] = imputed_column

    # Verification of the remaining percentage of missing values
    print(subset_products_data.isnull().mean())
```

```
Warning: Column 'Variable' not found in the DataFrame.
Warning: Column 'Méthode' not found in the DataFrame.
Warning: Column 'Erreur' not found in the DataFrame.
Warning: Column 'Pourcentage d'erreur' not found in the DataFrame.
index                          0.000000
SK_ID_CURR                     0.000000
NAME_CONTRACT_TYPE             0.000000
CODE_GENDER                    0.000000
FLAG_OWN_CAR                   0.000000
                                 ...
AMT_REQ_CREDIT_BUREAU_YEAR     0.128469
TARGET                         0.393895
DAYS_EMPLOYED_PERC             0.000000
INCOME_CREDIT_PERC             0.000000
INCOME_PER_PERSON              0.000000
Length: 79, dtype: float64
```

```
[ ] from sklearn.impute import KNNImputer

    columns_to_impute = knn_results
    # Create a KNNImputer instance
    imputer = KNNImputer()

    # Imputation of missing values for each selected column
    for column in columns_to_impute:
        if column not in subset_products_data.columns:
            print(f"Warning: Column '{column}' not found in the DataFrame.")
            continue

        # Extract the column as a 2D array (required by KNNImputer)
        column_data = subset_products_data[[column]].values

        # Impute missing values using fit_transform
        imputed_column = imputer.fit_transform(column_data)

        # Assign the imputed values back to the DataFrame
        subset_products_data[column] = imputed_column

    # Verification of the remaining percentage of missing values
    print(subset_products_data.isnull().mean())
```

```
Warning: Column 'Variable' not found in the DataFrame.
Warning: Column 'Méthode' not found in the DataFrame.
Warning: Column 'Erreur' not found in the DataFrame.
Warning: Column 'Pourcentage d'erreur' not found in the DataFrame.
index                          0.000000
SK_ID_CURR                     0.000000
NAME_CONTRACT_TYPE             0.000000
CODE_GENDER                    0.000000
FLAG_OWN_CAR                   0.000000
                                 ...
AMT_REQ_CREDIT_BUREAU_YEAR     0.128469
TARGET                         0.393895
DAYS_EMPLOYED_PERC             0.000000
INCOME_CREDIT_PERC             0.000000
INCOME_PER_PERSON              0.000000
Length: 79, dtype: float64
```

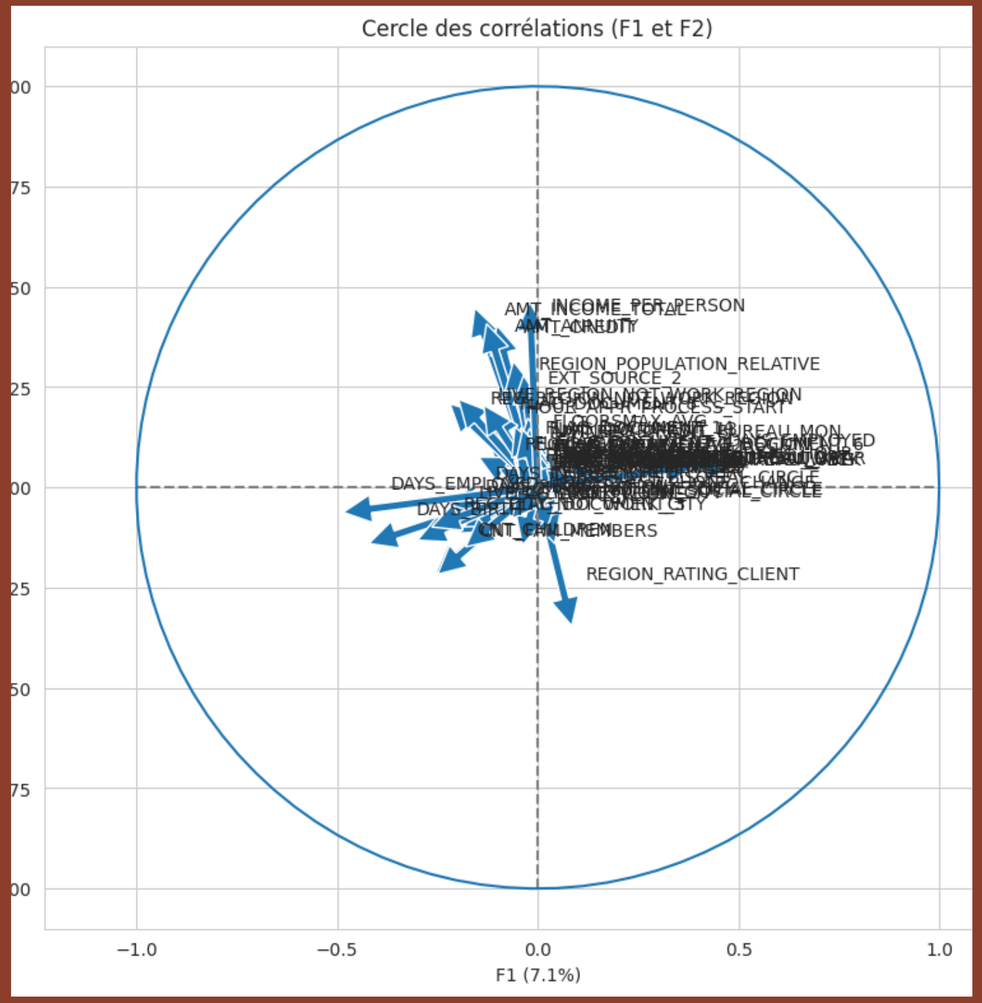
ENCODING

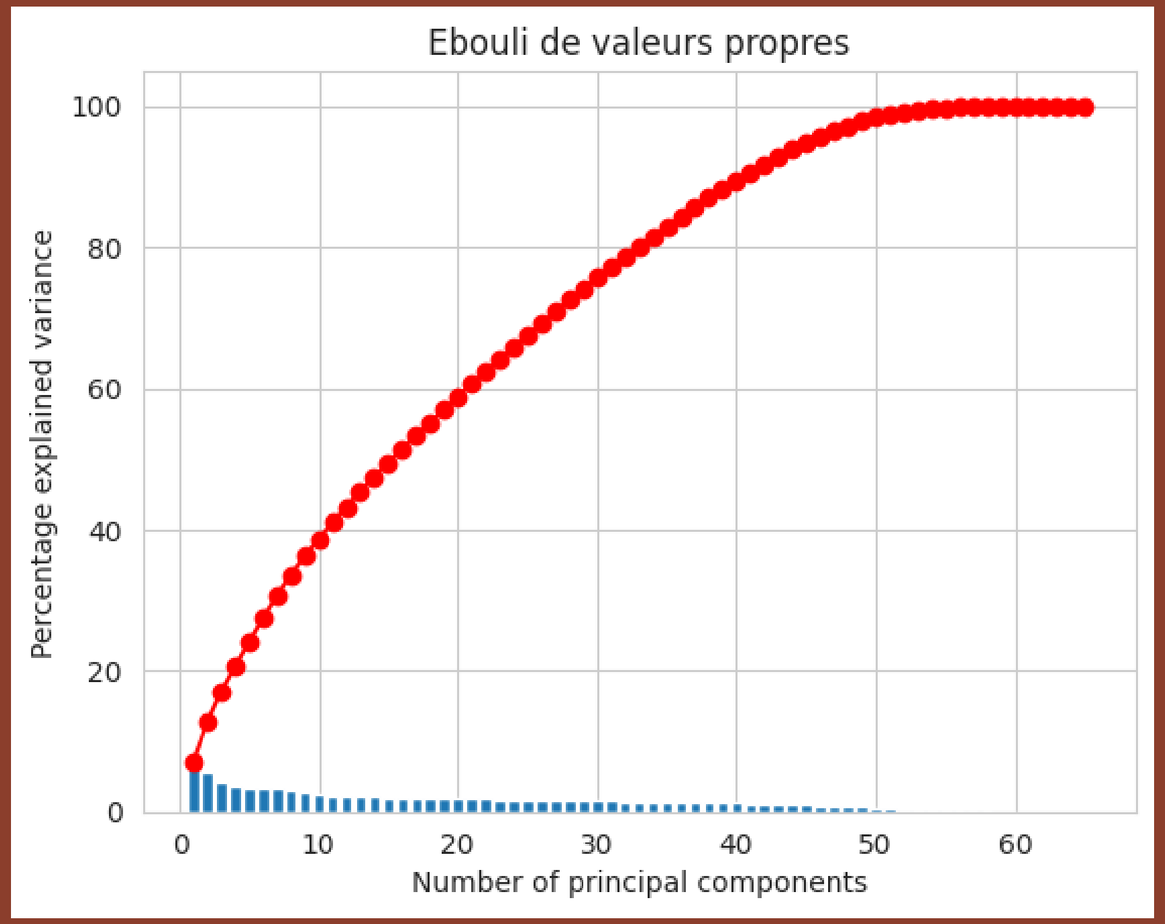
Le LabelEncoder est une classe fournie par la bibliothèque scikit-learn, utilisée pour convertir des variables catégorielles en variables numériques. Dans de nombreux algorithmes d'apprentissage automatique, il est nécessaire de convertir les données catégorielles en données numériques pour les utiliser efficacement dans les modèles.

Le LabelEncoder attribue un entier unique à chaque catégorie présente dans la colonne catégorielle. Par exemple, si une colonne contient les catégories "rouge", "vert" et "bleu", le LabelEncoder peut leur attribuer respectivement les entiers 0, 1 et 2.

```python
# Importation du LabelEncoder depuis scikit-learn
from sklearn.preprocessing import LabelEncoder

# Initialisation du LabelEncoder et des compteurs
le = LabelEncoder()          # Création d'un objet LabelEncoder
le_count = 0                 # Compteur pour suivre le nombre de colonnes encodées
col_encoded = []             # Liste pour stocker les noms des colonnes encodées

# Boucle à travers les colonnes du dataframe
for col in data:

    # Vérification du type de données de la colonne
    if data[col].dtype == 'object':

        # Vérification du nombre unique de catégories dans la colonne
        if len(list(data[col].unique())) <= 2:

            # Encodage de la colonne avec le LabelEncoder
            le.fit(data[col])              # Ajustement du LabelEncoder sur les valeurs
```

ACP


Cercle des corrélations (F1 et F2)
INCOME_PER_PERSON
REGION_POPULATION_RELATIVE
EXT_SOURCE_2
REGION_RATING_CLIENT
−1.0
−0.5
0.0
0.5
1.0
F1 (7.1%)


Ebouli de valeurs propres
Percentage explained variance
0
20
40
60
80
100
0
10
20
30
40
50
60
Number of principal components

Modèle Dummy regressor

Le DummyRegressor est une classe qui permet de créer un modèle de régression très simple utilisé comme point de référence ou de comparaison avec d'autres modèles plus complexes. Il s'agit d'un modèle basé sur des règles simples pour la prédiction, et il est principalement utilisé pour évaluer la performance d'autres modèles de régression.

```
[49] import pandas as pd
     from sklearn.model_selection import train_test_split  # Importez train_test_split
     from sklearn.dummy import DummyRegressor
     from sklearn.metrics import mean_squared_error
     import numpy as np


     X = data.drop('TARGET', axis=1)
     y = data['TARGET']

     # Divisez les données en ensembles d'entraînement et de test
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)



     # Divisez les données en ensembles d'entraînement et de test
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

     # Créez un Dummy Regressor
     dummy_regressor = DummyRegressor(strategy='mean')

     # Entraînez le modèle sur l'ensemble d'entraînement
     dummy_regressor.fit(X_train, y_train)

     # Faites des prédictions sur l'ensemble de test
     y_pred = dummy_regressor.predict(X_test)

     # Calculez la métrique (par exemple, Mean Squared Error) pour évaluer la baseline
     mse_baseline = mean_squared_error(y_test, y_pred)

     print(f'Mean Squared Error (Baseline): {mse_baseline}')


     Mean Squared Error (Baseline): 0.03590733528285285
```

PIPELINE D'APPRENTISSAGE AUTOMATIQUE AVEC SURÉCHANTILLONNAGE SMOTE

On utilise la méthode smote car les données sont imbalanced (la méthode doit être intégré lors du training avec CV + gridsearch) et non à part. Sinon le rééquilibrage des données n'est pas bon. Il faut utiliser un pipeline fourni par imblearn car celui de sickit learn n’intègre pas la méthode smote.

```
#Pipeline avec SMOTE pour l'équilibrage de classes et recherche des meilleurs hyperparamètres

# Division des données en ensembles de formation et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Création d'un pipeline avec SMOTE pour l'équilibrage des classes
pipeline = Pipeline([
    ('smote', SMOTE(random_state=42))  # Utilisation de SMOTE pour équilibrer les classes
])

# Paramètres pour la grille de recherche des hyperparamètres
param_grid = {
    'smote__sampling_strategy': [0.5, 0.75, 1.0],  # Différentes stratégies d'échantillonnage pour SMOTE
    'classifier__kernel': ['linear', 'poly', 'rbf', 'sigmoid'], # Différents noyaux pour le SVM
    'classifier__C': [0.1, 1, 10],  # Paramètre de régularisation C pour le SVM
}

# Initialisation de la grille de recherche avec le pipeline
grid_search = GridSearchCV(pipeline, param_grid, scoring='f1', cv=5)  # Utilisation de GridSearchCV pour la recherche des meilleurs hyperparamètres
```

Précision (Precision) : C'est la proportion de prédictions positives correctes parmi toutes les prédictions positives. Dans le contexte des prêts, une haute précision signifie que lorsque le modèle prédit que le client est éligible pour un prêt, il a raison la plupart du temps.

Rappel (Recall) : C'est la proportion de clients éligibles pour un prêt que le modèle a correctement identifiés. Un rappel élevé signifie que le modèle capture un grand pourcentage de clients qui sont réellement éligibles.

Score F1 : C'est la moyenne pondérée de la précision et du rappel. Il est utile lorsque les classes sont déséquilibrées.

Exactitude (Accuracy) : C'est la proportion de prédictions correctes parmi toutes les prédictions. Cependant, l'exactitude peut être trompeuse si les classes sont déséquilibrées.

En comparant les métriques, on choisit le modèle de régression logistique.

ON COMPARE DIFFÉRENTS MODÈLES À L'AIDE DES MÉTRIQUES LE F1 SCORE

```python
# Boucle sur les modèles
for model_name, model in models_to_test.items():
    # Mettre à jour le modèle dans le pipeline
    pipeline.steps[-1] = ('classifier', model)

    # Initialiser la grille de recherche avec le pipeline
    grid_search = GridSearchCV(pipeline, param_grid, scoring='f1', cv=5)

    # Adapter le modèle avec les données d'entraînement
    grid_search.fit(X_train, y_train)

    # Prédiction sur les données d'entraînement
    y_pred = grid_search.predict(X_train)

    # Calculer le score F1
    f1 = f1_score(y_train, y_pred)

    # Afficher le score F1 pour le modèle actuel
    print(f"Score F1 pour {model_name}: {f1:.4f}")
```

```
Score F1 pour Logistic Regression: 0.1707
Score F1 pour SVM: 0.1560
Score F1 pour Random Forest: 1.0000
Score F1 pour XGBoost: 1.0000
```

ROC PRECISION RECALL SCORE MÉTIER

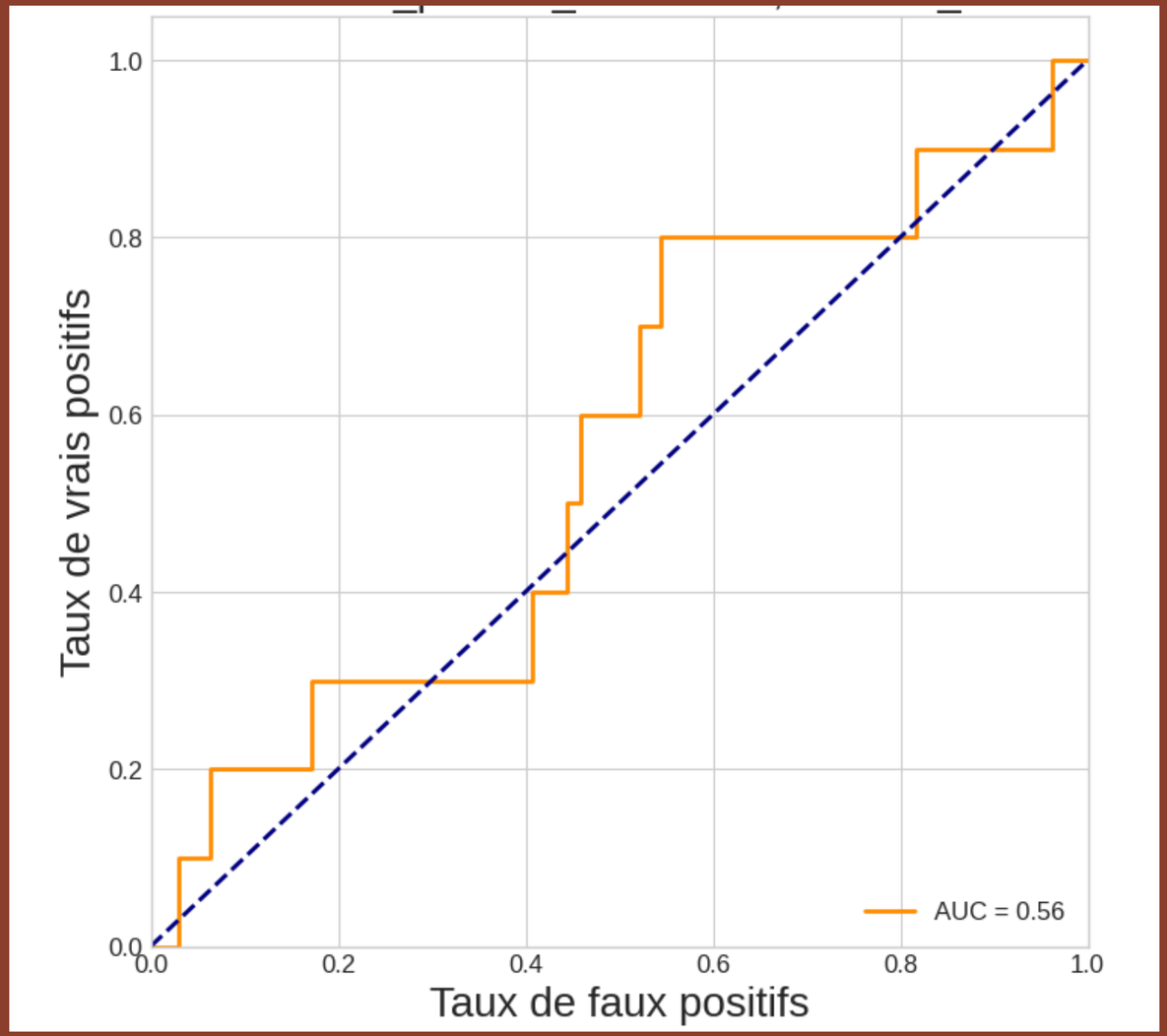
```
8] pd.set_option('display.max_rows', 1000)
   final_results
```

	name	ROC	PRECISION	RECALL	BestParamsM1	BestParamsM2	BestParamsM3	Score métier
0	logistic	0.618380	0.135128	0.480057	{'classifier__C': 1, 'classifier__penalty': 'l1'}	{'classifier__C': 1000, 'classifier__penalty':...	{'classifier__C': 100, 'classifier__penalty': ...	100.0
1	SVM	0.564974	0.079490	0.340456	{'classifier__C': 1.0, 'classifier__gamma': 's...	{'classifier__C': 10, 'classifier__gamma': 'sc...	{'classifier__C': 10, 'classifier__gamma': 'sc...	100.0
2	XGBoost	0.710019	0.355556	0.416904	{'classifier__alpha': 0.001, 'classifier__boos...	{'classifier__alpha': 0.1, 'classifier__booste...	{'classifier__alpha': 0.001, 'classifier__boos...	100.0
3	RF	0.694979	0.248366	0.394112	{'classifier__criterion': 'entropy', 'classifi...	{'classifier__criterion': 'gini', 'classifier_...	{'classifier__criterion': 'gini', 'classifier_...	100.0

COURBE ROC DE NOS MODÈLE

La courbe ROC et l'AUC sont utilisées pour évaluer la capacité d'un modèle de classification à faire la distinction entre les exemples positifs et négatifs.

Une courbe ROC proche du coin supérieur gauche de la figure et une AUC proche de 1 indiquent une bonne performance du modèle.



LE MODÈLE CHOISI

On choisi le modèle XGBoost après comparaison des différentes métriques avec les autres modèles.

Modèle choisi

On entraine mainteant le modèle XGBoost

```
[53] # Entraîner le meilleur modèle avec toutes les données sans validation croisée
     # J'ai choisi d'utiliser SMOTE même s'il n'y a pas de validation croisée car les données sont très déséquilibrées

     smote = SMOTE(random_state=11)
     X_train, y_train = smote.fit_resample(X_train, y_train)
```

```
# On entraîne nos données
classifier = XGBClassifier(alpha=0.001, booster="gblinear",
                           gamma=0, learning_rate=0.1,
                           max_depth=8, reg_lambda=0.5)
classifier.fit(X_train, y_train)

# On évalue le modèle sur l'ensemble de test et ajoute les résultats à un DataFrame
results = pd.DataFrame(columns=["name", "ROC", "PRECISION", "RECALL", "Score métier"])
results = evaluate_model_on_test("XGBoost", X_test, y_test, classifier, results)
results
```

	name	ROC	PRECISION	RECALL	Score métier
0	XGBoost	0.744403	0.06383	0.3	100.0

PRÉDICTION

Prédiction sur les données test

```
# Predictions
print("Predictions sur les données test")
y_test_pred = classifier.predict(X_test)

print("Les 5 premires prédictions sur le jeux de données test")
print(y_test_pred[0:5])

print('score : ', classifier.score(X_test,y_test))
```

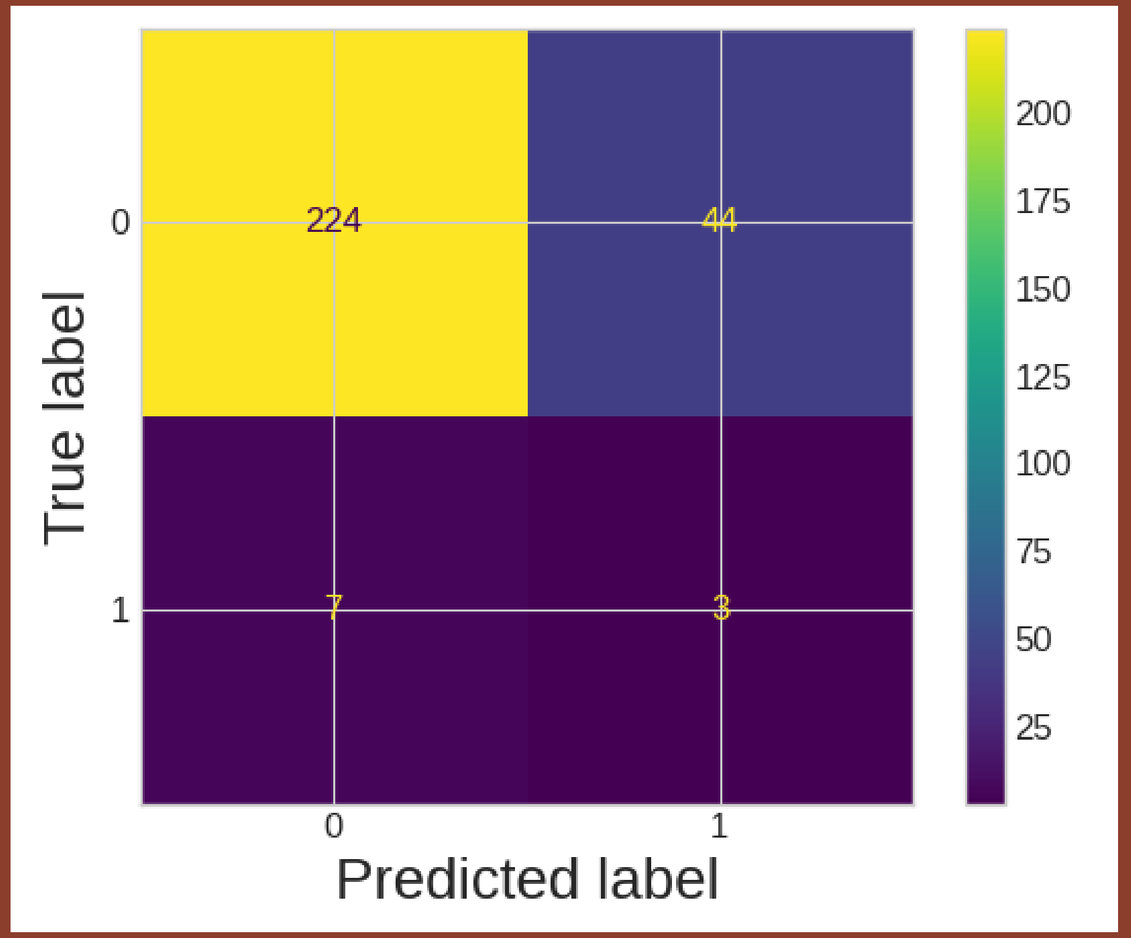
```
Predictions sur les données test
Les 5 premires prédictions sur le jeux de données test
[0 0 0 0 0]
score :  0.8165467625899281
```

```
[58] # Commentaire sur l'importation de la bibliothèque collections
     import collections

     # Compter les occurrences de chaque classe dans y_test
     collections.Counter(y_test)
```

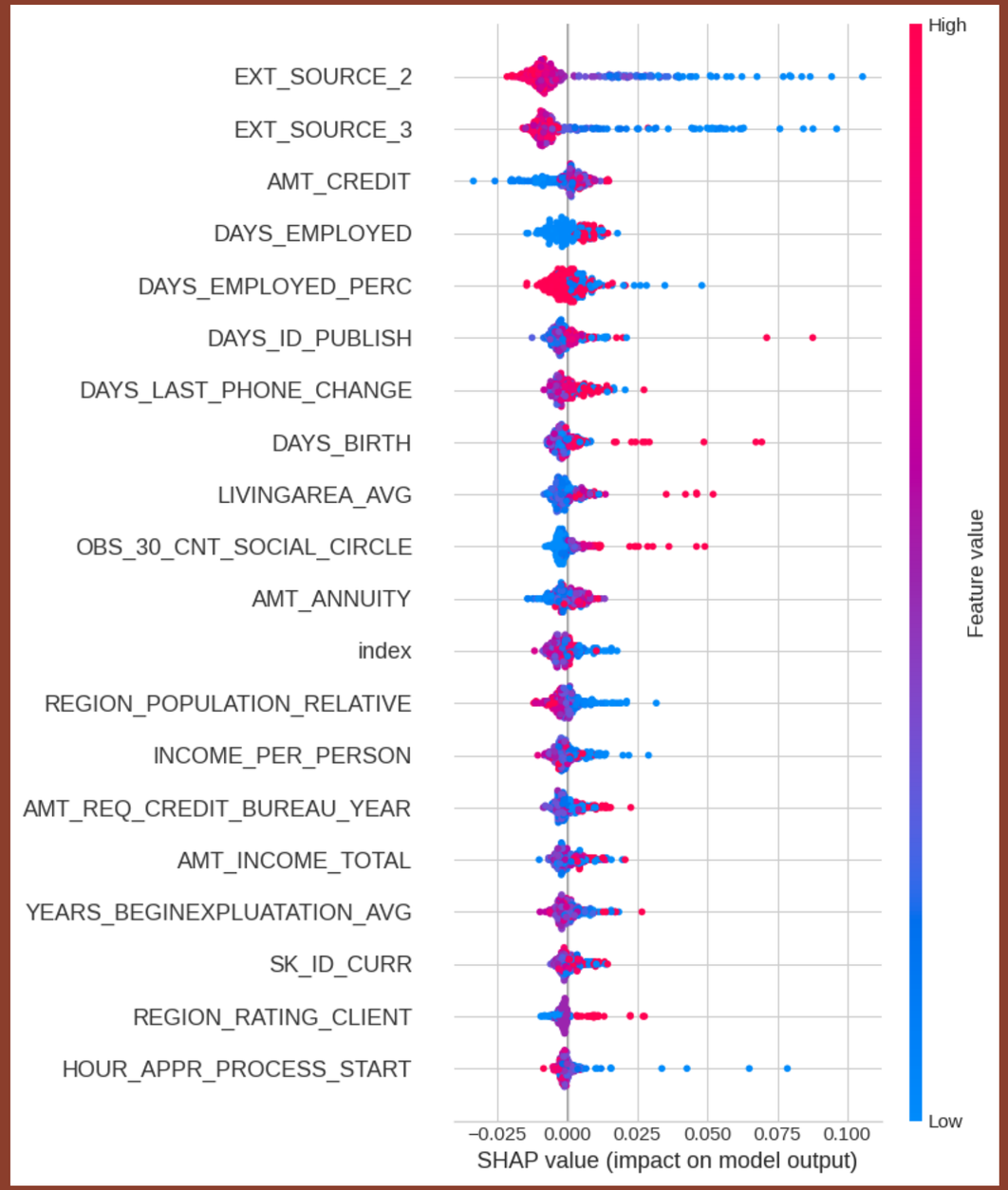
```
Counter({0.0: 268, 1.0: 10})
```

```
[59] # Compter les occurrences de chaque classe dans y_test_pred
     collections.Counter(y_test_pred)
```

```
Counter({0: 231, 1: 47})
```



FEATURE IMPORTANCE



CONCLUSION

Nous avons entrepris la construction d'un modèle de scoring pour évaluer le risque de défaut de paiement des clients. Notre objectif principal était de développer un modèle prédictif capable de classer les clients en fonction de leur probabilité de défaut de paiement, afin d'aider notre institution à prendre des décisions éclairées en matière de prêts.

Pour atteindre cet objectif, nous avons suivi une approche méthodique en plusieurs étapes. Tout d'abord, nous avons exploré et prétraité les données, en effectuant des analyses exploratoires pour comprendre la distribution des variables et en traitant les valeurs manquantes ainsi que les variables catégorielles.

Par la suite, nous avons divisé notre ensemble de données en ensembles d'entraînement et de test, puis nous avons sélectionné et entraîné plusieurs modèles de machine learning, notamment la régression logistique, le SVM, XGBoost, et le RandomForest. Nous avons utilisé des techniques d'optimisation de modèle telles que la validation croisée et la recherche sur grille pour optimiser les hyperparamètres de chaque modèle.

Enfin, nous avons évalué les performances de chaque modèle en utilisant plusieurs métriques, notamment l'AUC-ROC, la précision, le rappel et le score métier, en tenant compte des considérations commerciales spécifiques. Nous avons identifié le meilleur modèle en fonction de ces métriques et avons conclu que le modèle XGBoost était le plus performant pour notre problème de prédiction de défaut de paiement.
Le feature importance nous a aussi permis de savoir quelles étaient les variables avec le plus d’impact sur notre modèle telles que EXT_SOURCE_2 , EXT_SOURCE_3 et AMT_CREDIT.

En résumé, ce projet a abouti à la construction réussie d'un modèle de scoring fiable pour évaluer le risque de défaut de paiement des clients. Ce modèle pourra être utilisé par notre institution financière pour prendre des décisions éclairées en matière de prêts, améliorant ainsi la gestion des risques et contribuant à la durabilité financière de l'entreprise. Cependant, il est important de noter que ce modèle devra être continuellement surveillé et mis à jour pour s'adapter aux changements dans les données et dans l'environnement commercial.