book

June 15, 2025

```
<h1 style="font-size:48px; font-weight:bold;"> Credit Scoring en Afrique de l'Ouest</h1>
<h2 style="font-weight:normal;">Modélisation prédictive du risque de crédit</h2>
<p style="font-size:18px; margin-top:40px;">Par Kodjo Jean DEGBEVI</p>
<p style="font-size:14px;">Juin 2025</p>
```

Table des matières

Nom de la variable	Type	Description
ID	Identifiant	Identifiant unique de l'enregistrement
customer_id	Identifiant	Identifiant unique du client emprunteur
country_id	Identifiant	Identifiant du pays (permet de croiser avec une table pays, si dispo)
tbl_loan_id	Identifiant	Identifiant du prêt dans la table principale des prêts
lender_id	Identifiant	Identifiant du prêteur associé au prêt
loan_type	Catégoriel	Type de prêt (ex : "business", "personal", etc.)
Total_Amount	Numérique	Montant total du prêt demandé
Total_Amount_to_Repay	Numérique	Montant total que l'emprunteur doit rembourser (avec intérêts/frais)
disbursement_date	Date	Date de décaissement du prêt
due_date	Date	Date d'échéance du remboursement
duration	Numérique	Durée du prêt (en jours ou mois)
New_versus_Repeat	Catégoriel	Statut du client : nouveau ou récurrent
Amount_Funded_By_Lender	Numérique	Montant réellement financé par le prêteur
Lender_portion_Funded	Numérique	Part du montant total financé par ce prêteur
Lender_portion_to_be_repaid	Numérique	Part du remboursement revenant au prêteur
target	Binaire	Variable cible : défaut (1) ou non défaut (0)

[164]:
```
import sys
import os
import importlib
import time
import pandas as pd
```

```
from pandas.api.types import CategoricalDtype
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import missingno as mns
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE

from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier

from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score, precision_score, recall_score, f1_score, roc_curve, auc
from sklearn.metrics import make_scorer, fbeta_score

from sklearn.ensemble import StackingClassifier
#sklearn.pipeline import Pipeline
from imblearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import StratifiedKFold, cross_val_score

import shap
from tqdm import tqdm
import joblib
```

1 Collecte et premières observations

[165]:
```
data = pd.read_csv("../Data/Train.csv")
df = data.copy()
df.sample(10)
```

[165]:

	ID	customer_id	country_id	tbl_loan_id	lender_id	\
22623	ID_242468217708267278	242468	Kenya	217708	267278	
33211	ID_267934229564267278	267934	Kenya	229564	267278	
12838	ID_267181214591267278	267181	Kenya	214591	267278	
29054	ID_310219371013267278	310219	Kenya	371013	267278	
43015	ID_259583290234267278	259583	Kenya	290234	267278	
52187	ID_244559228407267278	244559	Kenya	228407	267278	
66415	ID_256615255176267278	256615	Kenya	255176	267278	
68473	ID_261535288775267278	261535	Kenya	288775	267278	
50527	ID_256353229141267278	256353	Kenya	229141	267278	
49300	ID_259111287864267278	259111	Kenya	287864	267278	

```
      loan_type  Total_Amount  Total_Amount_to_Repay disbursement_date  \
22623    Type_1        4570.0                 4734.0        2022-07-20
33211    Type_1        2689.0                 2747.0        2022-08-01
12838    Type_1        4765.0                 4765.0        2022-07-13
29054    Type_7        4240.0                 4389.0        2024-09-13
43015    Type_1        3639.0                 3665.0        2022-10-29
52187    Type_1        5718.0                 5894.0        2022-07-30
66415    Type_1         250.0                  254.0        2022-09-09
68473    Type_1       37490.0                38631.0        2022-10-26
50527    Type_1       12992.0                13388.0        2022-08-01
49300    Type_1         730.0                  736.0        2022-10-25

         due_date  duration New_versus_Repeat  Amount_Funded_By_Lender  \
22623  2022-07-27         7       Repeat Loan                   1371.0
33211  2022-08-08         7       Repeat Loan                    806.7
12838  2022-07-20         7       Repeat Loan                   1429.5
29054  2024-09-20         7       Repeat Loan                    848.0
43015  2022-11-05         7       Repeat Loan                   1091.7
52187  2022-08-06         7       Repeat Loan                   1715.4
66415  2022-09-16         7       Repeat Loan                     75.0
68473  2022-11-02         7       Repeat Loan                    602.6
50527  2022-08-08         7       Repeat Loan                   3897.6
49300  2022-11-01         7       Repeat Loan                    219.0

       Lender_portion_Funded  Lender_portion_to_be_repaid  target
22623               0.300000                       1420.0       0
33211               0.300000                        824.0       0
12838               0.300000                       1430.0       0
29054               0.200000                        878.0       0
43015               0.300000                       1100.0       0
52187               0.300000                       1768.0       0
66415               0.300000                         76.0       0
68473               0.016074                        621.0       0
50527               0.300000                       4016.0       0
49300               0.300000                        221.0       0
```

[166]:
```
df.shape
```

[166]:
```
(68654, 16)
```

[167]:
```
df.columns
```

[167]:
```
Index(['ID', 'customer_id', 'country_id', 'tbl_loan_id', 'lender_id',
       'loan_type', 'Total_Amount', 'Total_Amount_to_Repay',
       'disbursement_date', 'due_date', 'duration', 'New_versus_Repeat',
       'Amount_Funded_By_Lender', 'Lender_portion_Funded',
```

```
       'Lender_portion_to_be_repaid', 'target'],
      dtype='object')
```

[168]:
```
df.describe()
```

[168]:

	customer_id	tbl_loan_id	lender_id	Total_Amount
count	68654.000000	68654.000000	68654.000000	6.865400e+04
mean	254390.256780	263056.266248	266420.528462	1.483683e+04
std	26642.719918	39486.661487	3590.999004	1.416499e+05
min	145.000000	101323.000000	245684.000000	2.000000e+00
25%	248945.750000	233942.250000	267278.000000	2.295000e+03
50%	255361.000000	260305.500000	267278.000000	5.249000e+03
75%	262269.250000	286962.750000	267278.000000	1.145000e+04
max	312737.000000	375320.000000	267278.000000	2.300000e+07

	Total_Amount_to_Repay	duration	Amount_Funded_By_Lender
count	6.865400e+04	68654.000000	6.865400e+04
mean	1.563993e+04	8.544586	2.545663e+03
std	1.650784e+05	13.343145	1.192272e+04
min	0.000000e+00	1.000000	0.000000e+00
25%	2.329000e+03	7.000000	2.340000e+02
50%	5.325000e+03	7.000000	9.150000e+02
75%	1.165000e+04	7.000000	2.272650e+03
max	2.541500e+07	1096.000000	1.600000e+06

	Lender_portion_Funded	Lender_portion_to_be_repaid	target
count	68654.000000	6.865400e+04	68654.000000
mean	0.218679	2.652621e+03	0.018324
std	0.129832	1.338006e+04	0.134120
min	0.000000	0.000000e+00	0.000000
25%	0.118712	2.390000e+02	0.000000
50%	0.300000	9.340000e+02	0.000000
75%	0.300000	2.317000e+03	0.000000
max	1.168119	1.821338e+06	1.000000

[169]:
```
df.describe(include="object")
```

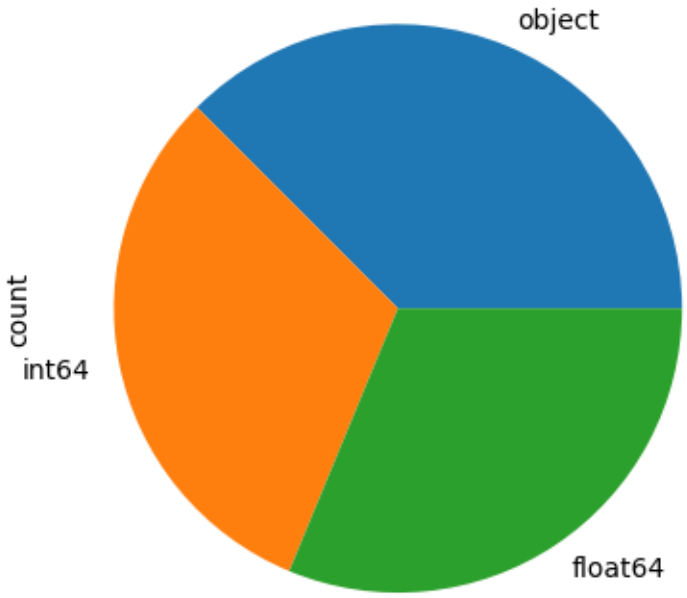
[169]:

	ID	country_id	loan_type	disbursement_date
count	68654	68654	68654	68654
unique	68654	1	22	768
top	ID_249117268933267278	Kenya	Type_1	2022-07-16
freq	1	68654	61723	938

	due_date	New_versus_Repeat
count	68654	68654
unique	893	2
top	2022-07-23	Repeat Loan

```
freq            940             68087
```

[170]:
```
df.dtypes.value_counts().plot.pie()
plt.show()
```



[171]:
```
df.dtypes
```

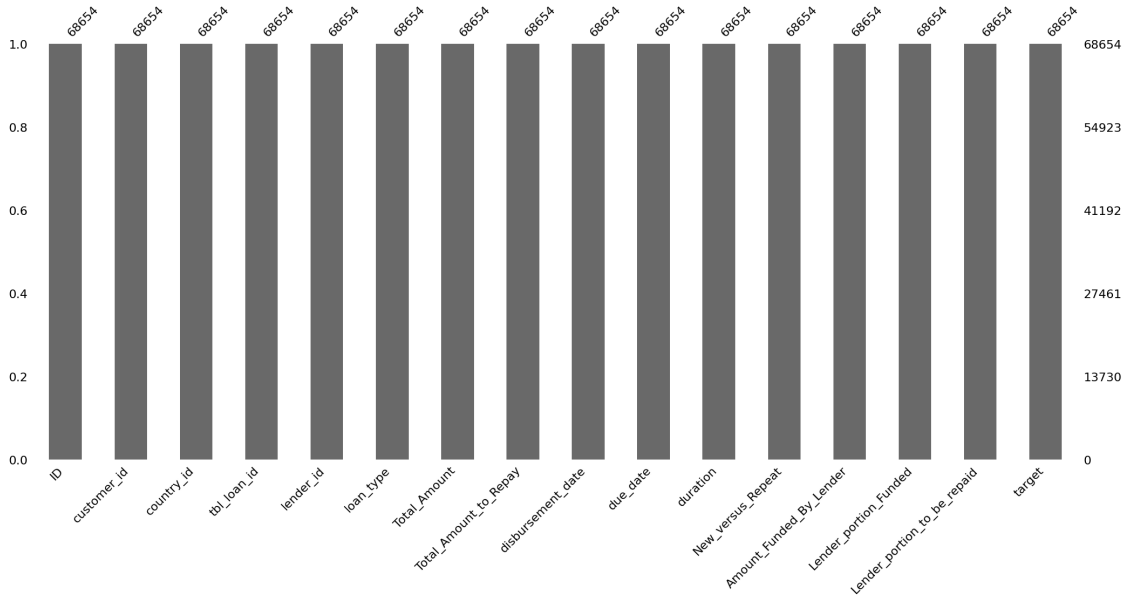
[171]:
```
ID                           object
customer_id                   int64
country_id                   object
tbl_loan_id                   int64
lender_id                     int64
loan_type                    object
Total_Amount                float64
Total_Amount_to_Repay       float64
disbursement_date            object
due_date                     object
duration                      int64
New_versus_Repeat            object
Amount_Funded_By_Lender     float64
Lender_portion_Funded       float64
Lender_portion_to_be_repaid float64
target                        int64
```

```
dtype: object
```

2 Analyse de forme

2.1 Missing data

[172]:
```
mns.bar(df)
plt.show()
```



Pas de valeurs manquantes.

2.2 S'assurer que les colonnes de date sont en format datetime

[173]:
```
df['disbursement_date'] = pd.to_datetime(df['disbursement_date'],␣
↪errors='coerce')
df['due_date'] = pd.to_datetime(df['due_date'], errors='coerce')
```

2.3 Créer de variables dérivées

Pour mieux capturer les caractéristiques temporelles

[174]:
```
df['disbursement_month'] = df['disbursement_date'].dt.month
df['disbursement_dayofweek'] = df['disbursement_date'].dt.dayofweek
```

S'assurer que la variable duration est fiable :

[175]:
```
((df['due_date'] - df['disbursement_date']).dt.days - df['duration']).describe()
```

[175]:
```
count    68654.0
mean         0.0
std          0.0
min          0.0
25%          0.0
50%          0.0
75%          0.0
max          0.0
dtype: float64
```

L'écart est nul donc pas besoin de dériver une nouvelle variable.

On peut garder duration

Capturons la charge relative au prêt, le taux d'effort

[176]:
```
df['repayment_ratio'] = df['Total_Amount_to_Repay'].divide(df['Total_Amount'])
```

2.4 Encoder les colonnes catégorielles

[177]:
```
Hot_encoder = OneHotEncoder(drop='first', handle_unknown='ignore')
Hot_encoder.fit(df[['New_versus_Repeat']])
encoded_var= pd.DataFrame(Hot_encoder.transform(df[['New_versus_Repeat']]).
 ↪toarray(), columns=Hot_encoder.get_feature_names_out(['New_versus_Repeat']))
df = pd.concat([df, encoded_var], axis=1)

loan_type_dtype = CategoricalDtype(categories=df['loan_type'].unique().
 ↪tolist(), ordered=False)
df['loan_type_encoded'] = df['loan_type'].astype(loan_type_dtype).cat.codes

pd.set_option('display.max_columns', 25)
df.sample(5)
```

[177]:

	ID	customer_id	country_id	tbl_loan_id	lender_id
68636	ID_252111263516267278	252111	Kenya	263516	267278
45091	ID_248188257971267278	248188	Kenya	257971	267278
9946	ID_261827299294267278	261827	Kenya	299294	267278
66272	ID_259757239476267278	259757	Kenya	239476	267278
47658	ID_256419273764267278	256419	Kenya	273764	267278

	loan_type	Total_Amount	Total_Amount_to_Repay	disbursement_date
68636	Type_1	4599.0	4599.0	2022-09-20
45091	Type_1	11041.0	11243.0	2022-09-12
9946	Type_1	3346.0	3346.0	2022-11-15
66272	Type_1	29694.0	29694.0	2022-08-18
47658	Type_1	2240.0	2256.0	2022-10-03

	due_date	duration	New_versus_Repeat	Amount_Funded_By_Lender

```
68636 2022-09-27         7       Repeat Loan                    0.00
45091 2022-09-19         7       Repeat Loan                 3312.30
9946  2022-11-22         7       Repeat Loan                    0.00
66272 2022-08-25         7       Repeat Loan                 3293.14
47658 2022-10-10         7       Repeat Loan                    0.00

       Lender_portion_Funded  Lender_portion_to_be_repaid  target  \
68636               0.000000                          0.0       0
45091               0.300000                       3373.0       0
9946                0.000000                          0.0       0
66272               0.110903                       3293.0       0
47658               0.000000                          0.0       0

       disbursement_month  disbursement_dayofweek  repayment_ratio  \
68636                   9                       1         1.000000
45091                   9                       0         1.018295
9946                   11                       1         1.000000
66272                   8                       3         1.000000
47658                  10                       0         1.007143

       New_versus_Repeat_Repeat Loan  loan_type_encoded
68636                            1.0                  0
45091                            1.0                  0
9946                             1.0                  0
66272                            1.0                  0
47658                            1.0                  0
```

2.5 Supprimer les colonnes inutiles

[178]:
```
cols_to_drop = [
    'ID', 'customer_id', 'lender_id', 'tbl_loan_id', 'disbursement_date',␣
 ↪'due_date',
    'loan_type', 'New_versus_Repeat', 'country_id'
]
df.drop(columns=cols_to_drop, inplace=True)
print(df.shape)
df.dtypes
```

```
(68654, 12)
```

[178]:
```
Total_Amount                   float64
Total_Amount_to_Repay          float64
duration                         int64
Amount_Funded_By_Lender        float64
Lender_portion_Funded          float64
Lender_portion_to_be_repaid    float64
target                           int64
```

```
disbursement_month                int32
disbursement_dayofweek            int32
repayment_ratio                 float64
New_versus_Repeat_Repeat Loan   float64
loan_type_encoded                  int8
dtype: object
```

3 Analyse de fond

3.1 Séparation des featrues de la cible

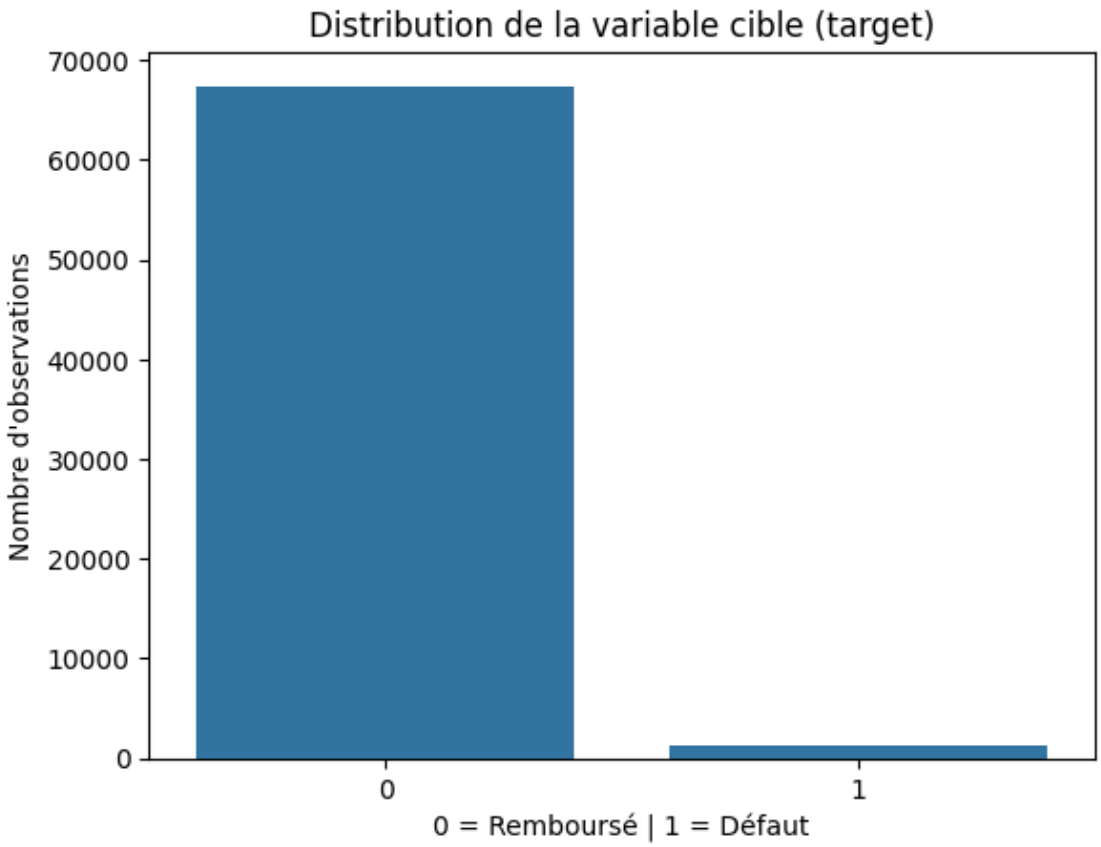
[19]:
```
X = df.drop(columns='target')
y = df[['target']]
```

3.2 Distribution de la cible

[20]:
```
y.value_counts(normalize=True) * 100
```

[20]:
```
target
0         98.167623
1          1.832377
Name: proportion, dtype: float64
```

[21]:
```
sns.countplot(y, x='target')
plt.title("Distribution de la variable cible (target)")
plt.xlabel("0 = Remboursé | 1 = Défaut")
plt.ylabel("Nombre d'observations")
plt.show()
```



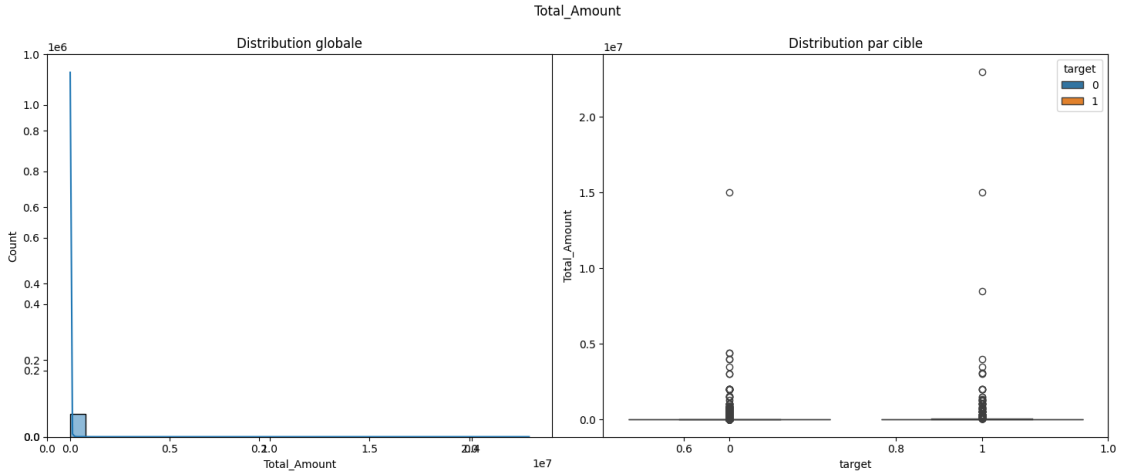
Cible très désiquilibrée. On aura peut-être besoin d'une augmentation smote

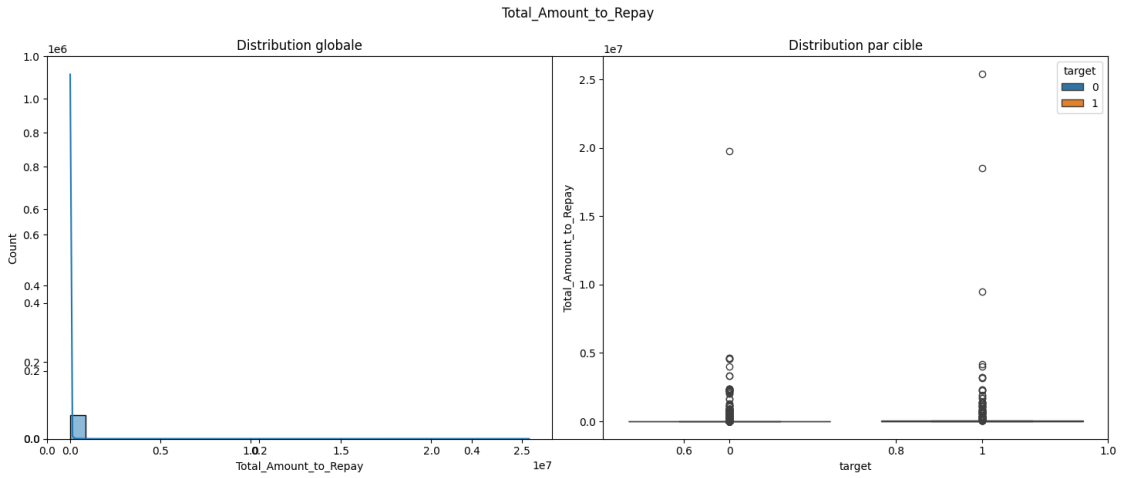
3.3 Analyse des features

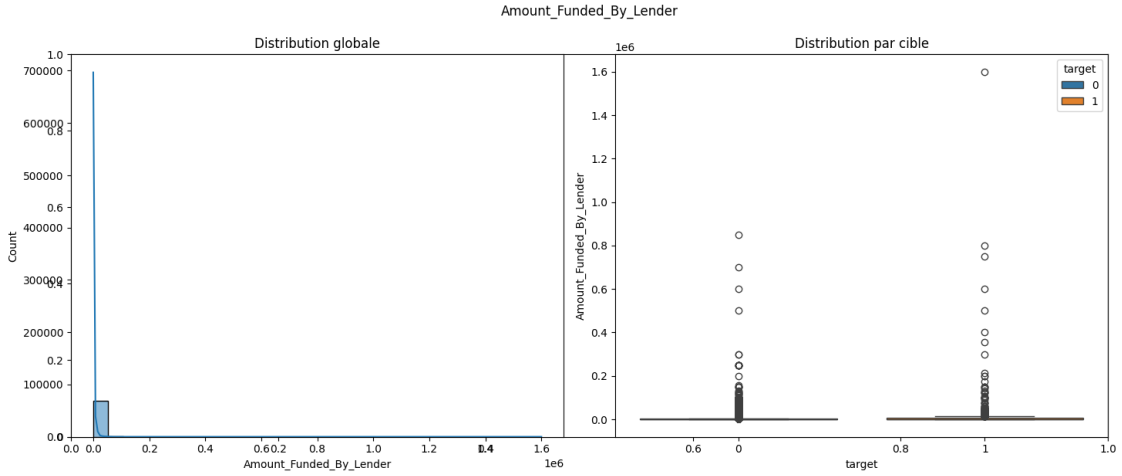
3.3.1 Distribution de certaines features

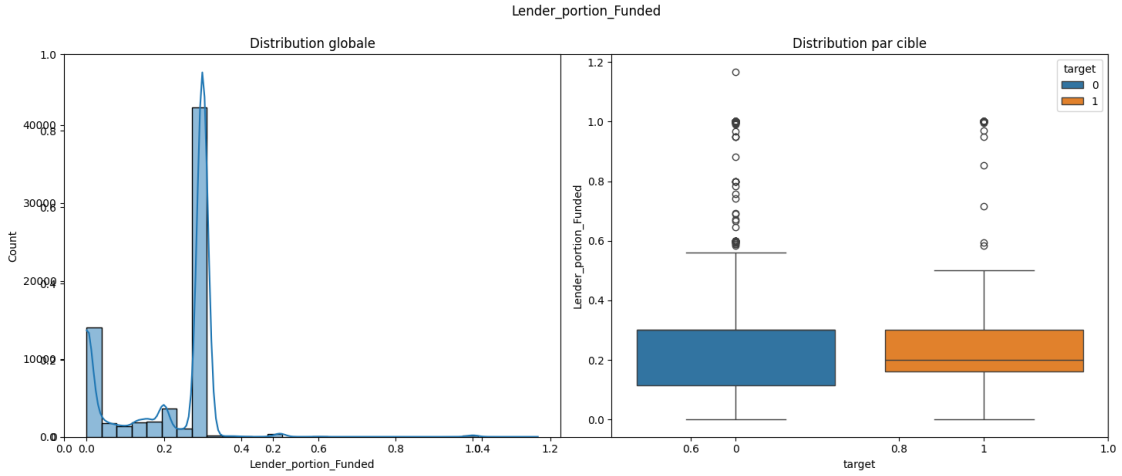
[22]:
```
for col in X.select_dtypes(include=['float64']).columns:
    plt.figure(figsize=(15, 7))
    plt.title(f'{col}\n\n', loc='center')

    # Histogramme global
    plt.subplot(1, 2, 1)
    sns.histplot(X[col], kde=True, bins=30)
    plt.title(f'Distribution globale')

    # Histogramme par classe (target = 0 ou 1)
    plt.subplot(1, 2, 2)
    #sns.histplot(data=X.assign(target=y), x=col, hue='target', kde=True,␣
↪bins=30, multiple='stack')
    sns.boxplot(data=X.assign(target=y), x='target', y=col, hue='target')
    plt.title(f'\nDistribution par cible')
```

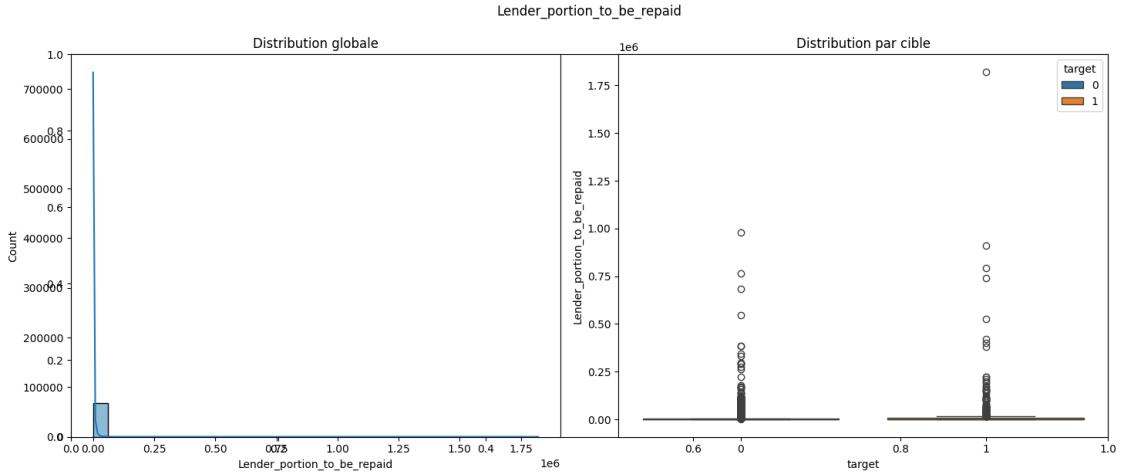
```
plt.tight_layout()
plt.show()
```

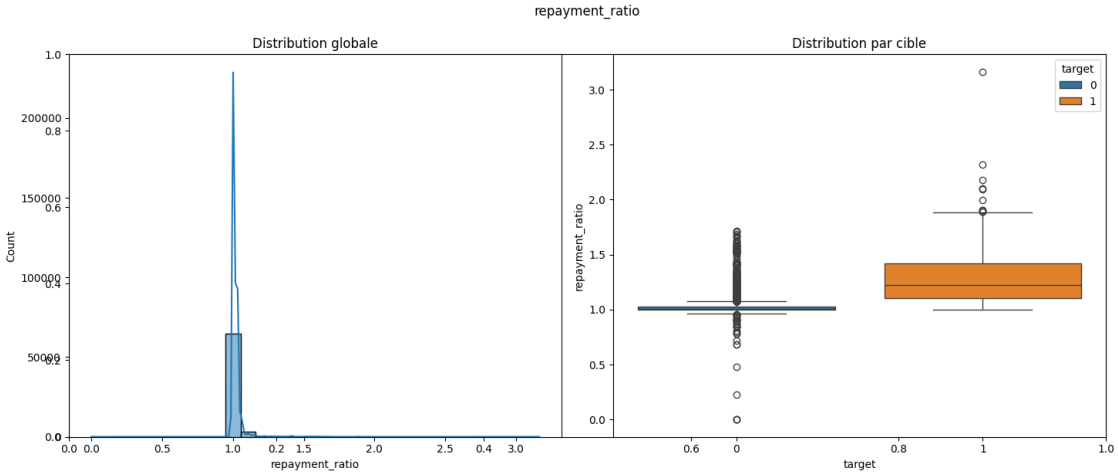


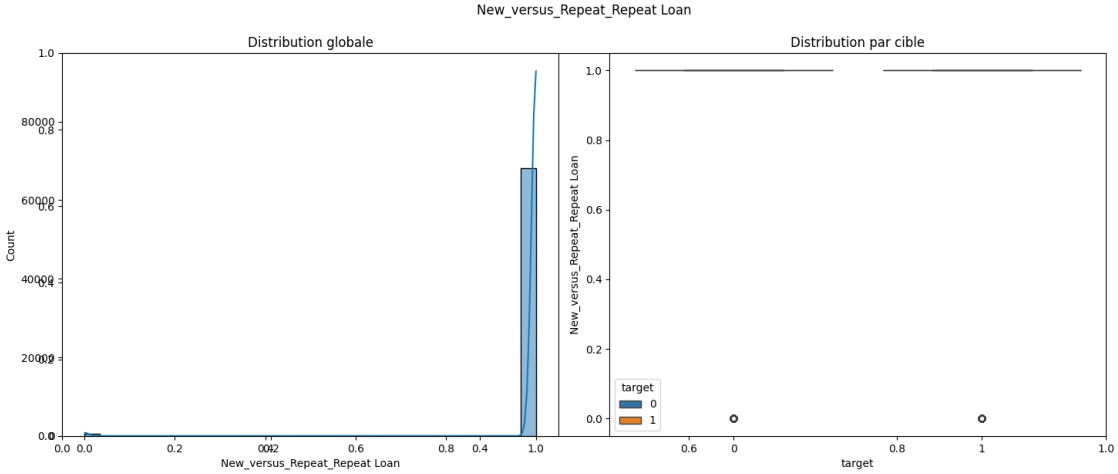



Amount_Funded_By_Lender
Distribution globale
Distribution par cible
Count
Amount_Funded_By_Lender
target
0
1
1e6


Lender_portion_Funded
Distribution globale
Distribution par cible
Count
Lender_portion_Funded
target
0
1


Lender_portion_to_be_repaid
Distribution globale
Distribution par cible
Count
Lender_portion_to_be_repaid
target
0
1
1e6

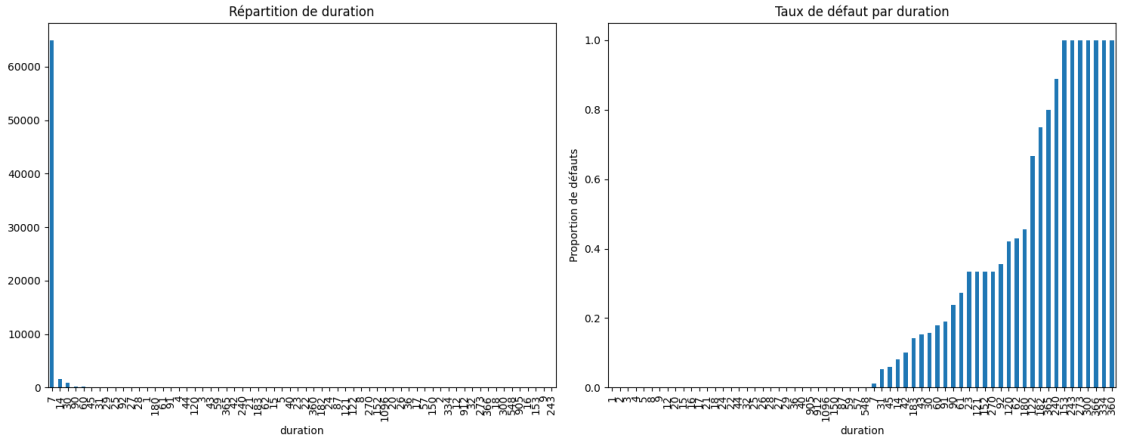


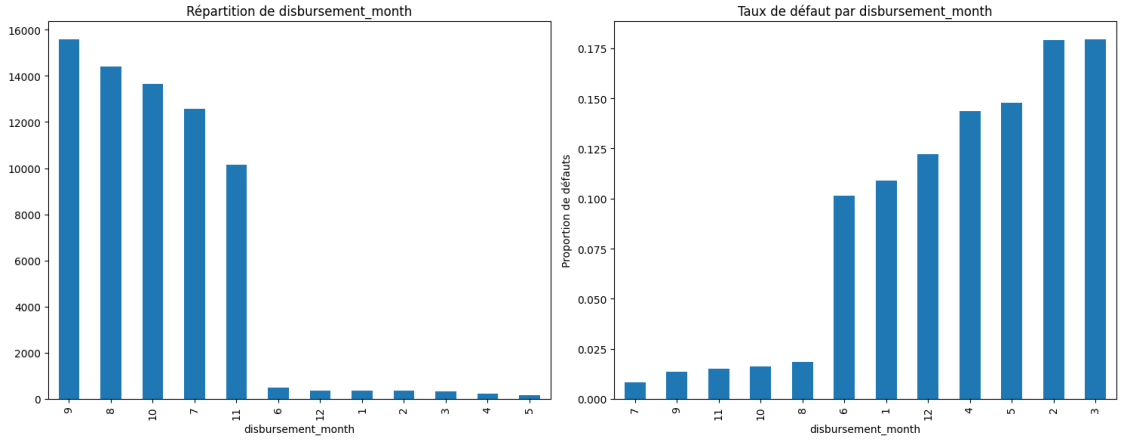


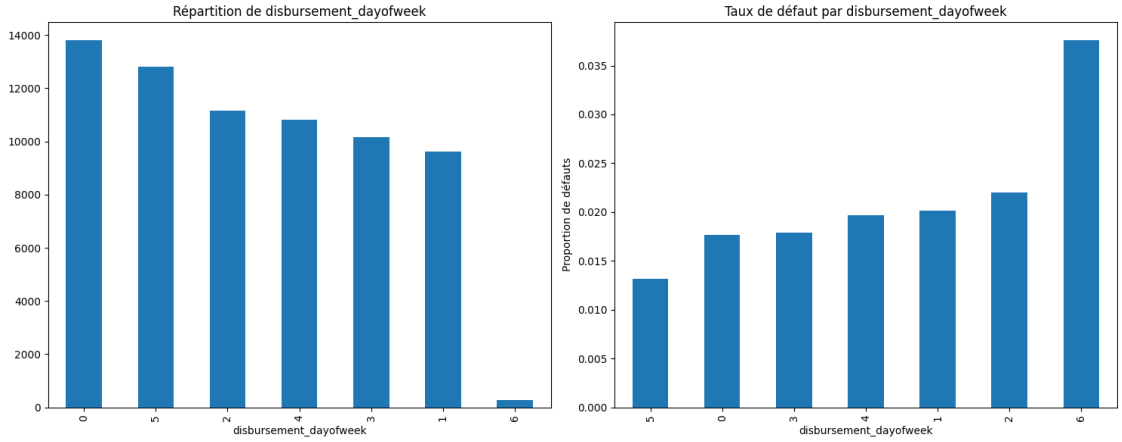
- **Les features '*Total_Amount*','*Total_Amount_to_Repay*' et '*Amount_Funded_By_Lender*' suivent pratiquement la même distribution et ne dégagent pas de tendances particulières.**
  - **Par contre, elles présentent beaucoup d'outliers. Nous continuerons avec elles comme cela pour le moment.**
- **'*Lender_portion_Funded*' et *'Lender_portion_to_be_repaid'* aussi présentent d'importants outliers et montre visuellement pas de tendances dans les données. Nous les gardons aussi telles quelles pour le moment.**
- **`Par contre avec *'repayment_ratio'* nous voyons que les prêts à défaut ont des valeurs élevées.`**

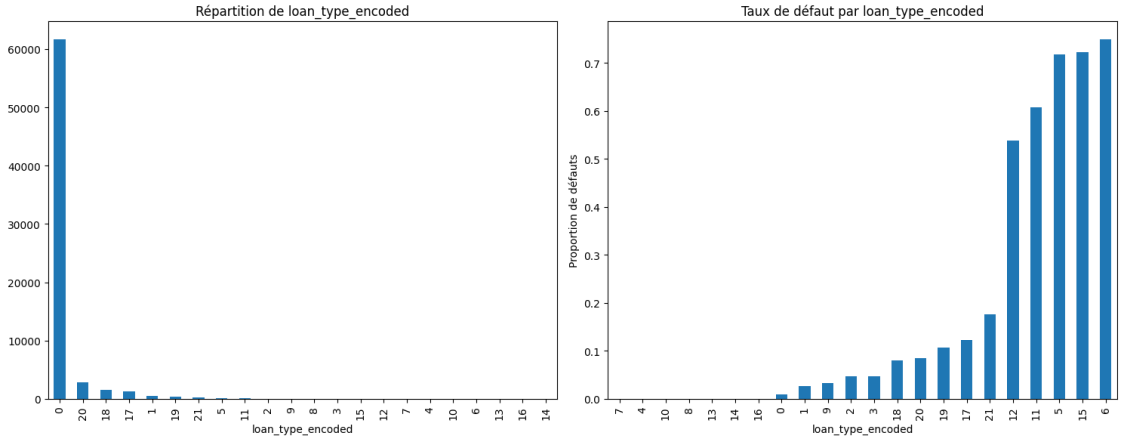
3.3.2 Observation des proportions de défaut par features entiers

[23]:
```
for feature in X.select_dtypes(include=['int8', 'int16', 'int32', 'int64']).
 ↪columns:
    plt.figure(figsize=(15, 6))
    plt.subplot(1, 2, 1)
    X[feature].value_counts().plot(kind='bar')
    plt.title(f"Répartition de {feature}")

    plt.subplot(1, 2, 2)
    X.join(y).groupby(feature)['target'].mean().sort_values().plot(kind='bar')
    plt.title(f"Taux de défaut par {feature}")
    plt.ylabel("Proportion de défauts")

    plt.tight_layout()
    plt.show()
```







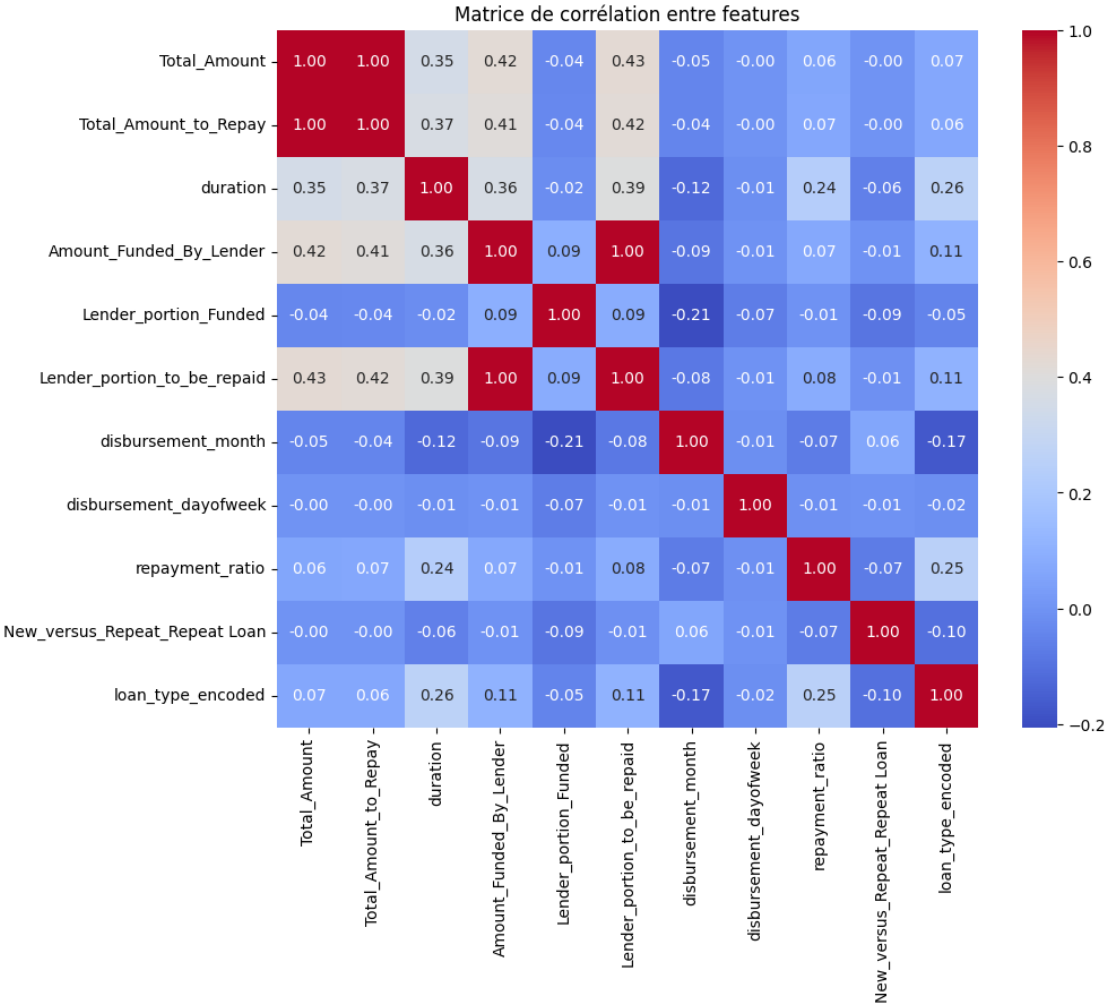


- **Les prêts ont presque tous une 'duration' de 7.**
  - Cette 'duration' est aussi les plus remboursé avec un taux de défaut d'environ $10^{-2}\%$
- **Les prêts sont plus souvent débourser du mois de Juillet à Novembre.**
  - *Et les prêts de cette période connaissent moins de défaut de payement que durant le reste de l'année.*
    - On peut remarquer qu'étant donné qu'il y a moins de déboursement le reste de l'année, c'est normal que la proportion de défaut soit aussi élevée en ces moment.
- **Les prêts sont débourser presque autant tous les jours de la semaine avec une hausse les lundi et samedi (*premier et dernier jour ouvrables*) sauf le dimanche qui n'en connait pratiquement pas (*le dimanche est généralement férié*).**
  - *Et tous les jours connaissent presque autant de défauts de payement sauf le dimanche dont environ 3.5% des prêts déboursés connaissent un défaut de payement. Les déboursement de lundi et de samedi connaisse moins de défaut.*
- **Les prêts sont presque tous de type 'Type_0' et ce type de prêts est très souvnet remboursé normalement. Contrairement aux types 'Type12', 'Type11', 'Type5', 'Type15', 'Type6' Les types 'Type20', 'Type18', 'Type17' sont aussi sollicité mais très peu et connaissent un taux de défaut pas très élevé (environ 15%)**

Nous verons si cela nous servira dans la suite !

3.4 Corrélations entre features

[24]:
```
corr_matrix = X.join(y).corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix.drop('target', axis=0).drop('target', axis=1),␣
 ↪annot=True, fmt=".2f", cmap='coolwarm')
plt.title("Matrice de corrélation entre features")
plt.show()
```



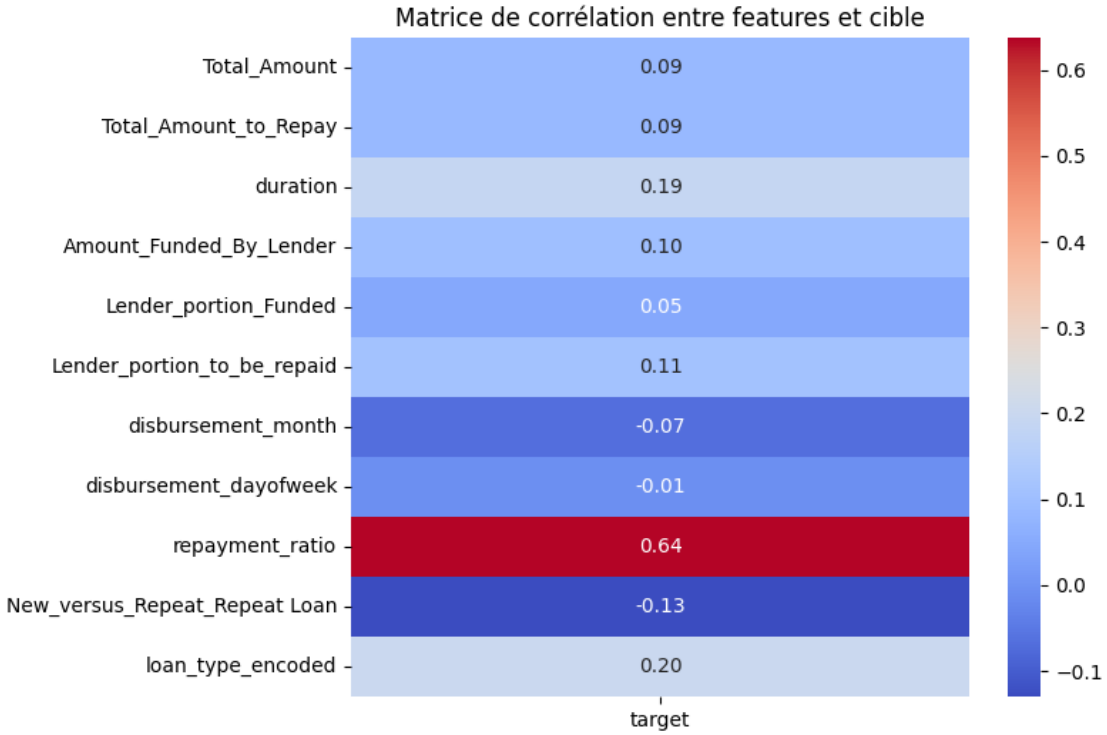
Nous remarquons que les features '*Total_Amount_to_Repay*' et '*Total_Amount*' nous donne exactement la même information, à la même échelle.

Pareil pour *'Lender_portion_to_be_repaid'* et *'Amount_Funded_By_Lender'*

Nous éliminerons en une dans chaque paire dans la suite.

3.5 Corrélations entre features et target

[25]:
```
plt.figure(figsize=(7, 6))
plt.title("Matrice de corrélation entre features et cible")
sns.heatmap(corr_matrix[['target']].drop('target', axis=0), annot=True, fmt=".
↪2f", cmap='coolwarm')
plt.show()
```



Supprimons déja les features qui sont corrélées à moins de 0.1 (en arrondi) avec la cible.

[26]:
```
def corr_select(column, threshold):
    return abs(round(corr_matrix[['target']].loc[column].item(), 1)) < threshold

X_new = (X.copy()).drop([col for col in X.columns if corr_select(col, 0.1)],␣
↪axis=1)
X_new.columns.to_list()
```

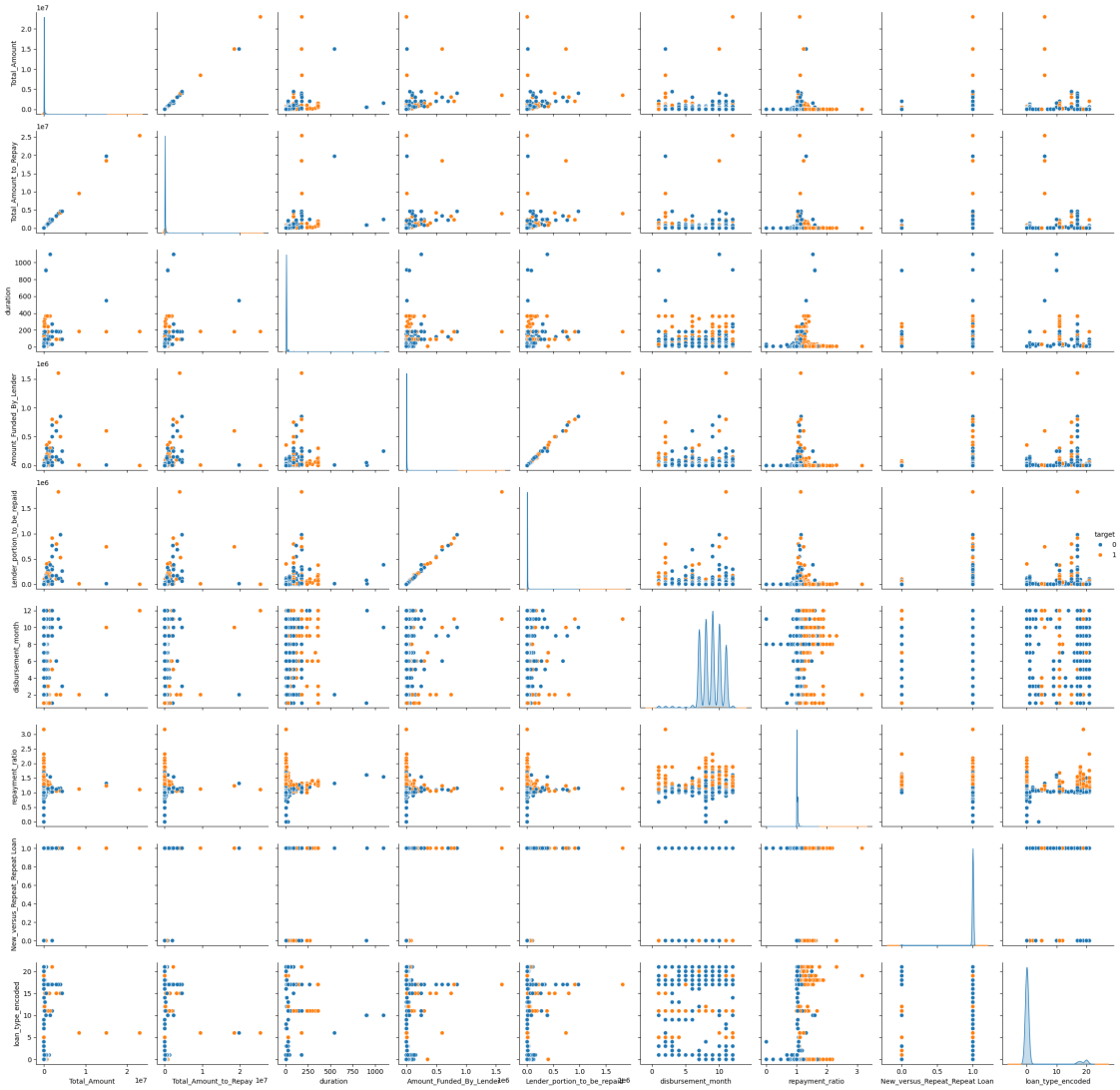
[26]:
```
['Total_Amount',
 'Total_Amount_to_Repay',
 'duration',
 'Amount_Funded_By_Lender',
 'Lender_portion_to_be_repaid',
 'disbursement_month',
```

```
'repayment_ratio',
'New_versus_Repeat_Repeat Loan',
'loan_type_encoded']
```

3.6 Analyse multivariée

3.6.1 Pairplots

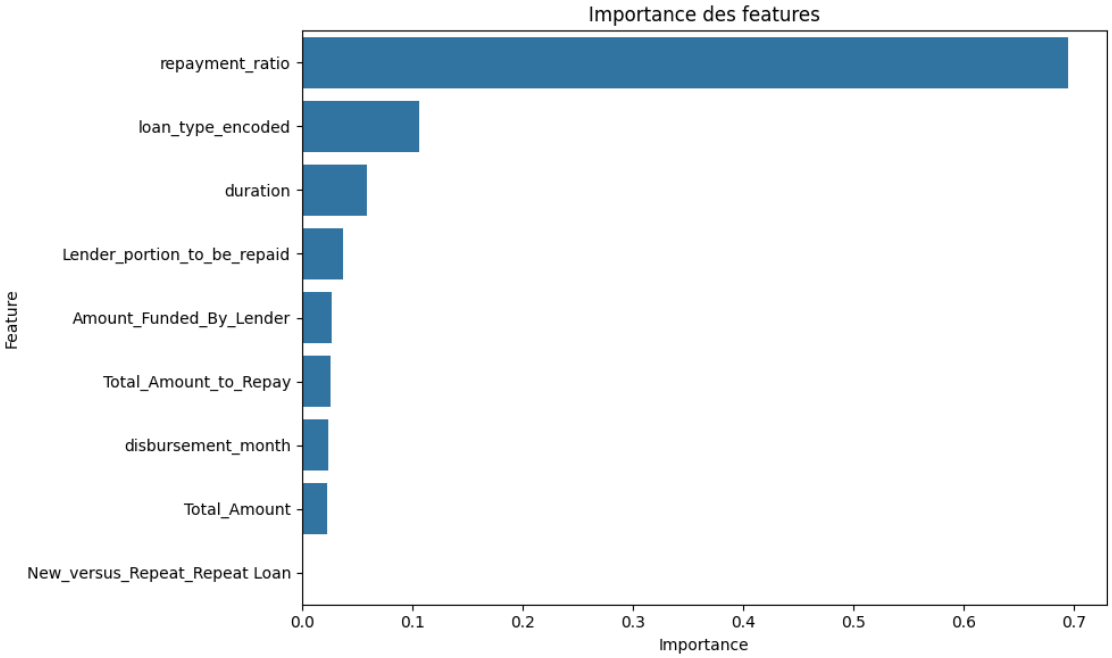
[26]:
```
# Pairplot avec queues features
sns.pairplot(X.drop([col for col in X.columns if corr_select(col, 0.1)],
                    axis=1).assign(target=y),
                    hue="target",
                    diag_kind="kde")
plt.tight_layout()
plt.show()
```



- **Déja, nous voyons que 'repayment_ratio' a une séparation nette entre classes.**
  - Feature clairement la plus importante et donc plus informative.
- **Quasi colinéarité entre 'Amount_Funded_By_Lunder' et 'Lender_portion_to_be_repaid' et aussi 'Total_Amount' et 'Total_Amount_to_repaid'**
  - Nous pouvons en éliminer un de chaque groupe sans vraiment perdre d'informations.
- **Pas de séparation (très peu d'information) pour 'New_versus_Repeat_encoded', 'loan_type_encoded', 'disbursement_month', 'Total_Amount', 'Total_Amount_to_repaid'**

3.7 Sélection des features d'entrainement

[27]:
```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(random_state=42, n_estimators=100, max_depth=10,␣
 ↪class_weight='balanced', n_jobs=-1)
model.fit(X_new, y.values.ravel())
importances = model.feature_importances_
feature_importances = pd.DataFrame({
    'Feature': X_new.columns,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_importances)
plt.title("Importance des features")
plt.tight_layout()
plt.show()
```



rapayment_ratio est de loin, la plus informative avec une importance d'environ 0.7. Les autres features sont moins importantes mais des combinaisons peuvent fournir de bonne informations.

Au vue de toutes les analyses précédentes et des résultats de ce test d'importance de features, ***pour ne aller plus en profondeur, nous continuerons avec les 6 features les plus importantes.***

[28]:
```
#features_selected = feature_importances[feature_importances['Importance'] > 0.
↪05]['Feature'].tolist()
X_selected = X_new[feature_importances['Feature'].head(6).tolist()]
X_selected.columns.to_list()
```

[28]:
```
['repayment_ratio',
 'loan_type_encoded',
 'duration',
 'Lender_portion_to_be_repaid',
 'Amount_Funded_By_Lender',
 'Total_Amount_to_Repay']
```

4 Préparation des données pour l'entrainement

4.1 Spliting

[29]:
```
# Séparation des données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.
↪2, random_state=42, stratify=y)
```

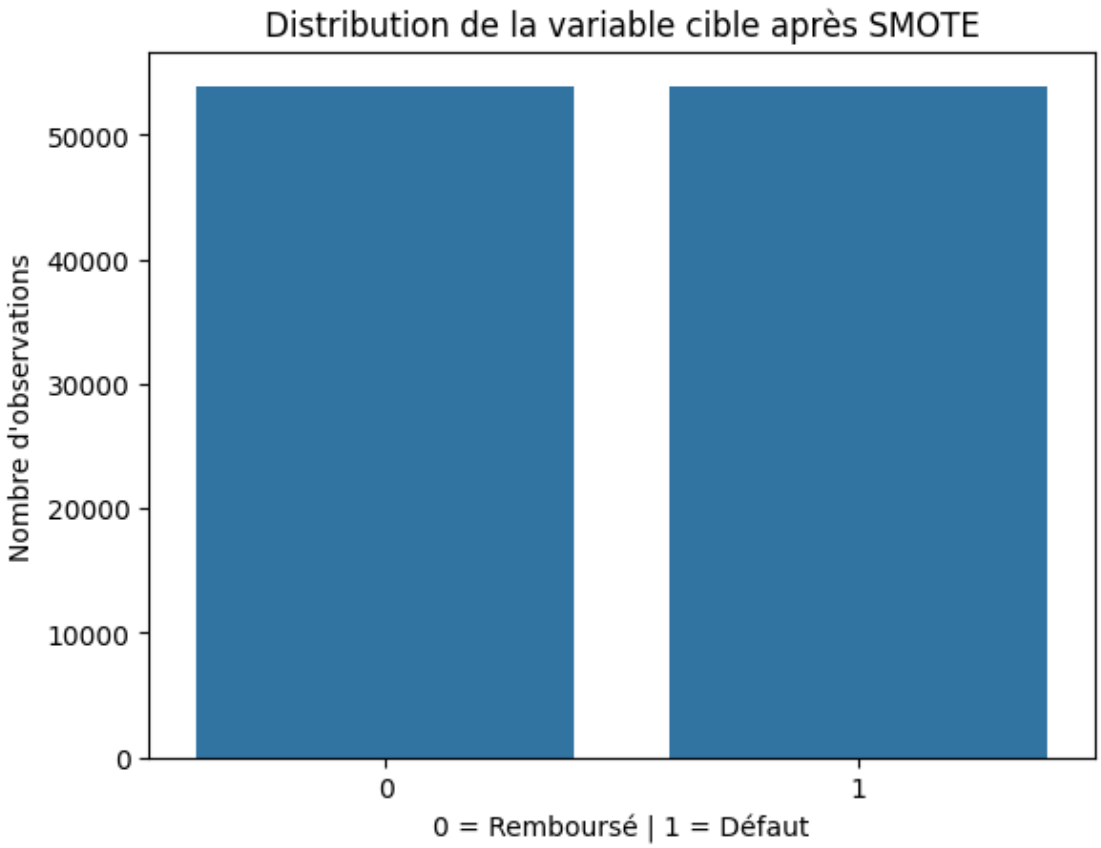
4.2 Normalisation

[30]:
```
# Pour une regression logistic par exemple
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled, X_test_scaled = scaler.transform(X_train), scaler.
↪transform(X_test)
```

4.3 Gestion du deséquilibre de la cible

[31]:
```
smote1, smote2 = SMOTE(random_state=42), SMOTE(random_state=42)
X_train_res, y_train_res = smote1.fit_resample(X_train, y_train)
X_train_scaled_res, y_train_scaled_res = smote2.fit_resample(X_train_scaled,␣
↪y_train)

sns.countplot(y_train_res, x='target')
plt.title("Distribution de la variable cible après SMOTE")
```

```
plt.xlabel("0 = Remboursé | 1 = Défaut")
plt.ylabel("Nombre d'observations")
plt.show()
```



5 Modèles, entrainement et évaluation

Nous choisissons de comparer RandomForest, LinearRegression et XGBoost

Modèle	Avantages	Inconvénients	Normalisation ?
LogisticRegression	Simple, rapide, interprétable	Pas performant si relations non-linéaires	Obligatoire
RandomForest	Gère non-linéarités, robustesse, importance des features	Moins bon sur petits jeux très déséquilibrés	Inutile
XGBoost	Très performant, gère bien les classes déséquilibrées	Moins interprétable, plus lent	Inutile

5.1 Models et recherhce d'hyper-paramètres

5.1.1 Random Forest

[ ]:
```
rf = RandomForestClassifier(random_state=42)

param_grid_rf = {
    'n_estimators': [100, 300, 500],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

grid_rf = GridSearchCV(rf, param_grid_rf, scoring=make_scorer(fbeta_score,␣
 ↪beta=1.5), cv=3, n_jobs=-1, verbose=True)
grid_rf.fit(X_train_res, y_train_res.values.ravel())

print("Best params RF:", grid_rf.best_params_)
print("Best FBetaScore:", grid_rf.best_score_)
```

5.1.2 Logistic Regression

[ ]:
```
lr = LogisticRegression(max_iter=1000, random_state=2)

param_grid_lr = {
    'solver': ['saga', 'liblinear'],
    'penalty': ['l2'],
    'C': [0.01, 0.1, 1, 10]
}

grid_lr = GridSearchCV(lr, param_grid_lr, scoring=make_scorer(fbeta_score,␣
 ↪beta=1.5), cv=3, n_jobs=-1, verbose= True)
grid_lr.fit(X_train_scaled_res, y_train_res.values.ravel())

print("Best params LR:", grid_lr.best_params_)
print("Best FBetaScore:", grid_lr.best_score_)
```

5.1.3 XGBoost

[ ]:
```
xgb = XGBClassifier(eval_metric='logloss', random_state=42)

param_grid_xgb = {
    'max_depth': [3, 4, 6],
    'learning_rate': [0.01, 0.05, 0.1],
    'subsample': [0.8, 1],
    'colsample_bytree': [0.8, 1],
    'n_estimators': [200, 500]
}
```

```
grid_xgb = GridSearchCV(xgb, param_grid_xgb, scoring=make_scorer(fbeta_score,␣
↪beta=1.5), cv=3, n_jobs=-1, verbose=True)
grid_xgb.fit(X_train_res, y_train_res.values.ravel())

print("Best params XGBoost:", grid_xgb.best_params_)
print("Best FBetaScore:", grid_xgb.best_score_)
```

5.1.4 Sauvegarde des modèles

[ ]:
```
models_founded = {
    'RandomForest' : {'best_model' : grid_rf.best_estimator_,
                      'best_params' : grid_rf.best_params_
                      },
    'LogisticRegression' : {'best_model' : grid_lr.best_estimator_,
                            'best_params' : grid_lr.best_params_
                            },
    'XGBoost' : {'best_model' : grid_xgb.best_estimator_,
                 'best_params' : grid_xgb.best_params_
                 }
}
```

[32]:
```
models_man = {
    'RandomForest' : RandomForestClassifier(n_estimators=200,␣
↪criterion='log_loss', random_state=42, n_jobs=-1),
    'LogisticRegression' : LogisticRegression(max_iter=5000, solver='saga',␣
↪penalty='l2', random_state=2, n_jobs=-1),
    'XGBoost' : XGBClassifier(n_estimators=500, eval_metric='logloss',␣
↪learning_rate=0.1, random_state=42, n_jobs=-1)
}
```

[36]:
```
models = models_man.copy()
```

5.2 Entrainement et évaluation

5.2.1 Fonction de train et d'évaluation

[58]:
```
def train_evaluate(model, X_test, y_test, X_train=None, y_train=None,␣
↪model_name='Model', verbose=False):
    """

    Retunrs:
        _Entraîne le modèle si nécessaire et évalue ses performances sur les␣
↪données de test.
      Retourne un tuple contenant dans cet ordre une liste des opérations␣
↪effectuées, le rapport de classification, la matrice de confusion,
```

```
    l'accuracy, la précision, le rappel, le score F1 et les paramètre de la␣
 ↪courbe ROC._
  """

  operations_done = []
  if isinstance(X_train, (pd.DataFrame, np.ndarray)) and isinstance(y_train,␣
 ↪(pd.DataFrame, np.ndarray)) :
    if verbose :
      print(f"-------{model_name} training ... ")
      model.fit(X_train, y_train.values.ravel())
      print("-----------Training terminate.\n")
    else : model.fit(X_train, y_train.values.ravel())
    operations_done.append("Training")

  if verbose : print(f"-------{model_name} evaluation ...")
  y_pred = model.predict(X_test)
  y_pred_proba = model.predict_proba(X_test)[:, 1]
  operations_done.append("Evaluation")
  if verbose : print("-----------Evaluation terminate.\n")

  return (operations_done,
          classification_report(y_test, y_pred),
          confusion_matrix(y_test, y_pred),
          accuracy_score(y_test, y_pred),
          precision_score(y_test, y_pred),
          recall_score(y_test, y_pred),
          f1_score(y_test, y_pred),
          roc_curve(y_test, y_pred_proba)
      )
```

5.2.2 Train et Eval

[59]:
```
train_eval_data = {
    'RandomForest' : (X_train_res, y_train_res, X_test, y_test),
    'LogisticRegression' : (X_train_scaled_res, y_train_scaled_res,␣
 ↪X_test_scaled, y_test),
    'XGBoost' : (X_train_res, y_train_res, X_test, y_test)
}


reports = {}
confusion_mat = {}
metrics = {'RandomForest': {}, 'LogisticRegression': {}, 'XGBoost': {}}

# Évaluation
for name, model in models_man.items():
  print(f"\n_____{name}_____\n")
```

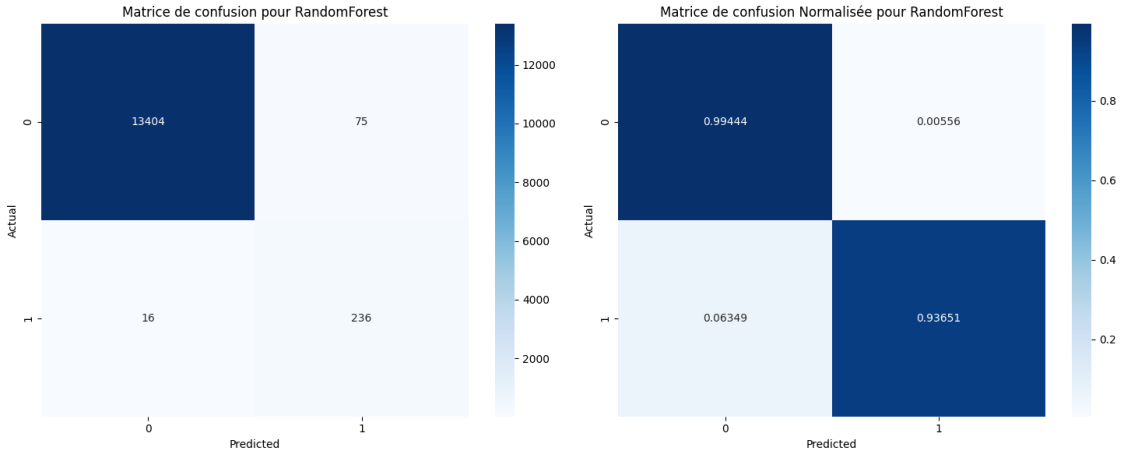
```
    x_tr, y_tr = train_eval_data[name][0], train_eval_data[name][1]
    x_te, y_te = train_eval_data[name][2], train_eval_data[name][3]
    _, reports[name], confusion_mat[name], metrics[name]['Accurancy'],␣
    ↪metrics[name]['Precision'], metrics[name]['RecallScore'],␣
    ↪metrics[name]['F1Score'], metrics[name]['RocCurve'] = train_evaluate(model,␣
    ↪x_te, y_te, x_tr, y_tr, model_name=name)

    print(f"     Basic metrics :")
    print(f"\tAccuracy ---- {metrics[name]['Accurancy']:.5f}")
    print(f"\tPrecision --- {metrics[name]['Precision']:.5f}")
    print(f"\tRecall ------ {metrics[name]['RecallScore']:.5f}")
    print(f"\tF1 Score ---- {metrics[name]['F1Score']:.5f}\n")
```

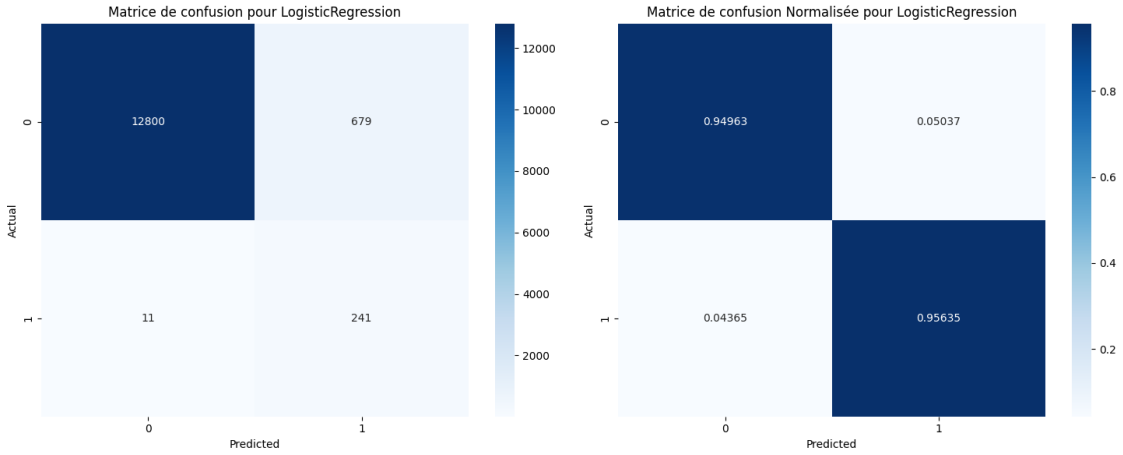
```
_____RandomForest_____

     Basic metrics :
        Accuracy ---- 0.99337
        Precision --- 0.75884
        Recall ------ 0.93651
        F1 Score ---- 0.83837


_____LogisticRegression_____

     Basic metrics :
        Accuracy ---- 0.94975
        Precision --- 0.26196
        Recall ------ 0.95635
        F1 Score ---- 0.41126


_____XGBoost_____

     Basic metrics :
        Accuracy ---- 0.99126
        Precision --- 0.70370
        Recall ------ 0.90476
        F1 Score ---- 0.79167
```

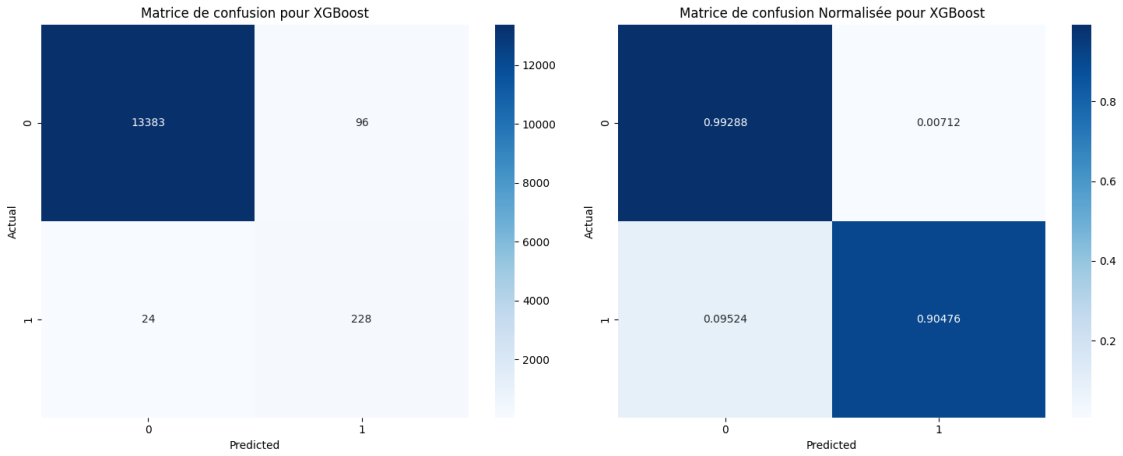
5.2.3 Confusion matrix

[60]:
```
for name, _ in models.items():
    cm = confusion_mat[name]
    plt.figure(figsize=(15, 6))

    plt.subplot(1, 2, 1)
```

```
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f"Matrice de confusion pour {name}")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")

    plt.subplot(1, 2, 2)
    sns.heatmap(cm/cm.sum(axis=1, keepdims=True),fmt='.5f', annot=True,␣
↪cmap='Blues')
    plt.title(f"Matrice de confusion Normalisée pour {name}")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")

    plt.tight_layout()
    plt.show()
```



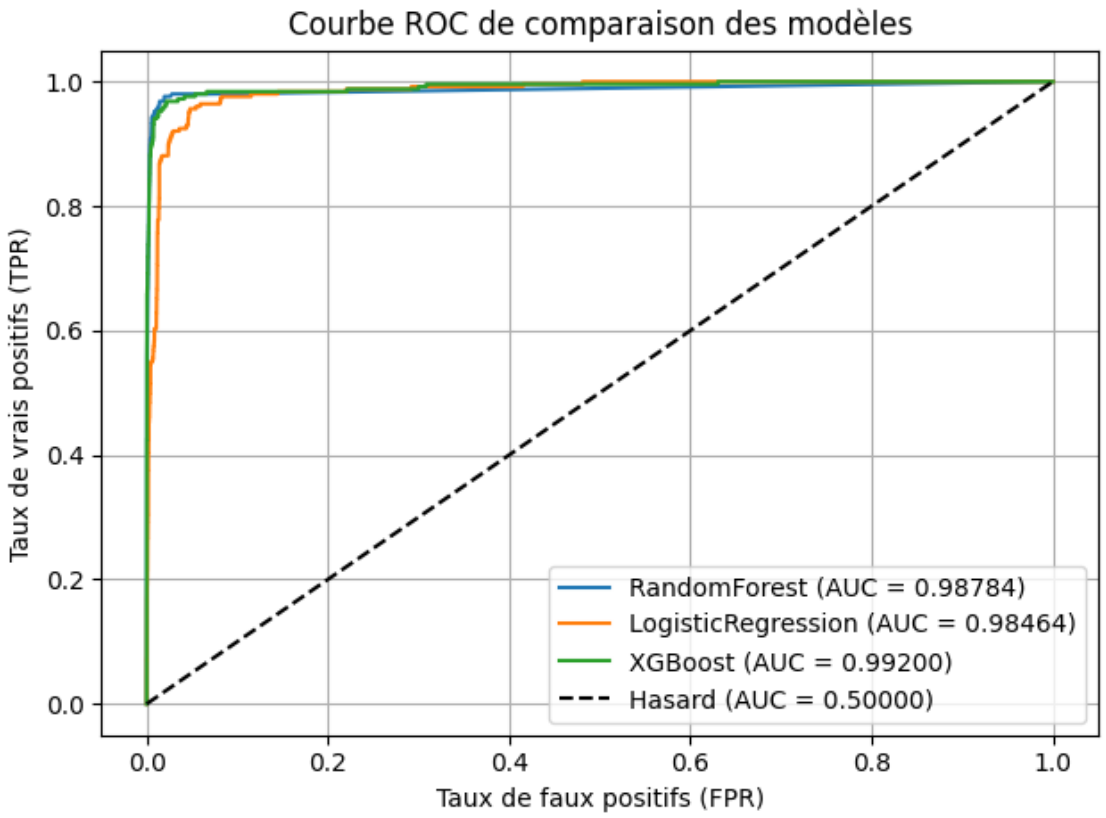




5.2.4 Roc-Auc

[61]:
```python
print(f"Roc_Auc :")
for name, _ in models.items():
    fpr, tpr, _ = metrics[name]['RocCurve']
    roc_auc = auc(fpr, tpr)
    print(f"\t{name} : {roc_auc:.5f}")

    plt.plot(fpr, tpr, label=f'{name} (AUC = {roc_auc:.5f})')

# Ligne diagonale "hasard"
plt.plot([0, 1], [0, 1], 'k--', label='Hasard (AUC = 0.50000)')

plt.title('Courbe ROC de comparaison des modèles')
plt.xlabel('Taux de faux positifs (FPR)')
plt.ylabel('Taux de vrais positifs (TPR)')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Roc_Auc :
        RandomForest : 0.98784
        LogisticRegression : 0.98464
        XGBoost : 0.99200
```



5.3 Analyse des performances

1. Random Forest

Metric	Value	Interprétation
Accuracy	0.99337	Très haute → Possible biais dû au déséquilibre de classes
Precision	0.75884	Bonne → peu de faux positifs
Recall	0.93651	Excellente détection des défauts
F1 Score	0.83837	Très équilibré
ROC AUC	0.98784	Très bonne capacité de discrimination

Conclusion : Modèle très performant, bon équilibre entre précision et rappel. Fiable pour détecter les défauts tout en limitant les refus injustifiés.

---

2. Logistic Regression

Metric	Value	Interprétation
Accuracy	0.94975	Correct, mais inférieur aux autres modèles

Metric	Value	Interprétation
Precision	0.26196	Faible : beaucoup de faux positifs
Recall	0.95635	Très élevé : capte quasiment tous les défauts
F1 Score	0.41126	Déséquilibré : la faiblesse de la précision plombe la performance globale
ROC AUC	0.98464	Très bon pouvoir de séparation entre clients sûrs et à risque

Conclusion : Mauvais seul, mais utile dans un ensemble grâce à son bon rappel et sa perspective linéaire complémentaire.

`3. XGBoost`

Metric	Value	Interprétation
Accuracy	0.99126	Très élevée
Precision	0.70370	Solide
Recall	0.90476	Très bon
F1 Score	0.79167	Bon équilibre, un peu en retrait par rapport à RandomForest
ROC AUC	0.99200	Le meilleur score AUC, très forte capacité à classer les clients

Conclusion : Excellent modèle, légèrement moins équilibré que RandomForest en F1, mais surpasse tous en AUC. Parfait comme base puissante dans un ensemble.

5.4 Résumé comparatif des modèles

Modèle	Accuracy	Precision	Recall	F1 Score	ROC AUC	Verdict
Random Forest	0.99337	0.75884	0.93651	0.83837	0.98784	Meilleur équilibre global
Logistic Reg.	0.94975	0.26196	0.95635	0.41126	0.98464	Faible seul, utile en stacking
XGBoost	0.99126	0.70370	0.90476	0.79167	0.99200	Meilleur modèle individuel

> Tous les modèles ont une *excellente capacité à classer correctement les clients à risque vs non à risque*, même Logistic Regression, qui semblait faible en F1 score. *Son ROC AUC élevé prouve qu'elle reste utile dans un ensemble.*

5.5 Amélioration possible

- Les trois modèles apportent des forces complémentaires :
  - XGBoost excelle en discrimination globale (AUC).
  - RandomForest offre un très bon équilibre général.
  - Logistic Regression a un très fort rappel, utile dans un ensemble.

Stacking de ces modèles peut offrir une synergie puissante : *robustesse + diversité des erreurs.*

5.6 Technique d'ensemble

Vue les résultats précédents, essayant une technique d'ensemble (le stacking) pour espérer avoir de bien meilleurs résultats.

5.6.1 Définition de l'ensemble de stacking

[64]:
```
# Les base learners pour le StackingClassifier

base_learners = [
    ('RandomForest', Pipeline([
        ('smote', SMOTE()),
        ('rf', models_man['RandomForest'])
    ])),

    ('XGBoost', Pipeline([
        ('smote', SMOTE()),
        ('xgb', models_man['XGBoost'])
    ])),

    ('LogisticRegression', Pipeline([
        ('scaler', StandardScaler()),
        ('smote', SMOTE()),
        ('lr', models_man['LogisticRegression'])
    ]))
]


"""
base_learners = [
    ('RandomForest', Pipeline([
        ('smote', SMOTE()),
        ('rf', models['RandomForest']['best_model'])
    ])),

    ('XGBoost', Pipeline([
        ('smote', SMOTE()),
        ('xgb', models['XGBoost']['best_model'])
    ])),

    ('LogisticRegression', Pipeline([
        ('scaler', StandardScaler()),
        ('smote', SMOTE()),
        ('lr', models['LogisticRegression']['best_model'])
    ]))
```

```
]
"""

# Stacking classifier
stack_clf = StackingClassifier(
    estimators=base_learners,
    final_estimator=Pipeline([('scaler2', StandardScaler()),␣
 ↪('LogisticRegression', LogisticRegression())]),
    cv=5,
    passthrough=False
)
```

5.6.2 Train et Eval du modèle

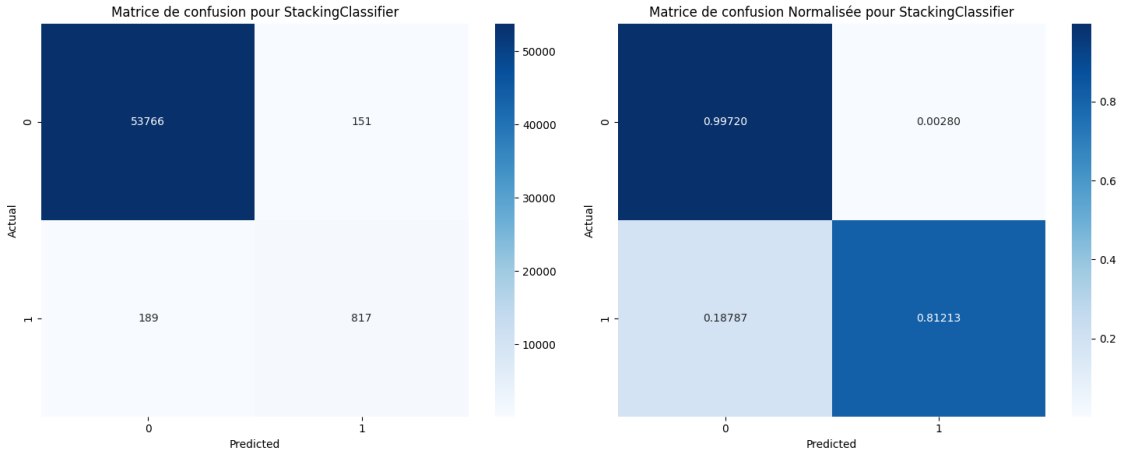
[66]:
```
name = 'StackingClassifier'
metrics[name]={}
_, reports[name], confusion_mat[name], metrics[name]['Accurancy'],␣
 ↪metrics[name]['Precision'], metrics[name]['RecallScore'],␣
 ↪metrics[name]['F1Score'], (fpr, tpr, _ ) = train_evaluate(stack_clf,␣
 ↪X_train, y_train, X_test, y_test, verbose=True)

print(f"\tModel: {name}")
print(f"Accuracy: {metrics[name]['Accurancy']:.5f}")
print(f"Precision: {metrics[name]['Precision']:.5f}")
print(f"Recall: {metrics[name]['RecallScore']:.5f}")
print(f"F1 Score: {metrics[name]['F1Score']:.5f}")
print(f"Roc_Auc Score: {auc(fpr, tpr) :.5f}\n")
```

```
-------Model training …
-----------Training terminate.

-------Model evaluation …
-----------Evaluation terminate.

        Model: StackingClassifier
Accuracy: 0.99381
Precision: 0.84544
Recall: 0.81014
F1 Score: 0.82741
Roc_Auc Score: 0.99145
```

5.6.3 Matrices de confusion

[41]:
```
cm = confusion_mat[name]
plt.figure(figsize=(15, 6))

plt.subplot(1, 2, 1)
```

```
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title(f"Matrice de confusion pour {name}")
plt.xlabel("Predicted")
plt.ylabel("Actual")

plt.subplot(1, 2, 2)
sns.heatmap(cm/cm.sum(axis=1, keepdims=True),fmt='.5f', annot=True,␣
 ↪cmap='Blues')
plt.title(f"Matrice de confusion Normalisée pour {name}")
plt.xlabel("Predicted")
plt.ylabel("Actual")

plt.tight_layout()
plt.show()
```



5.7 Bilan du Stacking

Récap des preformances :

Modèle	Accuracy	Precision	Recall	F1 Score	AUC-ROC
RandomForest	0.99337	0.75884	**0.93651**	**0.83837**	0.98784
LogisticRegression	0.94975	0.26196	0.95635	0.41126	0.98464
XGBoost	0.99126	0.70370	0.90476	0.79167	**0.99200**
StackingClassifier	**0.99381**	**0.84401**	0.81213	0.82776	0.99153

Métrique	Observation
Accuracy	Très haute, légèrement au-dessus de tous les modèles de base
Precision	**Meilleure (0.84544)** → le stacking réduit efficacement les faux positifs
Recall	Moins bon que RandomForest ($0.81 < 0.936$) → quelques défauts non détectés

Métrique	Observation
F1 Score	Très bon équilibre, presque au niveau du meilleur (RandomForest)
ROC AUC	Excellente discrimination (0.99145), proche de XGBoost

Conclusion :

Le **StackingClassifier** tire profit de la complémentarité des modèles :

- Il ***réduit les faux positifs*** grâce à une meilleure précision.
- Il ***maintient une excellente capacité de généralisation*** (AUC > 0.99).
- Il ***compense les faiblesses individuelles***, notamment celles de la régression logistique.

5.8 Voyons ce que donne une cross-validation sur l'ensemble du stack

[ ]:
```
print('Validation Croisée sur StackingClassifier')
for metric in tqdm(['accuracy', 'precision', 'recall', 'f1', 'roc_auc']):
    scores = cross_val_score(stack_clf,
                             X_train,
                             y_train.values.ravel(),
                             cv=StratifiedKFold(
                                    n_splits=5,
                                    shuffle=True,
                                    random_state=42),
                             scoring=metric)
    print(f"{metric.upper()} moyen avec StackingClassifier : {np.mean(scores):.
 ↪4f}")
```

Récap des preformances :

Modèle	Accuracy	Precision	Recall	F1 Score	AUC-ROC
RandomForest	**0.99527**	0.88163	**0.85714**	**0.86922**	0.98641
LogisticRegression	0.98784	0.77070	0.48016	0.59169	0.97742
XGBoost	0.99403	0.84274	0.82937	0.83600	0.99264
Stacking_model	0.99512	0.88703	0.84127	0.86354	0.99286
Stacking_model with cross-val	_	_	_	_	_

5.9 Réentrainement du model et sauvegarde de tous les éléments nécessaires

[ ]:
```
# Réentrainement du stacking
start = time.time()
stack_clf.fit(X_train, y_train.values.ravel())
print(f"Réentrainement du StackingClassifier terminé en {time.time() - start:.
 ↪2f} secondes.\n")
```

```
Réentrainement du StackingClassifier terminé en 1095.76 secondes.
```

[ ]:
```
# Sauvegarde des éléments de preprocessing utiles
joblib.dump(Hot_encoder, "../API/processing_elements/Hot_encoder.pkl")
joblib.dump(X_train.columns.tolist(), "../API/processing_elements/feature_list.
 ↪pkl")

# Suavegarde du dtype de la variable 'loan_type'
joblib.dump(loan_type_dtype, "../API/processing_elements/loan_type_dtype.pkl")

# Sauvegarde du scaler
joblib.dump(scaler, "../API/processing_elements/scaler.pkl")

# Sauvegarde du modèle final complet
joblib.dump(stack_clf, "../API/ml_models/stacking_model.pkl")

# Sauvegarde des pipelines complets des bases learners :
joblib.dump(base_learners[0][1], "../API/ml_models/RandomForest_pipeline.pkl")
joblib.dump(base_learners[1][1], "../API/ml_models/XGBoost_pipeline.pkl")
joblib.dump(base_learners[2][1], "../API/ml_models/LogisticRegression_pipeline.
 ↪pkl")
```

[ ]:
```
['../API/ml_models/LogisticRegression_pipeline.pkl']
```

5.10 Voyons maintenant l'interprètabilité de notre model final :

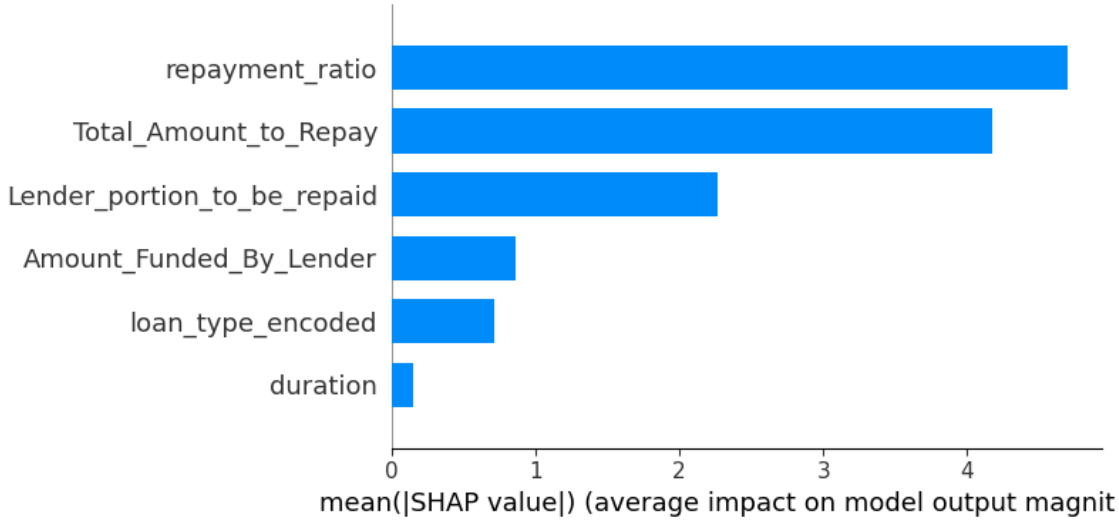
"Nous allons appliqués SHAP sur le meilleur estimateur individuel (XGBoost), utilisé comme proxy interprétable du modèle de stacking."

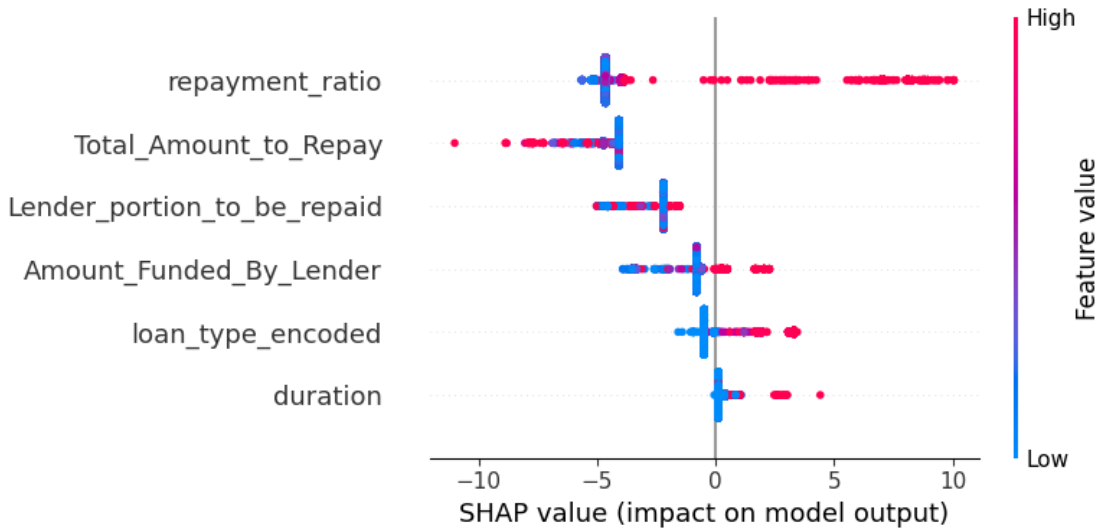
[77]:
```
xgb_model = stack_clf.named_estimators_['XGBoost'].named_steps['xgb']

explainer = shap.Explainer(xgb_model)

shap_values = explainer(X_test_scaled)

# Sauvegarde
joblib.dump(explainer, "../API/shap_explainer/explainer.pkl")
```

[77]:
```
['../API/shap_explainer/explainer.pkl']
```

5.10.1 Summary Plot (global, top features)

[81]:
```
# Résumé global des effets
shap.summary_plot(shap_values, X_test, plot_type="bar")  # Top features
shap.summary_plot(shap_values, X_test)  # Distribution par feature
```

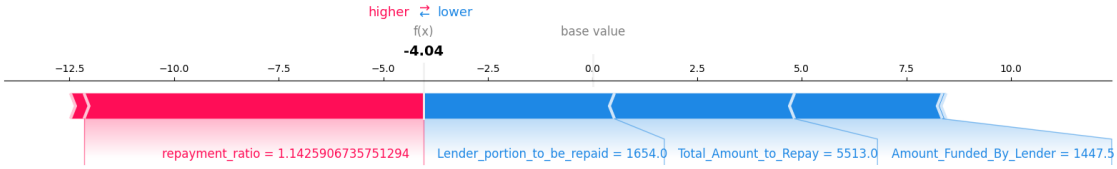




5.10.2 Force Plot (pour une prédiction spécifique)

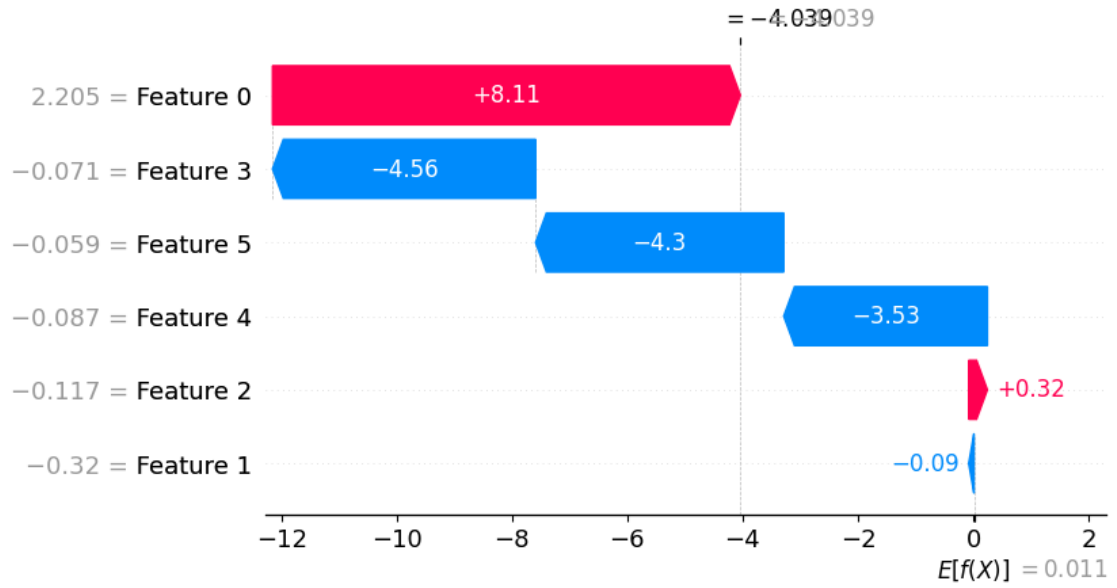
[88]:
```
shap.initjs()
i = 0
shap.force_plot(explainer.expected_value, shap_values[i].values, X_test.
  ↪iloc[i], matplotlib=True, show=True)
```

```
<IPython.core.display.HTML object>
```



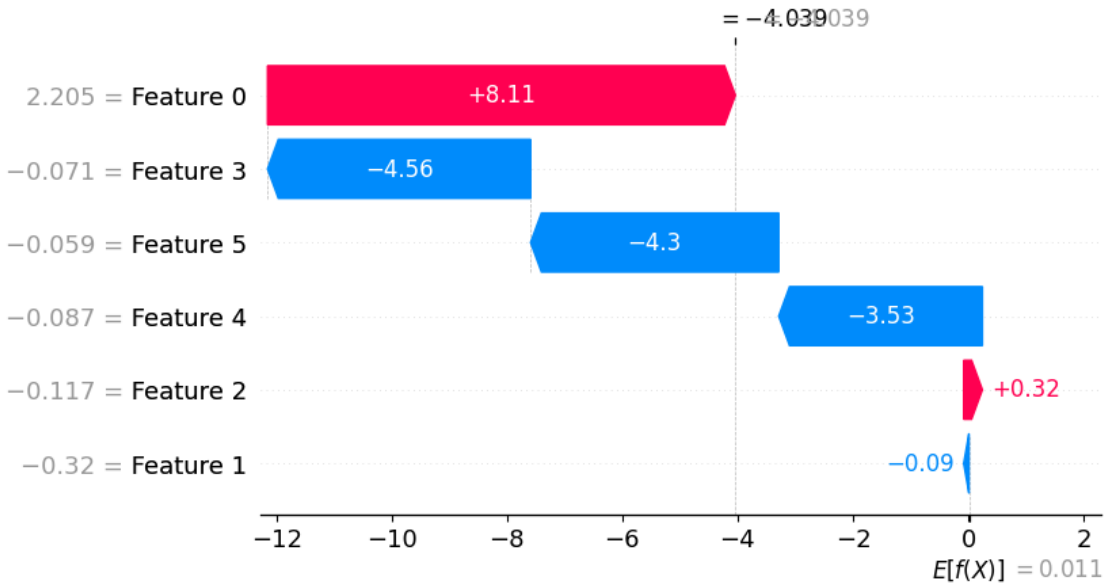
5.10.3 Waterfall Plot (décision individuelle, statique)

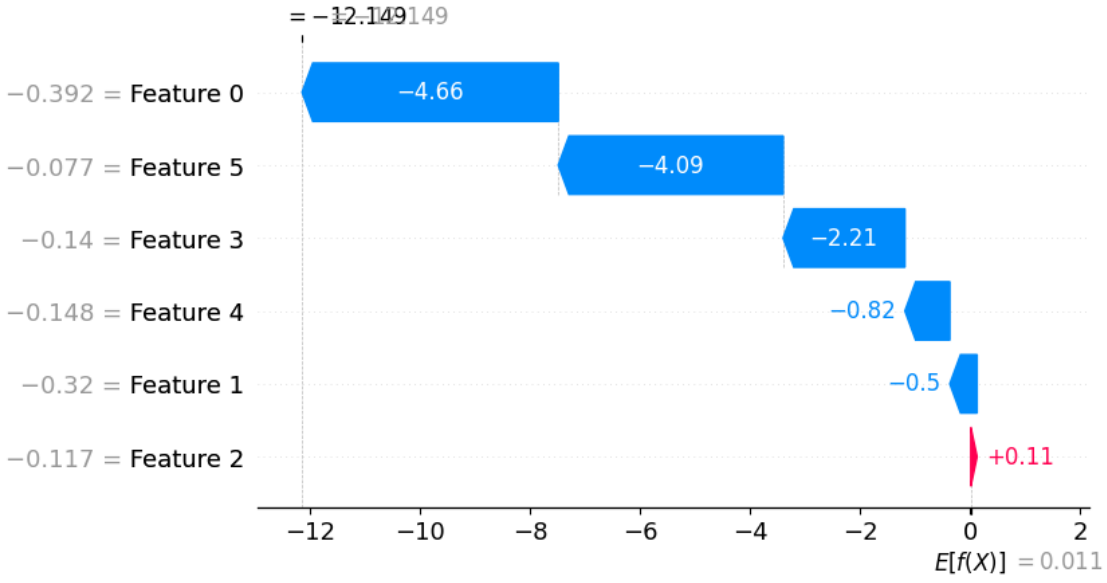
[83]:
```
shap.plots.waterfall(shap_values[i])
```

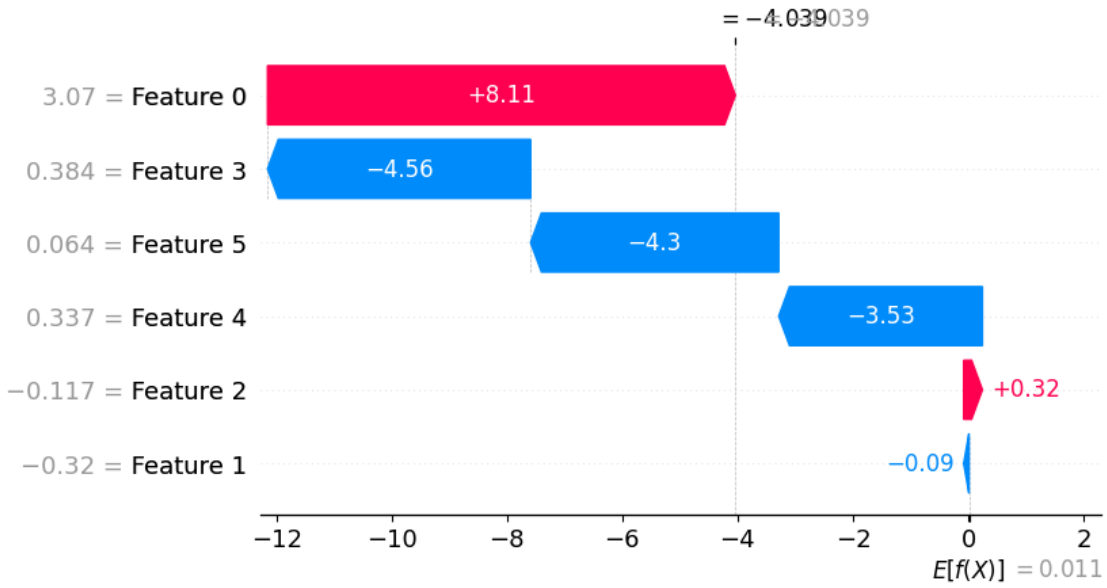


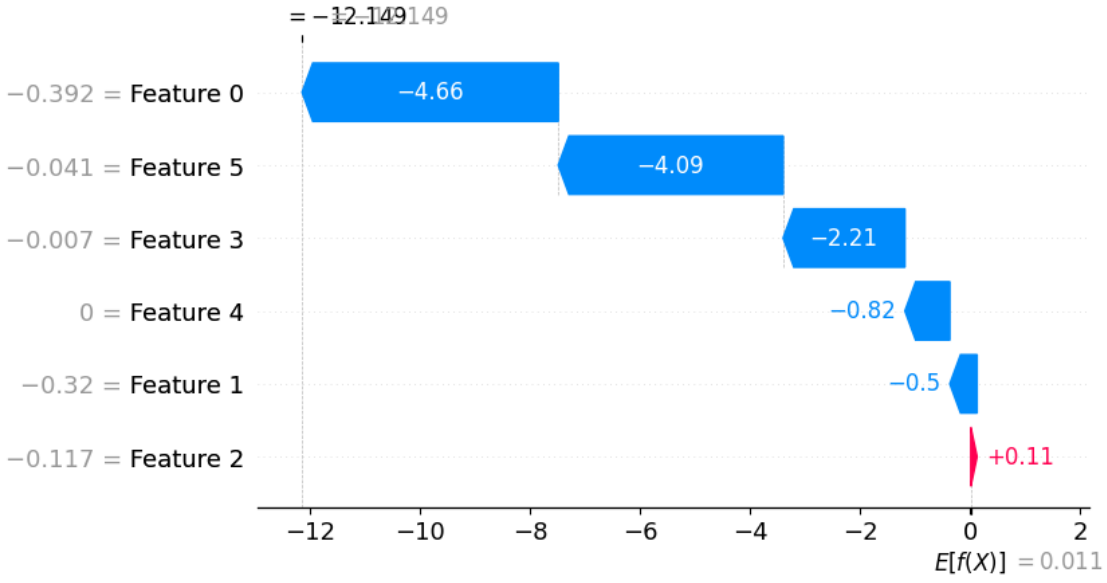
5.10.4 Comparaison de plusieurs individus (local SHAP values)

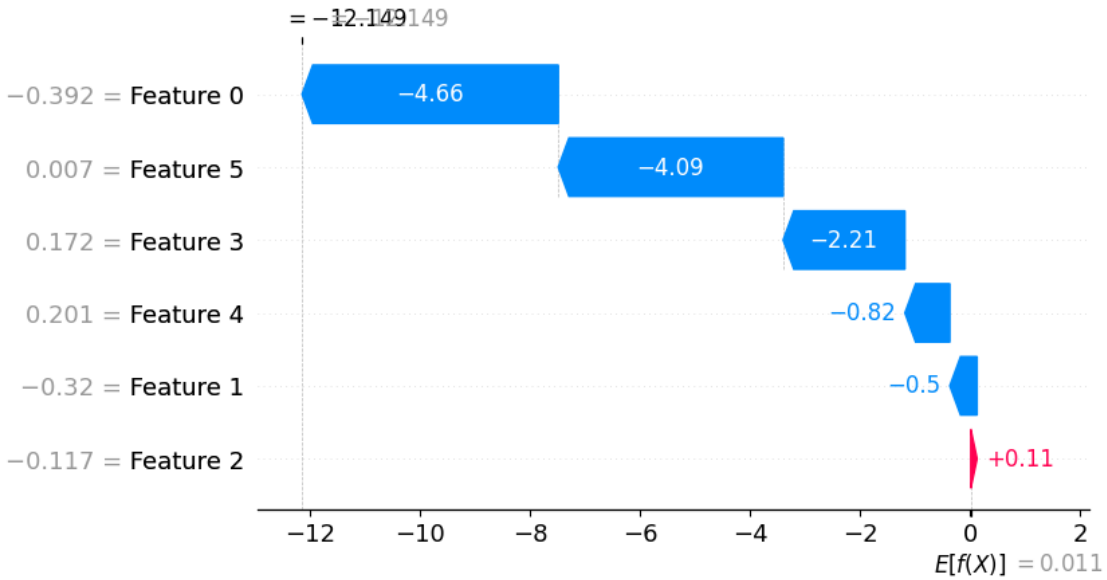
[86]:
```
subset = X_test_scaled[:5]
shap_values_subset = explainer(subset)

for i in range(len(subset)):
    shap.plots.waterfall(shap_values_subset[i])
    plt.tight_layout()
```


= −4.039039
2.205 = Feature 0
−0.071 = Feature 3
−0.059 = Feature 5
−0.087 = Feature 4
−0.117 = Feature 2
−0.32 = Feature 1
+8.11
−4.56
−4.3
−3.53
+0.32
−0.09
−12
−10
−8
−6
−4
−2
0
2
E[f(X)] = 0.011


= −12.149149
−0.392 = Feature 0
−0.077 = Feature 5
−0.14 = Feature 3
−0.148 = Feature 4
−0.32 = Feature 1
−0.117 = Feature 2
−4.66
−4.09
−2.21
−0.82
−0.5
+0.11
−12
−10
−8
−6
−4
−2
0
2
E[f(X)] = 0.011


= −4.039039
3.07 = Feature 0
0.384 = Feature 3
0.064 = Feature 5
0.337 = Feature 4
−0.117 = Feature 2
−0.32 = Feature 1
+8.11
−4.56
−4.3
−3.53
+0.32
−0.09
−12
−10
−8
−6
−4
−2
0
2
E[f(X)] = 0.011


= −12.149149
−0.392 = Feature 0
−0.041 = Feature 5
−0.007 = Feature 3
0 = Feature 4
−0.32 = Feature 1
−0.117 = Feature 2
−4.66
−4.09
−2.21
−0.82
−0.5
+0.11
−12
−10
−8
−6
−4
−2
0
2
E[f(X)] = 0.011




```
<Figure size 640x480 with 0 Axes>
```

6 Prédiction

[ ]:

[150]:
```
sys.path.append(os.path.abspath(os.path.join(os.getcwd(), '..')))
import API.processing_elements.preprocess as preprocess_module
importlib.reload(preprocess_module)  # recharge la dernière version
preprocessor = preprocess_module.preprocessor
#preprocessor = joblib.load("API/processing_elements/preprocess.py")
model = joblib.load("../API/ml_models/stacking_model.pkl")
```

[151]:
```
csv_path = "../Data/Test.csv"
new_df = pd.read_csv(csv_path)

print(f"Nombre de lignes à prédire : {len(new_df)}")
new_df.head()
```

```
Nombre de lignes à prédire : 18594
```

[151]:

	ID	customer_id	country_id	tbl_loan_id	lender_id	\
0	ID_269404226088267278	269404	Kenya	226088	267278	
1	ID_255356300042267278	255356	Kenya	300042	267278	
2	ID_257026243764267278	257026	Kenya	243764	267278	
3	ID_264617299409267278	264617	Kenya	299409	267278	
4	ID_247613296713267278	247613	Kenya	296713	267278	

```
  loan_type  Total_Amount  Total_Amount_to_Repay disbursement_date  \
0    Type_1        1919.0                 1989.0        2022-07-27
1    Type_1        2138.0                 2153.0        2022-11-16
2    Type_1        8254.0                 8304.0        2022-08-24
3    Type_1        3379.0                 3379.0        2022-11-15
4    Type_1         120.0                  120.0        2022-11-10

     due_date  duration New_versus_Repeat  Amount_Funded_By_Lender  \
0  2022-08-03         7       Repeat Loan                    575.7
1  2022-11-23         7       Repeat Loan                      0.0
2  2022-08-31         7       Repeat Loan                    207.0
3  2022-11-22         7       Repeat Loan                   1013.7
4  2022-11-17         7       Repeat Loan                     36.0

   Lender_portion_Funded  Lender_portion_to_be_repaid
0               0.300000                        597.0
1               0.000000                          0.0
2               0.025079                        208.0
3               0.300000                       1014.0
4               0.300000                         36.0
```

[152]:
```
X_new = preprocessor(new_df)
X_new.columns.to_list()
```

```
2025-06-15 01:51:56.380 | INFO     |
API.processing_elements.preprocess:preprocessor:20 -
 Preprocessing started
2025-06-15 01:51:56.607 | INFO     |
API.processing_elements.preprocess:preprocessor:69 -
 Preprocessing completed
```

[152]:
```
['repayment_ratio',
 'loan_type_encoded',
 'duration',
 'Lender_portion_to_be_repaid',
 'Amount_Funded_By_Lender',
 'Total_Amount_to_Repay']
```

[158]:
```
probas = model.predict_proba(X_new)[:, 1]  # probabilité de défaut
preds = (probas >= 0.5).astype(int)

print(f"{preds.sum()} défauts prédits sur un total de {len(preds)} prédictions.
 ↪")
```

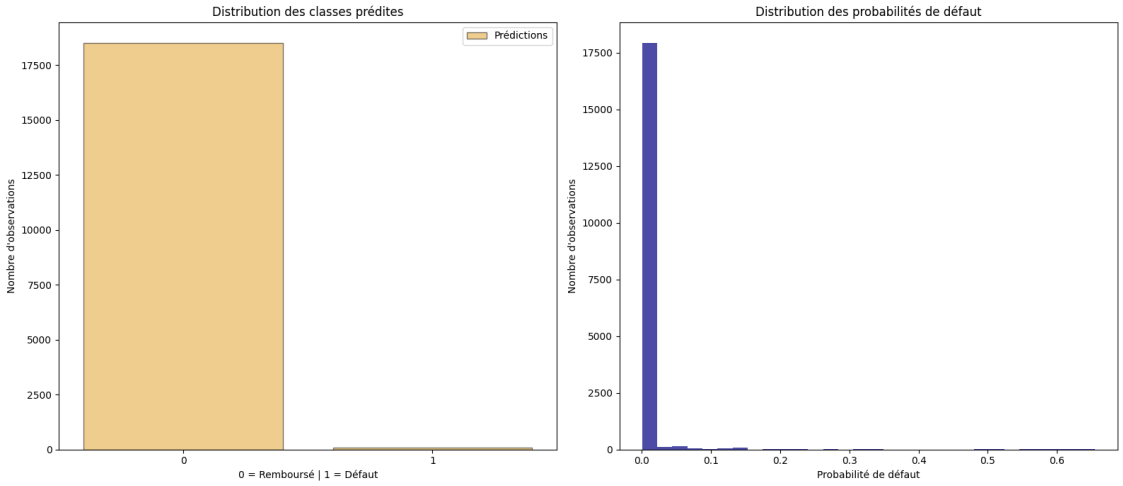
```
79 défauts prédits sur un total de 18594 prédictions.
```

[153]:
```
new_df['probability_default'] = probas
new_df['prediction_default'] = preds
```

```
# Sauvegarde avec prédictions
new_df.to_csv("../Data/nouvelle_donnee_predite.csv", index=False)

print("  Prédictions ajoutées et fichier sauvegardé : ../Data/
 ↪nouvelle_donnee_predite.csv")
```

```
 Prédictions ajoutées et fichier sauvegardé :
../Data/nouvelle_donnee_predite.csv
```

[154]:
```
plt.figure(figsize=(16, 7))
plt.subplot(1, 2, 1)
sns.countplot(x=new_df['prediction_default'], color='orange', alpha=0.5,␣
 ↪label='Prédictions', edgecolor='black')
plt.legend()
plt.title("Distribution des classes prédites")
plt.xlabel("0 = Remboursé | 1 = Défaut")
plt.ylabel("Nombre d'observations")

plt.subplot(1, 2, 2)
plt.hist(probas, bins=30, alpha=0.7, color='navy')
plt.title("Distribution des probabilités de défaut")
plt.xlabel("Probabilité de défaut")
plt.ylabel("Nombre d'observations")
plt.tight_layout()
plt.show()
```



7 Bulding du pipeline de processing pour l'API

[87]:
```
#from '../API/processing_elements/preprocess.py' import preprocessor

#pipeline = preprocessor(X_train)
#pipeline.fit(X_train)

# Enregistre le preprocess complet
#joblib.dump(pipeline, "../API/processing_elements/preprocess.pkl")
```

8

[162]:
```
!jupyter nbconvert --to html book.ipynb --output ../Docs/Rapport/book.html␣
 ↪--allow-errors
!jupyter nbconvert --to pdf book.ipynb --output ../Docs/Rapport/book.pdf␣
 ↪--allow-errors
!jupyter nbconvert --to slides book.ipynb --output ../Docs/Rapport/book.slides.
 ↪html --no-prompt --no-input --allow-errors
```

```
[NbConvertApp] Converting notebook book.ipynb to html
[NbConvertApp] WARNING | Alternative text is missing on 35 image(s).
[NbConvertApp] Writing 3720472 bytes to ../Docs/Rapport/book.html
[NbConvertApp] Converting notebook book.ipynb to pdf
[NbConvertApp] Support files will be in ../Docs/Rapport/book_files/
[NbConvertApp] Writing 149368 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 2020756 bytes to ../Docs/Rapport/book.pdf
[NbConvertApp] Converting notebook book.ipynb to slides
[NbConvertApp] WARNING | Alternative text is missing on 35 image(s).
[NbConvertApp] Writing 3586217 bytes to
../Docs/Rapport/book.slides.html.slides.html
```