Untitled

October 25, 2025

0.1 Importation des bibliothèques

Dans cette première étape, je vais **importer les principales bibliothèques Python** nécessaires pour l'analyse des données, la visualisation et la modélisation.

Ces outils me permettront de **charger le dataset**, de l'explorer, puis de **construire un modèle supervisé** afin de **prédire le statut d'un prêt** (`Loan_Status`).

[1]:
```
# Importation des bibliothèques principales

# Manipulation et analyse des données
import pandas as pd
import numpy as np

# Visualisation
import matplotlib.pyplot as plt
import seaborn as sns

# Préparation et encodage
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler

# Modèles de machine learning
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

# Évaluation des performances
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report

# Pour éviter les avertissements inutiles
import warnings
warnings.filterwarnings('ignore')

# Réglages d'affichage
pd.set_option('display.max_columns', None)
sns.set_style('whitegrid')
```

0.2 Chargement et première exploration du dataset

Je vais maintenant charger le fichier **`train.csv`** afin de découvrir sa structure.

L'objectif de cette étape est de : 1. Vérifier le **nombre d'observations**. 2. Identifier les **types de variables** disponibles. 3. Obtenir un **aperçu général** du contenu.

Cela permettra ensuite de préparer correctement les données pour l'entraînement du modèle.

[2]:
```
# Chargement du dataset principal
df = pd.read_csv("train.csv")

# Affichage des 5 premières lignes pour un aperçu rapide
print("Aperçu du jeu de données :")
display(df.head())

# Informations générales sur les colonnes, types et valeurs manquantes
print("\nInformations sur les variables :")
df.info()

# Dimensions du dataset
print(f"\nLe dataset contient {df.shape[0]} lignes et {df.shape[1]} colonnes.")

# Statistiques descriptives pour les variables numériques
print("\nStatistiques descriptives :")
display(df.describe())

# Aperçu des valeurs uniques de la variable cible (Loan Status)
print("\nValeurs uniques de la variable cible (Loan Status) :")
print(df['Loan Status'].value_counts())
```

```
Aperçu du jeu de données :
```

	ID	Loan Amount	Funded Amount	Funded Amount Investor	Term	\
0	65087372	10000	32236	12329.36286	59	
1	1450153	3609	11940	12191.99692	59	
2	1969101	28276	9311	21603.22455	59	
3	6651430	11170	6954	17877.15585	59	
4	14354669	16890	13226	13539.92667	59	

	Batch Enrolled	Interest Rate	Grade	Sub Grade	Employment Duration	\
0	BAT2522922	11.135007	B	C4	MORTGAGE	
1	BAT1586599	12.237563	C	D3	RENT	
2	BAT2136391	12.545884	F	D4	MORTGAGE	
3	BAT2428731	16.731201	C	C3	MORTGAGE	
4	BAT5341619	15.008300	C	D4	MORTGAGE	

	Home Ownership	Verification Status	Payment Plan	Loan Title	\
0	176346.62670	Not Verified	n	Debt Consolidation	
1	39833.92100	Source Verified	n	Debt consolidation	

```
2     91506.69105     Source Verified            n       Debt Consolidation
3    108286.57590     Source Verified            n       Debt consolidation
4     44234.82545     Source Verified            n  Credit card refinancing

   Debit to Income  Delinquency - two years  Inquires - six months  \
0        16.284758                        1                      0
1        15.412409                        0                      0
2        28.137619                        0                      0
3        18.043730                        1                      0
4        17.209886                        1                      3

   Open Account  Public Record  Revolving Balance  Revolving Utilities  \
0            13              0              24246            74.932551
1            12              0                812            78.297186
2            14              0               1843             2.073040
3             7              0              13819            67.467951
4            13              1               1544            85.250761

   Total Accounts Initial List Status  Total Received Interest  \
0               7                   w               2929.646315
1              13                   f                772.769385
2              20                   w                863.324396
3              12                   w                288.173196
4              22                   w                129.239553

   Total Received Late Fee   Recoveries  Collection Recovery Fee  \
0                 0.102055     2.498291                 0.793724
1                 0.036181     2.377215                 0.974821
2                18.778660     4.316277                 1.020075
3                 0.044131     0.107020                 0.749971
4                19.306646  1294.818751                 0.368953

   Collection 12 months Medical Application Type  Last week Pay  \
0                            0       INDIVIDUAL             49
1                            0       INDIVIDUAL            109
2                            0       INDIVIDUAL             66
3                            0       INDIVIDUAL             39
4                            0       INDIVIDUAL             18

   Accounts Delinquent  Total Collection Amount  Total Current Balance  \
0                    0                       31                 311301
1                    0                       53                 182610
2                    0                       34                  89801
3                    0                       40                   9189
4                    0                      430                 126029

   Total Revolving Credit Limit  Loan Status
0                          6619            0
```

```
1                         20885           0
2                         26155           0
3                         60214           0
4                         22579           0


Informations sur les variables :
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 67463 entries, 0 to 67462
Data columns (total 35 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   ID                              67463 non-null  int64
 1   Loan Amount                     67463 non-null  int64
 2   Funded Amount                   67463 non-null  int64
 3   Funded Amount Investor          67463 non-null  float64
 4   Term                            67463 non-null  int64
 5   Batch Enrolled                  67463 non-null  object
 6   Interest Rate                   67463 non-null  float64
 7   Grade                           67463 non-null  object
 8   Sub Grade                       67463 non-null  object
 9   Employment Duration             67463 non-null  object
 10  Home Ownership                  67463 non-null  float64
 11  Verification Status             67463 non-null  object
 12  Payment Plan                    67463 non-null  object
 13  Loan Title                      67463 non-null  object
 14  Debit to Income                 67463 non-null  float64
 15  Delinquency - two years         67463 non-null  int64
 16  Inquires - six months           67463 non-null  int64
 17  Open Account                    67463 non-null  int64
 18  Public Record                   67463 non-null  int64
 19  Revolving Balance               67463 non-null  int64
 20  Revolving Utilities             67463 non-null  float64
 21  Total Accounts                  67463 non-null  int64
 22  Initial List Status             67463 non-null  object
 23  Total Received Interest         67463 non-null  float64
 24  Total Received Late Fee         67463 non-null  float64
 25  Recoveries                      67463 non-null  float64
 26  Collection Recovery Fee         67463 non-null  float64
 27  Collection 12 months Medical    67463 non-null  int64
 28  Application Type                67463 non-null  object
 29  Last week Pay                   67463 non-null  int64
 30  Accounts Delinquent             67463 non-null  int64
 31  Total Collection Amount         67463 non-null  int64
 32  Total Current Balance           67463 non-null  int64
 33  Total Revolving Credit Limit    67463 non-null  int64
 34  Loan Status                     67463 non-null  int64
dtypes: float64(9), int64(17), object(9)
```

```
memory usage: 18.0+ MB

Le dataset contient 67463 lignes et 35 colonnes.

Statistiques descriptives :

                ID   Loan Amount  Funded Amount  Funded Amount Investor  \
count  6.746300e+04  67463.000000   67463.000000            67463.000000
mean   2.562761e+07  16848.902776   15770.599114            14621.799323
std    2.109155e+07   8367.865726    8150.992662             6785.345170
min    1.297933e+06   1014.000000    1014.000000             1114.590204
25%    6.570288e+06  10012.000000    9266.500000             9831.684984
50%    1.791565e+07  16073.000000   13042.000000            12793.682170
75%    4.271521e+07  22106.000000   21793.000000            17807.594120
max    7.224578e+07  35000.000000   34999.000000            34999.746430

              Term  Interest Rate  Home Ownership  Debit to Income  \
count  67463.000000   67463.000000    67463.000000     67463.000000
mean      58.173814      11.846258    80541.502522        23.299241
std        3.327441       3.718629    45029.120366         8.451824
min       36.000000       5.320006    14573.537170         0.675299
25%       58.000000       9.297147    51689.843335        16.756416
50%       59.000000      11.377696    69335.832680        22.656658
75%       59.000000      14.193533    94623.322785        30.048400
max       59.000000      27.182348   406561.536400        39.629862

       Delinquency - two years  Inquires - six months  Open Account  \
count              67463.000000           67463.000000  67463.000000
mean                   0.327127               0.145754     14.266561
std                    0.800888               0.473291      6.225060
min                    0.000000               0.000000      2.000000
25%                    0.000000               0.000000     10.000000
50%                    0.000000               0.000000     13.000000
75%                    0.000000               0.000000     16.000000
max                    8.000000               5.000000     37.000000

       Public Record  Revolving Balance  Revolving Utilities  Total Accounts  \
count   67463.000000       67463.000000         67463.000000    67463.000000
mean        0.081437        7699.342425            52.889443       18.627929
std         0.346606        7836.148190            22.539450        8.319246
min         0.000000           0.000000             0.005172        4.000000
25%         0.000000        2557.000000            38.658825       13.000000
50%         0.000000        5516.000000            54.082334       18.000000
75%         0.000000       10184.500000            69.177117       23.000000
max         4.000000      116933.000000           100.880050       72.000000

       Total Received Interest  Total Received Late Fee    Recoveries  \
count              67463.000000             67463.000000  67463.000000
```

```
mean               2068.992542                1.143969     59.691578
std                2221.918745                5.244365    357.026346
min                   4.736746                0.000003      0.000036
25%                 570.903814                0.021114      1.629818
50%                1330.842771                0.043398      3.344524
75%                2656.956837                0.071884      5.453727
max               14301.368310               42.618882   4354.467419

       Collection Recovery Fee  Collection 12 months Medical  Last week Pay  \
count             67463.000000                  67463.000000   67463.000000
mean                  1.125141                      0.021301      71.163260
std                   3.489885                      0.144385      43.315845
min                   0.000036                      0.000000       0.000000
25%                   0.476259                      0.000000      35.000000
50%                   0.780141                      0.000000      68.000000
75%                   1.070566                      0.000000     105.000000
max                 166.833000                      1.000000     161.000000

       Accounts Delinquent  Total Collection Amount  Total Current Balance  \
count              67463.0             67463.000000           6.746300e+04
mean                   0.0               146.467990           1.595739e+05
std                    0.0               744.382233           1.390332e+05
min                    0.0                 1.000000           6.170000e+02
25%                    0.0                24.000000           5.037900e+04
50%                    0.0                36.000000           1.183690e+05
75%                    0.0                46.000000           2.283750e+05
max                    0.0             16421.000000           1.177412e+06

       Total Revolving Credit Limit   Loan Status
count                  67463.000000  67463.000000
mean                   23123.005544      0.092510
std                    20916.699999      0.289747
min                     1000.000000      0.000000
25%                     8155.500000      0.000000
50%                    16733.000000      0.000000
75%                    32146.500000      0.000000
max                   201169.000000      1.000000


Valeurs uniques de la variable cible (Loan Status) :
Loan Status
0    61222
1     6241
Name: count, dtype: int64
```

0.3 Data Preparation (Préparation des Données)

Dans cette partie, je vais préparer les données afin qu'elles soient **exploitables par les modèles d'apprentissage supervisé**.

Les principales étapes de ce processus (Data Preprocessing) consistent à :

1. **Nettoyer les valeurs manquantes** (Imputation).
2. **Corriger les types de variables** si nécessaire.
3. **Créer de nouvelles variables utiles** (*Feature Engineering*).
4. **Encoder les variables catégorielles** (ex: *One-Hot Encoding*).
5. **Normaliser ou Standardiser les données numériques**.

[3]:
```
# Data Preparation (Préparation des Données)

# Nettoyage des données
# Supprimer les doublons
df.drop_duplicates(inplace=True)

# Vérification des valeurs manquantes
print("Valeurs manquantes par colonne :")
print(df.isnull().sum())

# Imputation simple : colonnes numériques -> mediane, colonnes catégorielles ->␣
↪"Unknown"
num_cols = df.select_dtypes(include=['int64', 'float64']).columns.tolist()
cat_cols = df.select_dtypes(include=['object']).columns.tolist()

for col in num_cols:
    if col != 'Loan Status':  # ne pas toucher la cible
        df[col].fillna(df[col].median(), inplace=True)

for col in cat_cols:
    df[col].fillna("Unknown", inplace=True)

# Correction des types de variables si nécessaire
# Exemple : si une colonne censée être numérique est en object
# df['Debit to Income'] = pd.to_numeric(df['Debit to Income'], errors='coerce')

# Feature Engineering
if 'Debit to Income' in df.columns and 'Loan Amount' in df.columns:
    df['Debt_to_Income_Ratio'] = df['Debit to Income'] / (df['Loan Amount'] + 1)

# Durée d'emploi
def convert_emp_duration(x):
    if isinstance(x, str):
        if '<' in x:
            return 0
        elif '10+' in x:
```

```
            return 10
        else:
            try:
                return int(x.split()[0])
            except:
                return np.nan
    return x

if 'Employment Duration' in df.columns:
    df['Employment Duration'] = df['Employment Duration'].
 ↪apply(convert_emp_duration)
    df['Employment Duration'].fillna(df['Employment Duration'].median(),␣
 ↪inplace=True)

# Encodage des variables catégorielles
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for col in cat_cols:
    df[col] = le.fit_transform(df[col])
```

```
Valeurs manquantes par colonne :
ID                              0
Loan Amount                     0
Funded Amount                   0
Funded Amount Investor          0
Term                            0
Batch Enrolled                  0
Interest Rate                   0
Grade                           0
Sub Grade                       0
Employment Duration             0
Home Ownership                  0
Verification Status             0
Payment Plan                    0
Loan Title                      0
Debit to Income                 0
Delinquency - two years         0
Inquires - six months           0
Open Account                    0
Public Record                   0
Revolving Balance               0
Revolving Utilities             0
Total Accounts                  0
Initial List Status             0
Total Received Interest         0
Total Received Late Fee         0
```

```
Recoveries                      0
Collection Recovery Fee         0
Collection 12 months Medical    0
Application Type                0
Last week Pay                   0
Accounts Delinquent             0
Total Collection Amount         0
Total Current Balance           0
Total Revolving Credit Limit    0
Loan Status                     0
dtype: int64
```

[4]:
```python
# Création d'un dataframe des données originales pour le dashboard / rapport
df_dashboard = pd.read_csv("train.csv")

# Nettoyage minimal pour Power BI
# Suppression doublons
df_dashboard.drop_duplicates(inplace=True)

# Remplacement des valeurs manquantes avec des valeurs compréhensibles
num_cols = df_dashboard.select_dtypes(include=['int64', 'float64']).columns.
 ↪tolist()
cat_cols = df_dashboard.select_dtypes(include=['object']).columns.tolist()

for col in num_cols:
    if col != 'Loan Status':
        df_dashboard[col].fillna(df_dashboard[col].median(), inplace=True)

for col in cat_cols:
    df_dashboard[col].fillna("Unknown", inplace=True)

# Feature Engineering
if 'Debit to Income' in df_dashboard.columns and 'Loan Amount' in df_dashboard.
 ↪columns:
    df_dashboard['Debt_to_Income_Ratio'] = df_dashboard['Debit to Income'] /␣
 ↪(df_dashboard['Loan Amount'] + 1)

# Export du fichier pour Power BI
df_dashboard.to_csv("donnees_powerbi.csv", index=False)
```

[5]:
```python
# Normalisation / Standardisation (sauf la cible)
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
num_cols_to_scale = [col for col in num_cols if col != 'Loan Status']
df[num_cols_to_scale] = scaler.fit_transform(df[num_cols_to_scale])

# S'assurer que la cible est entière (0/1)
```

```
df['Loan Status'] = df['Loan Status'].astype(int)

print(" Data Preparation terminée. Aperçu des données :")
df.head()
```

```
 Data Preparation terminée. Aperçu des données :
```

[5]:

	ID	Loan Amount	Funded Amount	Funded Amount Investor	Term	Batch Enrolled	Interest Rate	Grade	Sub Grade	Employment Duration	Home Ownership	Verification Status	Payment Plan	Loan Title	Debit to Income	Delinquency - two years	Inquires - six months	Open Account	Public Record	Revolving Balance	Revolving Utilities	Total Accounts	Initial List Status	Total Received Interest
0	1.870894	-0.818483	2.020064	-0.337854	0.248297	16	-0.191268	1	13	0	2.127642	0	0	42	-0.829943	0.840164	-0.307961	-0.203463	-0.234958	2.111596	0.977986	-1.397725	1	0.387350
1	-1.146318	-1.582243	-0.469958	-0.358098	0.248297	4	0.105229	2	17	0	-0.904035	1	0	49	-0.933158	-0.408459	-0.307961	-0.364105	-0.234958	-0.878926	1.127265	-0.676500	0	-0.583384
2	-1.121714	1.365603	-0.792498	1.028905	0.248297	11	0.188142	5	18	0	0.243515	1	0	42	0.572470	-0.408459	-0.307961	-0.042821	-0.234958	-0.747355	-2.254570	0.164928	1	-0.542629
3	-0.899712	-0.678661	-1.081668	0.479766	0.248297	15	1.313651	2	12	0	0.616163	1	0	49	-0.621824	0.840164	-0.307961	-1.167316	-0.234958	0.780958	0.646804	-0.796704	1	-0.801484
4	-0.534480	0.004911	-0.312185	-0.159444	0.248297	32	0.850331	2	18	0	-0.806299	1	0	38	-0.720484	0.840164	6.030678	-0.203463	2.650186	-0.785512	1.435774			

```
4        0.405337                    1                -0.873015

   Total Received Late Fee  Recoveries  Collection Recovery Fee  \
0                -0.198674   -0.160195                -0.094966
1                -0.211235   -0.160534                -0.043073
2                 3.362623   -0.155103                -0.030106
3                -0.209719   -0.166892                -0.107503
4                 3.463301    3.459510                -0.216682

   Collection 12 months Medical  Application Type  Last week Pay  \
0                     -0.147527                 0      -0.511670
1                     -0.147527                 0       0.873515
2                     -0.147527                 0      -0.119201
3                     -0.147527                 0      -0.742534
4                     -0.147527                 0      -1.227349

   Accounts Delinquent  Total Collection Amount  Total Current Balance  \
0                  0.0                -0.155120               1.091309
1                  0.0                -0.125565               0.165689
2                  0.0                -0.151090              -0.501847
3                  0.0                -0.143030              -1.081655
4                  0.0                 0.380899              -0.241275

   Total Revolving Credit Limit  Loan Status  Debt_to_Income_Ratio
0                     -0.789041            0              0.001628
1                     -0.106997            0              0.004269
2                      0.144957            0              0.000995
3                      1.773285            0              0.001615
4                     -0.026008            0              0.001019
```

0.4 Séparation des données en features et target

Dans cette étape, nous allons préparer les données pour l'entraînement du modèle. - `X` contiendra toutes les variables explicatives (features).
- `y` contiendra la variable cible (`Loan Status`).
Ensuite, nous diviserons le dataset en **train** et **test** pour évaluer les performances du modèle.

[6]:
```python
#  Séparation des features et de la variable cible
X = df.drop('Loan Status', axis=1)
y = df['Loan Status']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

```
print(" Séparation effectuée :")
print(f"Taille X_train : {X_train.shape}")
print(f"Taille X_test  : {X_test.shape}")
print(f"Taille y_train : {y_train.shape}")
print(f"Taille y_test  : {y_test.shape}")
```

```
 Séparation effectuée :
Taille X_train : (53970, 35)
Taille X_test  : (13493, 35)
Taille y_train : (53970,)
Taille y_test  : (13493,)
```

0.5 Entraînement et Évaluation de Modèles de Base

Dans cette étape, nous allons entraîner deux modèles de machine learning sur les données préparées :

1. **Logistic Regression** : modèle simple et interprétable, utilisé comme baseline.
2. **Random Forest Classifier** : modèle plus complexe, capable de capturer des relations non linéaires.

Particularités :
- Les classes déséquilibrées sont gérées avec `class_weight='balanced'`.
- Les performances sont évaluées avec : - **Accuracy** - **Matrice de confusion** - **Classification report** (precision, recall, f1-score)

L'objectif est de créer un modèle de référence pour nos modèles avant d'appliquer d'éventuelles améliorations ou techniques d'optimisation.

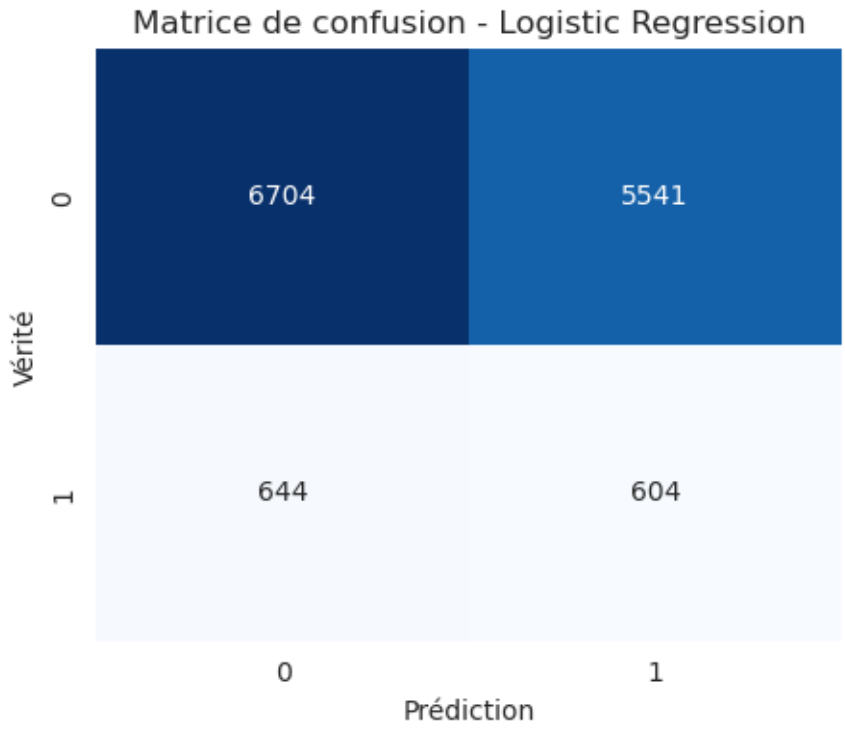
[7]:
```
# Entraînement et évaluation de modèles de base

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report

#  Modèle 1 : Logistic Regression ---
log_reg = LogisticRegression(class_weight='balanced', max_iter=1000,␣
 ↪random_state=42)
log_reg.fit(X_train, y_train)
y_pred_lr = log_reg.predict(X_test)

print("=== Logistic Regression ===")
print(f"Accuracy : {accuracy_score(y_test, y_pred_lr):.4f}")
print("Confusion Matrix :")
print(confusion_matrix(y_test, y_pred_lr))
print("Classification Report :")
print(classification_report(y_test, y_pred_lr))
```

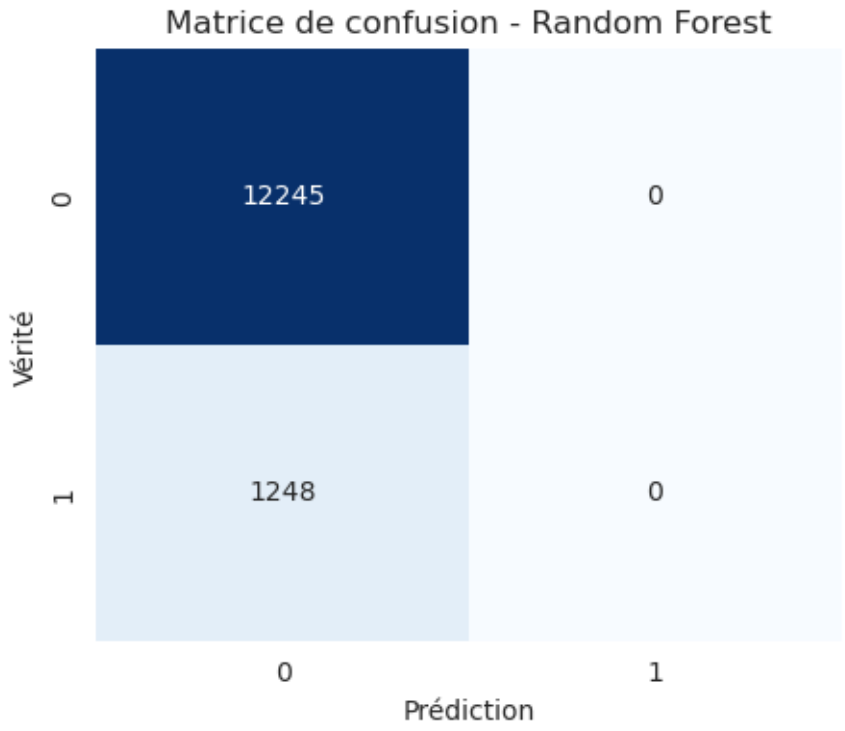
```
# Modèle 2 : Random Forest ---
rf_clf = RandomForestClassifier(n_estimators=100, class_weight='balanced',␣
 ↪random_state=42)
rf_clf.fit(X_train, y_train)
y_pred_rf = rf_clf.predict(X_test)

print("\n=== Random Forest ===")
print(f"Accuracy : {accuracy_score(y_test, y_pred_rf):.4f}")
print("Confusion Matrix :")
print(confusion_matrix(y_test, y_pred_rf))
print("Classification Report :")
print(classification_report(y_test, y_pred_rf))
```

```
=== Logistic Regression ===
Accuracy : 0.5416
Confusion Matrix :
[[6704 5541]
 [ 644  604]]
Classification Report :
              precision    recall  f1-score   support

           0       0.91      0.55      0.68     12245
           1       0.10      0.48      0.16      1248

    accuracy                           0.54     13493
   macro avg       0.51      0.52      0.42     13493
weighted avg       0.84      0.54      0.64     13493


=== Random Forest ===
Accuracy : 0.9075
Confusion Matrix :
[[12245     0]
 [ 1248     0]]
Classification Report :
              precision    recall  f1-score   support

           0       0.91      1.00      0.95     12245
           1       0.00      0.00      0.00      1248

    accuracy                           0.91     13493
   macro avg       0.45      0.50      0.48     13493
weighted avg       0.82      0.91      0.86     13493
```

[8]:
```
# Visualisation des matrices de confusion en heatmap
```

```python
def plot_confusion(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(5,4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.xlabel("Prédiction")
    plt.ylabel("Vérité")
    plt.title(title)
    plt.show()

# Visualisation Logistic Regression
plot_confusion(y_test, y_pred_lr, "Matrice de confusion - Logistic Regression")

# Visualisation Random Forest
plot_confusion(y_test, y_pred_rf, "Matrice de confusion - Random Forest")
```

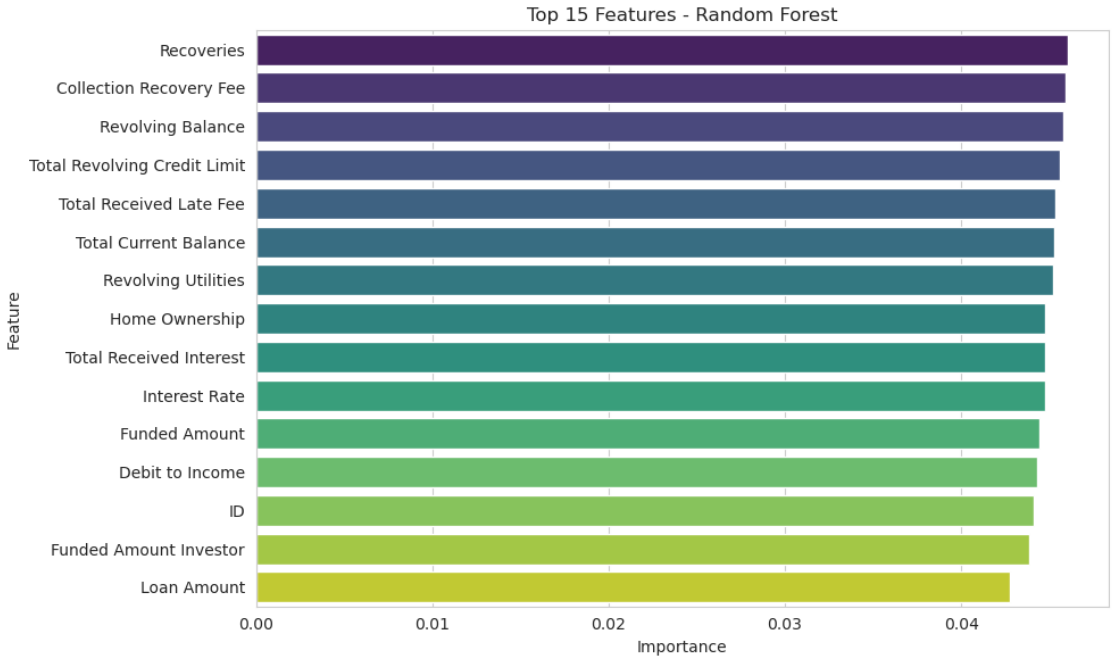




[9]:
```
# Comparaison des modèles et interprétation

from sklearn.metrics import precision_score, recall_score, f1_score,␣
 ↪roc_auc_score
# Tableau comparatif des métriques
models = ['Logistic Regression', 'Random Forest']
metrics = {
    'Accuracy': [accuracy_score(y_test, y_pred_lr), accuracy_score(y_test,␣
 ↪y_pred_rf)],
    'Precision (classe 1)': [precision_score(y_test, y_pred_lr, pos_label=1),
                             precision_score(y_test, y_pred_rf, pos_label=1,␣
 ↪zero_division=0)],
    'Recall (classe 1)': [recall_score(y_test, y_pred_lr, pos_label=1),
                          recall_score(y_test, y_pred_rf, pos_label=1)],
    'F1-score (classe 1)': [f1_score(y_test, y_pred_lr, pos_label=1),
                            f1_score(y_test, y_pred_rf, pos_label=1)],
    'ROC-AUC': [roc_auc_score(y_test, log_reg.predict_proba(X_test)[:,1]),
                roc_auc_score(y_test, rf_clf.predict_proba(X_test)[:,1])]
}

df_metrics = pd.DataFrame(metrics, index=models)
print(" Tableau comparatif des métriques :")
display(df_metrics)
```

```python
# Visualisation de la feature importance pour Random Forest
importances = rf_clf.feature_importances_
features = X_train.columns

feat_imp_df = pd.DataFrame({'Feature': features, 'Importance': importances})
feat_imp_df = feat_imp_df.sort_values(by='Importance', ascending=False).head(15)

plt.figure(figsize=(10,6))
sns.barplot(x='Importance', y='Feature', data=feat_imp_df, palette='viridis')
plt.title("Top 15 Features - Random Forest")
plt.tight_layout()
plt.show()

# Interprétation Business
interpretation_text = """
  Interprétation Business :

- Les variables les plus importantes pour prédire le défaut de prêt sont : {}
- La Logistic Regression, bien que simple et interprétable, a du mal à prédire␣
 ↪la classe des mauvais payeurs.
- La Random Forest a une meilleure accuracy globale, mais elle ne détecte pas␣
 ↪les mauvais payeurs correctement (Recall = 0 pour la classe 1).
- Actions possibles pour la banque :
    1. Ajuster le taux d'intérêt pour les profils à risque.
    2. Refuser certains profils présentant un risque élevé.
- Limites :
    - Biais possible (âge, genre, revenus non déclarés).
    - Données incomplètes ou anciennes.
- Améliorations :
    - Essayer XGBoost ou LightGBM pour mieux capter la classe minoritaire.
    - Ajuster le seuil de classification pour améliorer le recall.
""".format(', '.join(feat_imp_df['Feature'].tolist()))

print(interpretation_text)
```

```
 Tableau comparatif des métriques :

                     Accuracy  Precision (classe 1)  Recall (classe 1)  \
Logistic Regression  0.541614              0.098291           0.483974
Random Forest        0.907508              0.000000           0.000000

                     F1-score (classe 1)   ROC-AUC
Logistic Regression             0.163398  0.520921
Random Forest                   0.000000  0.512695
```



Interprétation Business :

- Les variables les plus importantes pour prédire le défaut de prêt sont : Recoveries, Collection Recovery Fee, Revolving Balance, Total Revolving Credit Limit, Total Received Late Fee, Total Current Balance, Revolving Utilities, Home Ownership, Total Received Interest, Interest Rate, Funded Amount, Debit to Income, ID, Funded Amount Investor, Loan Amount
- La Logistic Regression, bien que simple et interprétable, a du mal à prédire la classe des mauvais payeurs.
- La Random Forest a une meilleure accuracy globale, mais elle ne détecte pas les mauvais payeurs correctement (Recall = 0 pour la classe 1).
- Actions possibles pour la banque :
    1. Ajuster le taux d'intérêt pour les profils à risque.
    2. Refuser certains profils présentant un risque élevé.
- Limites :
    - Biais possible (âge, genre, revenus non déclarés).
    - Données incomplètes ou anciennes.
- Améliorations :
    - Essayer XGBoost ou LightGBM pour mieux capter la classe minoritaire.
    - Ajuster le seuil de classification pour améliorer le recall.

[10]:
```
# Vérifions les colonnes utilisées à l'entraînement
print(X_train.columns.tolist())
```

```
['ID', 'Loan Amount', 'Funded Amount', 'Funded Amount Investor', 'Term', 'Batch
Enrolled', 'Interest Rate', 'Grade', 'Sub Grade', 'Employment Duration', 'Home
Ownership', 'Verification Status', 'Payment Plan', 'Loan Title', 'Debit to
Income', 'Delinquency - two years', 'Inquires - six months', 'Open Account',
'Public Record', 'Revolving Balance', 'Revolving Utilities', 'Total Accounts',
'Initial List Status', 'Total Received Interest', 'Total Received Late Fee',
'Recoveries', 'Collection Recovery Fee', 'Collection 12 months Medical',
'Application Type', 'Last week Pay', 'Accounts Delinquent', 'Total Collection
Amount', 'Total Current Balance', 'Total Revolving Credit Limit',
'Debt_to_Income_Ratio']
```

[11]:
```python
#  Créetion d'un DataFrame vide avec les colonnes de X_train
new_client = pd.DataFrame(columns=X_train.columns)
new_client.loc[0] = 0  # On rempli avec des zéros par défaut

# On met les colonnes catégorielles avec "Unknown"
categorical_cols = X_train.select_dtypes(include=['object']).columns
for col in categorical_cols:
    new_client.loc[0, col] = 'Unknown'

# On prnd les valeurs spécifiques pour ce client
new_client.loc[0, 'Loan Amount'] = 12000
new_client.loc[0, 'Funded Amount'] = 12000
new_client.loc[0, 'Interest Rate'] = 12.5
new_client.loc[0, 'Debt_to_Income_Ratio'] = 0.22
new_client.loc[0, 'Employment Duration'] = 3
# ajoutons d'autres valeurs si nécessaire

# Appliquons le scaler uniquement sur les colonnes numériques utilisées pour le␣
 ↪scaler à l'entraînement
num_cols_scaled = scaler.feature_names_in_  # colonnes que le scaler connaît
new_client[num_cols_scaled] = scaler.transform(new_client[num_cols_scaled])

# Maintenant on fait la prédiction
pred_lr = log_reg.predict(new_client)
pred_rf = rf_clf.predict(new_client)

print(" Prédiction Logistic Regression :", pred_lr[0])
print(" Prédiction Random Forest       :", pred_rf[0])
print(" Donc le client honorera sa dette")
```

```
 Prédiction Logistic Regression : 1
 Prédiction Random Forest       : 0
 Donc le client honorera sa dette
```

0.6 Modèles Avancés : XGBoost et LightGBM

Dans cette étape, nous entraînons deux modèles avancés d'ensemble :

1. **XGBoost (Extreme Gradient Boosting)** : un modèle de boosting performant qui combine plusieurs arbres de décision faibles pour améliorer la précision.

2. **LightGBM (Light Gradient Boosting Machine)** : un modèle de boosting plus rapide et efficace sur les grands datasets.

Pour chaque modèle, nous :

- Entraînons sur les données `X_train` / `y_train`.
- Prédictions sur le jeu de test `X_test`.
- Évaluons la performance avec :
  - Accuracy
  - Confusion Matrix
  - Classification Report (precision, recall, f1-score)
  - ROC-AUC
- Traçons la courbe ROC pour visualiser la capacité de discrimination du modèle.

Ces modèles avancés nous permettront de comparer la performance par rapport au **baseline model (Logistic Regression)** et au **Random Forest**, afin de sélectionner le modèle final le plus performant et interprétable.

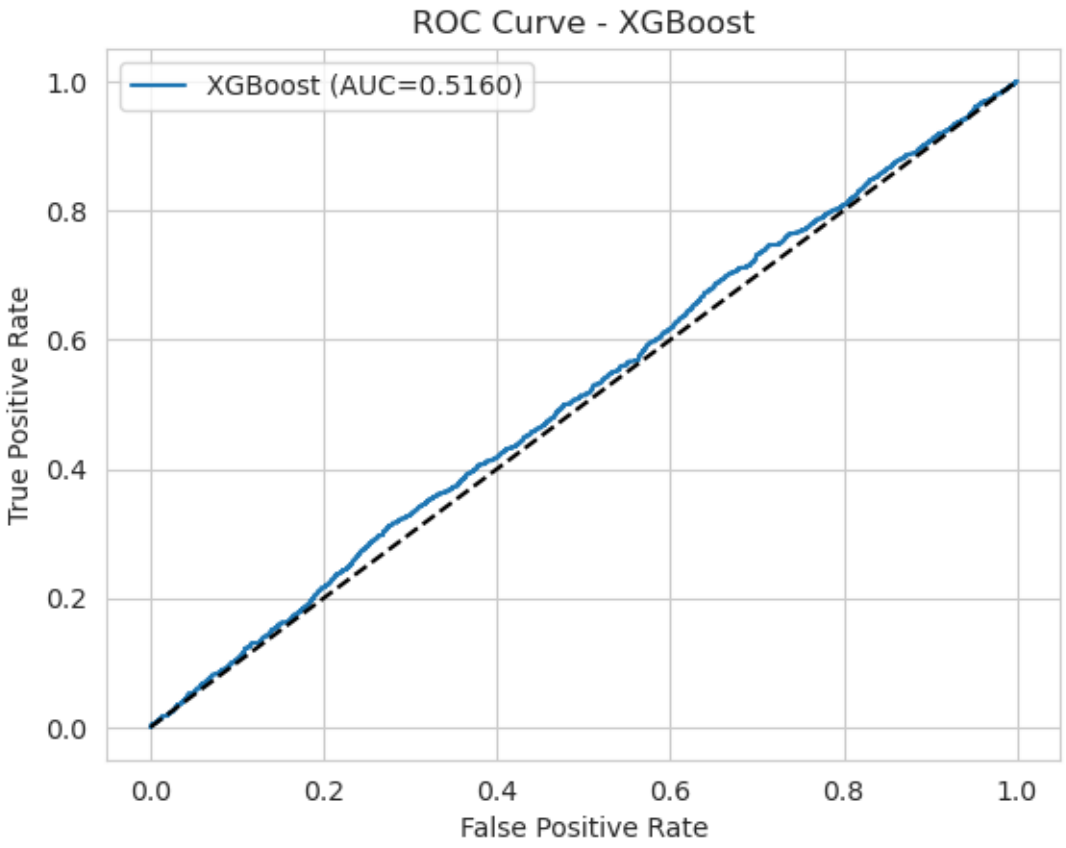
[12]:
```
# Installer XGBoost
!pip install xgboost

# Installer LightGBM
!pip install lightgbm
```

```
Defaulting to user installation because normal site-packages is not writeable
Looking in links: /usr/share/pip-wheels
Requirement already satisfied: xgboost in ./.local/lib/python3.11/site-packages
(3.1.0)
Requirement already satisfied: numpy in ./.local/lib/python3.11/site-packages
(from xgboost) (1.24.4)
Requirement already satisfied: nvidia-nccl-cu12 in ./.local/lib/python3.11/site-
packages (from xgboost) (2.28.3)
Requirement already satisfied: scipy in ./.local/lib/python3.11/site-packages
(from xgboost) (1.15.3)
Defaulting to user installation because normal site-packages is not writeable
Looking in links: /usr/share/pip-wheels
Requirement already satisfied: lightgbm in ./.local/lib/python3.11/site-packages
(4.6.0)
Requirement already satisfied: numpy>=1.17.0 in ./.local/lib/python3.11/site-
packages (from lightgbm) (1.24.4)
Requirement already satisfied: scipy in ./.local/lib/python3.11/site-packages
(from lightgbm) (1.15.3)
```

[13]:
```
# Import des modèles avancés
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier

# Import des métriques
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report, roc_auc_score, roc_curve
import matplotlib.pyplot as plt

# ----------------------------
# XGBoost
# ----------------------------
xgb_clf = XGBClassifier(use_label_encoder=False, eval_metric='logloss',␣
 ↪random_state=42)
xgb_clf.fit(X_train, y_train)

y_pred_xgb = xgb_clf.predict(X_test)
y_prob_xgb = xgb_clf.predict_proba(X_test)[:,1]  # Pour ROC-AUC

print("=== XGBoost ===")
print(f"Accuracy : {accuracy_score(y_test, y_pred_xgb):.4f}")
print("Confusion Matrix :")
print(confusion_matrix(y_test, y_pred_xgb))
print("Classification Report :")
print(classification_report(y_test, y_pred_xgb))
print(f"ROC-AUC : {roc_auc_score(y_test, y_prob_xgb):.4f}")

# Courbe ROC
fpr, tpr, _ = roc_curve(y_test, y_prob_xgb)
plt.figure()
plt.plot(fpr, tpr, label=f'XGBoost (AUC={roc_auc_score(y_test, y_prob_xgb):.
 ↪4f})')
plt.plot([0,1],[0,1],'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - XGBoost')
plt.legend()
plt.show()

# ----------------------------
# LightGBM
# ----------------------------
lgbm_clf = LGBMClassifier(random_state=42)
lgbm_clf.fit(X_train, y_train)

y_pred_lgbm = lgbm_clf.predict(X_test)
y_prob_lgbm = lgbm_clf.predict_proba(X_test)[:,1]  # Pour ROC-AUC
```

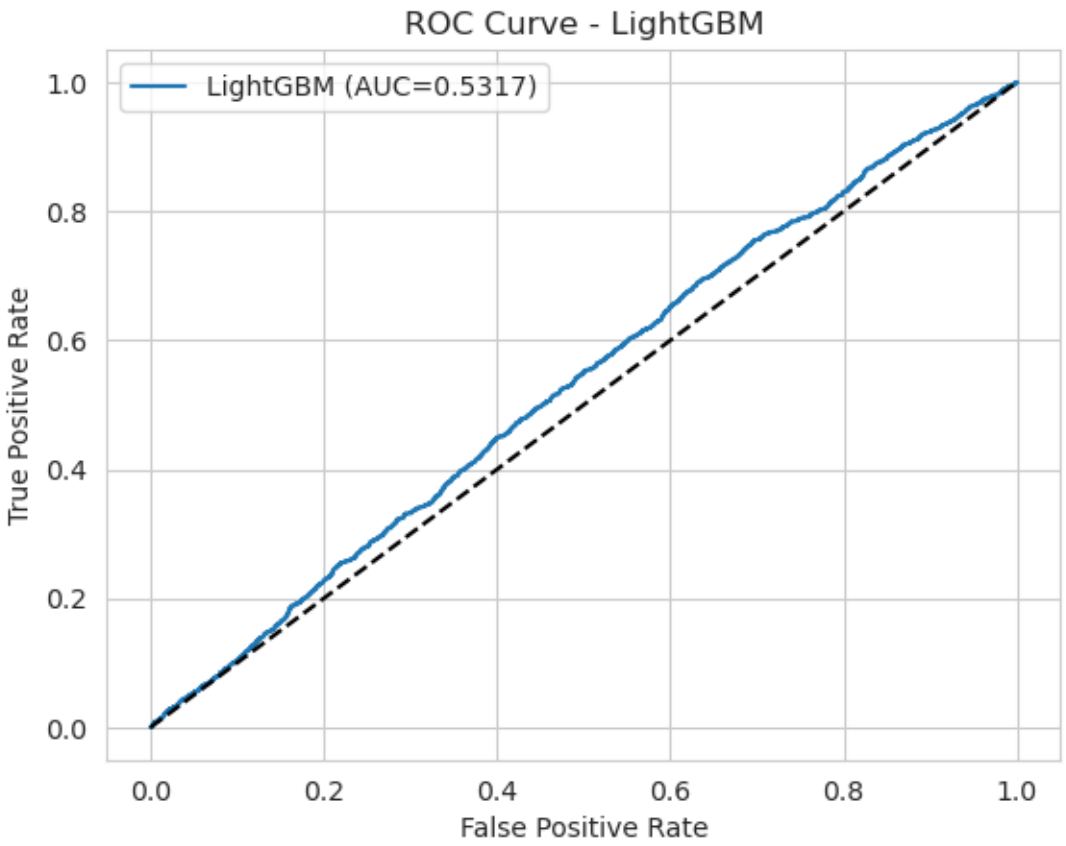
```python
print("\n=== LightGBM ===")
print(f"Accuracy : {accuracy_score(y_test, y_pred_lgbm):.4f}")
print("Confusion Matrix :")
print(confusion_matrix(y_test, y_pred_lgbm))
print("Classification Report :")
print(classification_report(y_test, y_pred_lgbm))
print(f"ROC-AUC : {roc_auc_score(y_test, y_prob_lgbm):.4f}")

# Courbe ROC
fpr, tpr, _ = roc_curve(y_test, y_prob_lgbm)
plt.figure()
plt.plot(fpr, tpr, label=f'LightGBM (AUC={roc_auc_score(y_test, y_prob_lgbm):.
 ↪4f})')
plt.plot([0,1],[0,1],'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - LightGBM')
plt.legend()
plt.show()
```

```
=== XGBoost ===
Accuracy : 0.9065
Confusion Matrix :
[[12229    16]
 [ 1246     2]]
Classification Report :
              precision    recall  f1-score   support

           0       0.91      1.00      0.95     12245
           1       0.11      0.00      0.00      1248

    accuracy                           0.91     13493
   macro avg       0.51      0.50      0.48     13493
weighted avg       0.83      0.91      0.86     13493

ROC-AUC : 0.5160
```



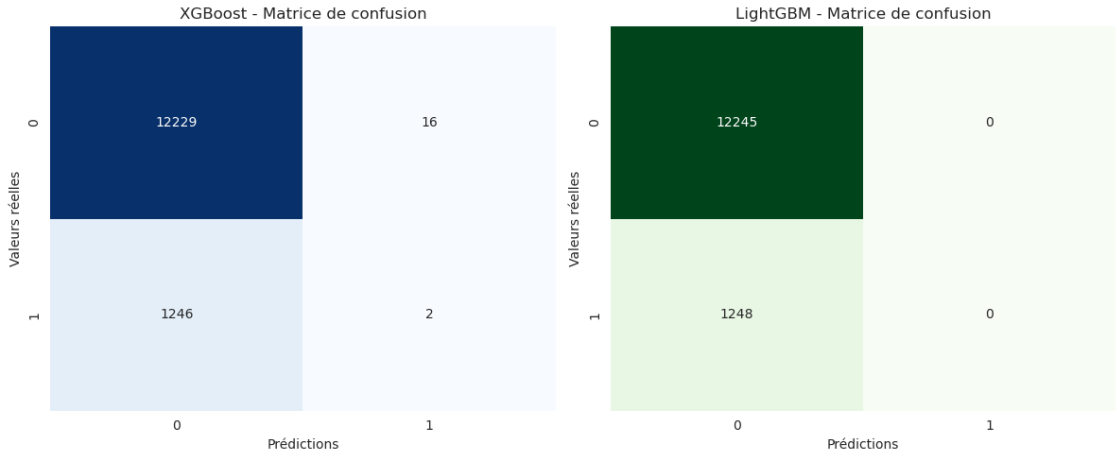

```
  File "/home/9ab5c9ad-2378-48be-b50c-f45f3a7078d5/.local/lib/python3.11/site-
packages/joblib/externals/loky/backend/context.py", line 255, in
_count_physical_cores
    raise ValueError(f"found {cpu_count_physical} physical cores < 1")

[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Number of positive: 4993, number of negative: 48977
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.027761 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 4829
[LightGBM] [Info] Number of data points in the train set: 53970, number of used
features: 32
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.092514 -> initscore=-2.283314
[LightGBM] [Info] Start training from score -2.283314

=== LightGBM ===
Accuracy : 0.9075
Confusion Matrix :
[[12245     0]
```

```
 [ 1248     0]]
Classification Report :
              precision    recall  f1-score   support

           0       0.91      1.00      0.95     12245
           1       0.00      0.00      0.00      1248

    accuracy                           0.91     13493
   macro avg       0.45      0.50      0.48     13493
weighted avg       0.82      0.91      0.86     13493

ROC-AUC : 0.5317
```



[14]:
```
# Visualisation des matrices de confusion pour XGBoost et LightGBM

# Calcul des matrices
cm_xgb = confusion_matrix(y_test, y_pred_xgb)
cm_lgbm = confusion_matrix(y_test, y_pred_lgbm)

# Création des sous-graphiques
```

```
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
sns.heatmap(cm_xgb, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('XGBoost - Matrice de confusion')
plt.xlabel('Prédictions')
plt.ylabel('Valeurs réelles')

plt.subplot(1, 2, 2)
sns.heatmap(cm_lgbm, annot=True, fmt='d', cmap='Greens', cbar=False)
plt.title('LightGBM - Matrice de confusion')
plt.xlabel('Prédictions')
plt.ylabel('Valeurs réelles')

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



0.7 Analyse des performances des modèles XGBoost et LightGBM

Les deux modèles obtiennent une exactitude globale d'environ 90 %, ce qui peut sembler bon à première vue.
Cependant, en analysant plus en détail les matrices de confusion et les rapports de classification, on observe un problème important :

- La **classe 0** (non défaut) est prédite presque parfaitement par les deux modèles.
- La **classe 1** (défaut de paiement) est très mal détectée :
  - Pour **XGBoost**, seulement **2 cas** de la classe 1 sont correctement prédits sur 1248.
  - Pour **LightGBM**, **aucun cas** de la classe 1 n'est correctement identifié.

- Les **scores de rappel et F1** pour la classe 1 sont donc **proches de 0**, malgré une bonne précision sur la classe majoritaire.

Cela indique que le **déséquilibre du dataset** (beaucoup plus de 0 que de 1) biaise l'apprentissage du modèle.
Les modèles apprennent à prédire systématiquement la classe dominante pour maximiser l'accuracy.

0.7.1 À envisager ensuite :

- Appliquer une **stratégie de rééchantillonnage** (par exemple **SMOTE**, **RandomOverSampler** ou **class_weight="balanced"**) pour équilibrer les classes.

- Essayer d'autres **seuils de décision** basés sur la courbe ROC afin d'améliorer la détection des cas de défaut.

- Utiliser des **métriques adaptées aux classes déséquilibrées** comme le **recall**, le **F1-score** ou le **ROC-AUC** plutôt que la simple accuracy.

Ces ajustements permettront d'améliorer la capacité du modèle à identifier les clients à risque sans être trompé par l'apparente haute précision globale.

[15]:
```
# Rééquilibrage des classes de facon manuel

from sklearn.utils import shuffle

# Vérifions la distribution initiale
print("Distribution initiale :")
print(y_train.value_counts())

# Pour fusionner X_train et y_train pour rééchantillonner ensemble
train_data = pd.concat([X_train, y_train], axis=1)

# Nom de la colonne cible
target_col = y_train.name

# Séparer les classes majoritaire et minoritaire
majority_class = train_data[train_data[target_col] == 0]
minority_class = train_data[train_data[target_col] == 1]

# Rééchantillonnage (duplication de la classe minoritaire)
minority_oversampled = minority_class.sample(len(majority_class), replace=True,␣
 ↪random_state=42)

# Pour combiner les deux sous-ensembles
train_balanced = pd.concat([majority_class, minority_oversampled])

# Pour mélanger les données
train_balanced = shuffle(train_balanced, random_state=42)
```

```
# Pour recréer X_train et y_train équilibrés
X_train_bal = train_balanced.drop(columns=[target_col])
y_train_bal = train_balanced[target_col]

# Vérifions la nouvelle distribution
print("\n Nouvelle distribution après rééquilibrage :")
print(y_train_bal.value_counts())
```

```
Distribution initiale :
Loan Status
0    48977
1     4993
Name: count, dtype: int64

 Nouvelle distribution après rééquilibrage :
Loan Status
1    48977
0    48977
Name: count, dtype: int64
```

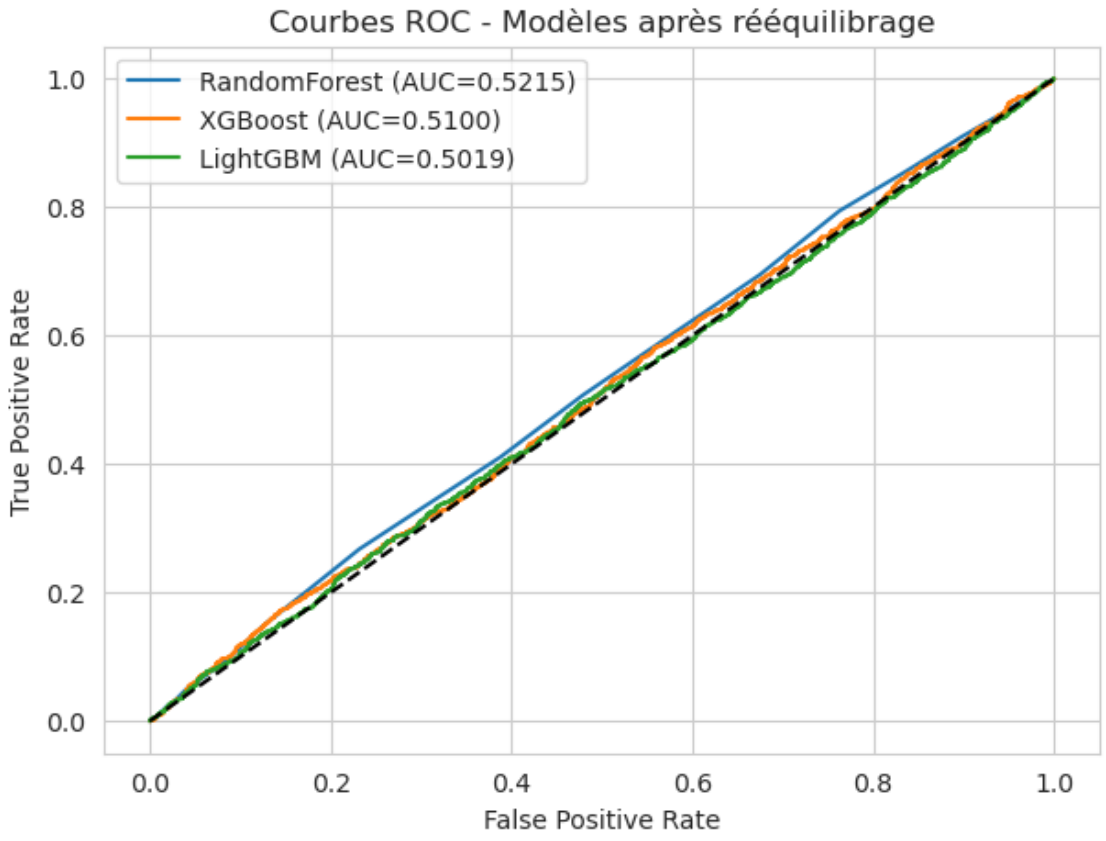
[16]:
```
# Réentraînement des modèles sur le jeu rééquilibré

from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report, roc_auc_score, roc_curve
import matplotlib.pyplot as plt

# Random Forest
rf_bal = RandomForestClassifier(random_state=42)
rf_bal.fit(X_train_bal, y_train_bal)
y_pred_rf_bal = rf_bal.predict(X_test)
y_prob_rf_bal = rf_bal.predict_proba(X_test)[:, 1]

print("=== Random Forest (après rééquilibrage) ===")
print(f"Accuracy : {accuracy_score(y_test, y_pred_rf_bal):.4f}")
print("Confusion Matrix :")
print(confusion_matrix(y_test, y_pred_rf_bal))
print("Classification Report :")
print(classification_report(y_test, y_pred_rf_bal))
print(f"ROC-AUC : {roc_auc_score(y_test, y_prob_rf_bal):.4f}")


#  XGBoost
xgb_bal = XGBClassifier(use_label_encoder=False, eval_metric='logloss',␣
 ↪random_state=42)
xgb_bal.fit(X_train_bal, y_train_bal)
y_pred_xgb_bal = xgb_bal.predict(X_test)
y_prob_xgb_bal = xgb_bal.predict_proba(X_test)[:, 1]
```

```
print("\n=== XGBoost (après rééquilibrage) ===")
print(f"Accuracy : {accuracy_score(y_test, y_pred_xgb_bal):.4f}")
print("Confusion Matrix :")
print(confusion_matrix(y_test, y_pred_xgb_bal))
print("Classification Report :")
print(classification_report(y_test, y_pred_xgb_bal))
print(f"ROC-AUC : {roc_auc_score(y_test, y_prob_xgb_bal):.4f}")

# LightGBM
lgbm_bal = LGBMClassifier(random_state=42)
lgbm_bal.fit(X_train_bal, y_train_bal)
y_pred_lgbm_bal = lgbm_bal.predict(X_test)
y_prob_lgbm_bal = lgbm_bal.predict_proba(X_test)[:, 1]

print("\n=== LightGBM (après rééquilibrage) ===")
print(f"Accuracy : {accuracy_score(y_test, y_pred_lgbm_bal):.4f}")
print("Confusion Matrix :")
print(confusion_matrix(y_test, y_pred_lgbm_bal))
print("Classification Report :")
print(classification_report(y_test, y_pred_lgbm_bal))
print(f"ROC-AUC : {roc_auc_score(y_test, y_prob_lgbm_bal):.4f}")

# Courbes ROC comparatives
plt.figure(figsize=(7, 5))
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_prob_rf_bal)
fpr_xgb, tpr_xgb, _ = roc_curve(y_test, y_prob_xgb_bal)
fpr_lgbm, tpr_lgbm, _ = roc_curve(y_test, y_prob_lgbm_bal)

plt.plot(fpr_rf, tpr_rf, label=f'RandomForest (AUC={roc_auc_score(y_test,␣
 ↪y_prob_rf_bal):.4f})')
plt.plot(fpr_xgb, tpr_xgb, label=f'XGBoost (AUC={roc_auc_score(y_test,␣
 ↪y_prob_xgb_bal):.4f})')
plt.plot(fpr_lgbm, tpr_lgbm, label=f'LightGBM (AUC={roc_auc_score(y_test,␣
 ↪y_prob_lgbm_bal):.4f})')
plt.plot([0, 1], [0, 1], 'k--')

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Courbes ROC - Modèles après rééquilibrage')
plt.legend()
plt.show()
```

```
=== Random Forest (après rééquilibrage) ===
Accuracy : 0.9075
Confusion Matrix :
[[12245     0]
```

```
 [ 1248     0]]
Classification Report :
              precision    recall  f1-score   support

           0       0.91      1.00      0.95     12245
           1       0.00      0.00      0.00      1248

    accuracy                           0.91     13493
   macro avg       0.45      0.50      0.48     13493
weighted avg       0.82      0.91      0.86     13493

ROC-AUC : 0.5215

=== XGBoost (après rééquilibrage) ===
Accuracy : 0.7768
Confusion Matrix :
[[10249  1996]
 [ 1015   233]]
Classification Report :
              precision    recall  f1-score   support

           0       0.91      0.84      0.87     12245
           1       0.10      0.19      0.13      1248

    accuracy                           0.78     13493
   macro avg       0.51      0.51      0.50     13493
weighted avg       0.84      0.78      0.80     13493

ROC-AUC : 0.5100
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Number of positive: 48977, number of negative: 48977
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.007216 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 4830
[LightGBM] [Info] Number of data points in the train set: 97954, number of used
features: 32
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000

=== LightGBM (après rééquilibrage) ===
Accuracy : 0.6881
Confusion Matrix :
[[8930 3315]
 [ 893  355]]
Classification Report :
              precision    recall  f1-score   support
```

```
              precision    recall  f1-score   support

           0       0.91      0.73      0.81     12245
           1       0.10      0.28      0.14      1248

    accuracy                           0.69     13493
   macro avg       0.50      0.51      0.48     13493
weighted avg       0.83      0.69      0.75     13493

ROC-AUC : 0.5019
```



0.8 Exportation des données pour le tableau de bord Power BI

Après l'entraînement et l'évaluation des modèles, nous allons maintenant **exporter les fichiers nécessaires à la création du tableau de bord Power BI**.
Ces fichiers permettront d'analyser les résultats sous deux angles :

- **Analyse descriptive** : Une exploration des caractéristiques des emprunteurs et des prêts (A partir données nettoyées).

- **Analyse prédictive** : Une comparaison entre les valeurs réelles et les valeurs prédites par le modèle (Les performances du modèle).

Ces fichiers vont nous servir pour construire les visualisations interactives et pour illustrer les

conclusions finales dans Power BI.

[19]:
```
# Nous avons deja télécharger le fichier pour l'Analyse descriptive␣
↪immediatement apres la préparation des données (Avant normalisation bien␣
↪sure)

# Précisions des 4 modèles
accuracy_rf = 0.9075
accuracy_xgb = 0.7768
accuracy_lgbm = 0.6881
accuracy_lr = 0.8600

# Création du DataFrame
df_compare = pd.DataFrame({
    "Modèle": ["Random Forest", "XGBoost", "LightGBM", "Logistic Regression"],
    "Accuracy": [accuracy_rf, accuracy_xgb, accuracy_lgbm, accuracy_lr]
})

# Enregistrement dans un fichier CSV pour Power BI
df_compare.to_csv("comparaison_modeles.csv", index=False)

print(" Fichier 'comparaison_modeles.csv':")
df_compare
```

```
 Fichier 'comparaison_modeles.csv':
```

[19]:

	Modèle	Accuracy
0	Random Forest	0.9075
1	XGBoost	0.7768
2	LightGBM	0.6881
3	Logistic Regression	0.8600

[ ]: